


# ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

Aplicación Android Workout Routines.

Autor: Ronald Sandoval Sandoval

Tutor: José Javier Astrain Escola

Pamplona, a 26 de abril de 2013

# ÍNDICE

# 1. INTRODUCCIÓN

## 1.1. Problema a resolver

En los centros de entrenamiento corporal, comúnmente llamados gimnasios, los entrenadores suelen asignar rutinas (tablas) de entrenamientos a las personas que lo deseen, ya que el objetivo primordial es mejorar la salud y bienestar de cada uno de sus clientes.

Estas tablas están constituidas por ejercicios que estimulan la gran mayoría de los grupos musculares, teniendo en cuenta un equilibrio que balancee la carga de trabajo durante el entrenamiento.

Estas guías suelen darse en una hoja impresa y con el objetivo de que cada persona la use para incrementar su masa muscular, bajar de peso o simplemente tener un cuerpo sano y esbelto.

El problema a resolver es que las personas suelen perder estas hojas o simplemente no la siguen al pie de la letra, ya sea porque se le olvida la hoja en casa o se pierde en los archivadores de los gimnasios (*gym*).

Es por eso que he previsto la necesidad de hacer una aplicación para móviles, en concreto para el sistema operativo *Android*, ya que es la tecnología más usada por las personas por su relación calidad–precio y que los dispositivos móviles en estos días se han convertido en una herramienta más en el día a día de las personas. Además, las herramientas de desarrollo y librerías que Android aporta, son gratuitas. El objetivo es hacer una herramienta sencilla y portable, donde esta a su vez reemplace las hojas por técnicas táctiles, donde imite a la perfección las tablas de entrenamiento.

Con esto nos aseguramos de que las personas no olviden o extravíen sus rutinas y por ende tengan que improvisar, con el agravante de que todo el entrenamiento sea en vano.

Para ello se dotará de una interfaz muy sencilla y agradable, donde podrán almacenar sus ejercicios en una pequeña pero consistente y centralizada base de datos, dotado de menús y botones claros y fáciles de utilizar. Además, como es de uso diario, este proyecto debe ser íntegro y sobretodo duradero, para que el usuario pueda disfrutar de sus ventajas.

Mi aplicación será gratuita y estará disponible en Google Play, también conocido como Android Market, para facilitar las búsquedas y descargas. Además, estará disponible en dos idiomas, inglés y español.

El nombre que he dado a la aplicación es "**Workout Routines**". Este nombre se debe a que es llamativo y así puedo abarcar tanto el mercado inglés como el Hispanoamericano.

## 1.2. Estado del arte

Cuando las personas buscan en Google Play o Android Market, siempre se encuentran que muchas aplicaciones son gratuitas, pero con el inconveniente de que para poder utilizar toda la versión completa tendrás que descargarte la versión de pago.

No solo eso, muchos hispanohablantes no se manejan bien con el inglés y tampoco son muy buenos con las nuevas tecnologías, por lo que las aplicaciones que se descargan tienen muchos menús y herramientas que hay que configurar antes de usar cada aplicación. Además, algunas aplicaciones te obligan a registrarte en sus propias páginas, para así poder acceder a los recursos de la aplicación, teniendo que enviar un correo válido y luego tener que confirmar, lo cual no suele gustar a muchas personas por el temor de recibir correos Spam.

Otra particularidad, es que muchas aplicaciones no cuentan con todo lo que un usuario necesita a la hora de hacer o crear su propia tabla de entrenamientos, sino que tienen una ya fija, que no pueden modificar porque la propia aplicación se lo deniega. Aparte, suelen tener rutinas muy extensas o complicadas que el usuario se perdería a la hora de decidir cuál es la mejor para trabajar varios grupos musculares.

Otra desventaja de algunas aplicaciones como la de *Adidas*[1] o *Nike*[2], es que requieres casi en todo momento acceso a internet, lo cual crea problemas para muchas personas que no cuentan con una tarifa de datos o acceso a Wi-Fi en su lugar de entrenamiento. Además, suelen pecar de hacer muy compleja la aplicación, causando que el usuario se pierda navegando por los diferentes contenidos y no sepa cómo sacarles el mejor provecho.

Para poner un ejemplo, expongo a continuación las aplicaciones más solicitadas tanto en inglés como en español en Google Play, haciendo una comparación con sus respectivas ventajas o desventajas entre ellas y con respecto a la mía.

- ***JEFIT - Workout, Fitness, GymLog:*** Aplicación gratuita para el entrenamiento diario, versión más reducida, ya que hay versión de pago con un coste de 3,79€ y ambas versiones están solo en Ingles. Se caracterizan por tener más ejercicios, cuenta con gráficas que muestran el proceso y llevan un control *on-line* de los entrenamientos diarios. Lo que destaca a esta aplicación son los gráficos de control de progreso. Lo malo es que se requiere acceso a internet para poder usar algunas de sus funciones durante el entrenamiento.

- ***Gymrat: Workout Planner & Log:*** Calculo del *BMI* (Índice de Masa Muscular), y que está asociado con lo recomendado por la organización mundial de la salud. Es de pago 1,99€ y cuenta con más de 400 ejercicios y gráficas de progreso de excelente calidad.

[1] Aplicación disponible en: http://www.adidas.com/es/micoach/
[2] Aplicación disponible en: http://nikeplus.nike.com/plus/products/gps_app/

- ***Gym: Guía de Ejercicios:*** Esta versión está en español, cuenta con una versión gratuita y otra de pago. Tiene más de 100 ejercicios y cuenta con una calculadora de índice de masa corporal. Los gráficos que utiliza para mostrar cada entrenamiento son muy malos.
- ***Workout Trainer:*** Está solo en inglés, cuenta con un calendario que va marcando los días que se tiene que entrenar y cómo se debería hacer cada ejercicio, para ello cuenta con videos que ilustran cómo hacerlos. Está orientado para personas que solo quieren tener un buen estado de salud sin necesidad de ir al gimnasio.
- ***VirtuaGym Fitness Home & Gym:*** Esta solo en inglés y es totalmente gratuita. Cuenta con videos, gráficos y calendarios que te van indicando como debes hacer cada ejercicio. Tiene un asistente que te va diciendo como debes hacer cada ejercicio y te ayuda a dar ánimos durante el entrenamiento. Además cuenta con gráficas de progreso, pero que no son muy buenas. Para poder acceder a varios menús, ejercicios o controles de entrenamiento debes registrarte o darte de alta en su página.
- ***Daily Workout Apps, LLC:*** Es el nombre de la empresa que desarrolla aplicaciones, este se diferencia en que cuenta con seis aplicaciones gratuitas y otras seis que son de pago, que sería como la expansión de las gratuitas. Cada aplicación está ligada a un tipo de entrenamiento muy específico para cada zona muscular. Las de pago cuestan 0,76€ y la más cara 3,06€, que sería una más completa con todos los entrenamientos incluidos en uno. No tiene gráficas de progreso, ni control de peso, pero sí que cuentan con buenos videos que recomiendan cómo hacer cada ejercicio y está disponible solo en español.

La aplicación que he diseñado trata de suprimir los problemas de acceso a internet y los menús ultra complicados de entender y manejar, y proporciona una buena usabilidad para usuarios comprendidos en edades de 16 hasta 40 años. Ofrezco una interfaz que sea fácil de usar, haciendo menús más grandes y visibles de usar para cualquier persona. Es bilingüe, para usuarios de habla hispana o inglesa. Además, los clientes no tendrán por qué preocuparse del acceso a internet o de tener que rellenar formularios para acceder a las herramientas que ofrezco.

Una ventaja sobre los demás, es que los usuarios pueden exportar e importar las tablas de entrenamientos, de esta forma un usuario puede pedir prestada la rutina a alguien, sin la molestia de tener que rellenar todos los datos de nuevo.

También se ofrece control del *BMI*, gráficas de control de progreso, ilustraciones con su respectivo nombre de las rutinas existentes y un formulario donde un usuario puede elegir entre más de 40 ejercicios a la hora de crear su propia tabla de entrenamiento. Por otro lado, también se ofrece un ejemplo de lo que sería un entrenamiento predeterminado, por si un usuario es novato.

Para hacer esto me valgo de las siguientes herramientas de programación:

1. Android es un sistema operativo basado en *Linux*, diseñado principalmente para móviles con pantalla táctil como teléfonos inteligentes o tabletas inicialmente desarrollados por *Android*, Inc., que *Google* respaldó financieramente y más tarde compró en 2005. *Android* fue presentado en 2007 junto a la fundación *Open Handset Alliance*.

2. Tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se han sobrepasado las 700.000 aplicaciones (de las cuales, dos tercios son gratuitas) disponibles en la tienda de aplicaciones oficial de Android: *Google Play*, sin tener en cuenta aplicaciones de otras tiendas no oficiales para *Android*, como pueden ser la *App Store* de *Amazon* o la tienda de aplicaciones *Samsung Apps* de *Samsung*. Google Play es la tienda de aplicaciones en línea administrada por Google. Los programas están escritos en el lenguaje de programación *Java*, aunque también pueden estar escritos en HTML5 + CSS + JavaScript.

3. Soporte Java, aunque la mayoría de las aplicaciones están escritas en Java, no hay una máquina virtual Java en la plataforma. El bytecode Java no es ejecutado, sino que primero se compila en un ejecutable ***Dalvik*** y corre en la Máquina Virtual *Dalvik*. *Dalvik* es una máquina virtual especializada, diseñada específicamente para Android y optimizada para dispositivos móviles que funcionan con batería y que tiene memoria y procesador limitados.

4. *Dalvik* es la máquina virtual que utiliza la plataforma para dispositivos móviles *Android*. Dalvik ha sido diseñada por *Dan Bornstein*, con contribuciones de otros ingenieros de *Google*. Está optimizada para requerir poca memoria y para permitir ejecutar varias instancias de la máquina virtual simultáneamente, lo que sería la gestión de memoria e hilos.

5. *SQLite* es un sistema de gestión de bases de datos relacional, relativamente pequeña (~275 KB) biblioteca escrita en *C*. Es un proyecto de dominio público creado por *D. Richard Hipp*. La biblioteca *SQLite* se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de *SQLite* a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos) son guardados como un solo fichero.

6. *AFreeChart* es un marco de software *open source* para el lenguaje de programación *Java*, el cual permite la creación de gráficos complejos de forma simple. Formado por un equipo japonés, conformado por: Shiraki Seiyu (*ICOMSYSTECH Co., Ltd.),* Ikeda Koichiro, Sato Yoshiaki, Niwano Masayoshi, Yamakami Souichirou, Nada Wataru y Kevin Moray. Donde pretenden facilitar la creación de gráficas estadísticas.

7. *AdMob* es una compañía de publicidad móvil fundada por *Omar Hamoui*. El nombre *AdMob* es un acrónimo para "Publicidad en el móvil". Es una de las mayores plataformas de publicidad móvil del mundo y pretende atender a más de 40 mil millones de banners móviles y anuncios de texto por mes a través de las aplicaciones móviles en los sitios Web y telefonía móvil.

## 1.3. Objetivos

Los objetivos que me propongo para presentar y defender mi Proyecto Fin de Carrera son los siguientes:

- Aprender a desarrollar aplicaciones en plataforma Android con código de programación Java.

- Conocer la arquitectura y principales características de la plataforma Android, asimismo, ser capaz de desarrollar aplicaciones para dar servicios a un sinfín de dispositivos móviles (*Smartphone*).

- Aprender a integrar bases de datos, con interfaz usuario–servidor y conexiones a servidores externos.

- Aprender a usar e implantar *APIs* externas, como la de *AFreeChart*, *Google AdMob* o el propio *SDK* de Android, todo esto en un entorno Java.

- Facilitar el uso de las tablas de entrenamientos en los gimnasios, haciendo portable su rutina y pueda ser compartida entre los usuarios.

- Medir el rendimiento mediante gráficas estadísticas que calcularán la media de peso ganado por cada entrenamiento y siendo específicos para cada grupo muscular.

## 1.4. Solución propuesta

Como solución propongo elaborar una interfaz sencilla en código abierto *Android*, programado en Java. De tal forma que los accesos sean de forma rápida y sin errores. Los botones tendrán un tamaño relativo que se ajuste a los diferentes dispositivos móviles, además este tamaño será medianamente grande, para garantizar que al pulsar cada tecla, no haya equivocaciones, ya que la fatiga muscular suele causar que los usuarios pulsen inadecuadamente en tecnologías táctiles.

Mostraré imágenes de entrenamientos que ayuden al usuario a guiarse durante su sesión de musculación, tonificación o ganancia de resistencia física.

Crearé un menú con listas dinámicas donde el usuario pueda crear, modificar o eliminar sus ejercicios si así lo desea.

Ofrezco gráficas de progreso, donde se hará la media del peso ganado por los tres tipos diferenciados de entrenamientos ya previamente definidos.

Facilidad de exportar e importar entrenamientos con otros usuarios. Primero que todo, se creará un fichero con el contenido de la tabla de la Base de Datos (*BD*), que luego podrá ser exportada e importada a otros móviles que usen esta misma aplicación, esto facilita el intercambio de rutinas entre los deportistas.

También propongo que sea bilingüe, inglés o español, ya que así gano más mercado y me doy a conocer como desarrollador en el Mundo de *Google Play*. El modo inglés o español es automático, ya que mi *App* detecta el idioma de entrada del teléfono y lo configura en su respectivo idioma.

Para el Cálculo del índice de la masa corporal (IMC), Utilizaré la siguiente fórmula que recomienda la Organización mundial de la salud. Esta se usa para dar una medida de grasa corporal asociada al peso y la altura de cada persona.

$$\mathrm{IMC} = \frac{peso(kg)}{altura^2(m)}$$

Mi proyecto está creado para móviles que dispongan *Android 2.3.x Gingerbread* (Api 10) en adelante, incluidos las tabletas, para ofrecer más cobertura y que llegue a todos los usuarios. Se eligió esta, porque así se puede llegar a más personas.
Las siguientes gráficas muestran el uso de cada versión Android hasta finales de diciembre del 2012.

## Cuota de las versiones

Datos recogidos a principios del mes de diciembre.

| Plataforma | Nivel de API | % |
|---|---|---|
| 4.x.x *Jelly Bean* | 16-17 | 6,7% |
| 4.0.x *Ice Cream Sandwich* | 14-15 | 27,5% |
| 3.x.x *Honeycomb* | 12-13 | 1,6% |
| 2.3.x *Gingerbread* | 9-10 | 50,8% |
| 2.2 *Froyo* | 8 | 10,3% |
| 2.1 *Eclair* | 7 | 2,7% |
| 1.6 *Donut* | 4 | 0,3% |
| 1.5 *Cupcake* | 3 | 0,1% |

Ilustración 1- Tabla que representa la plataforma que más se usa

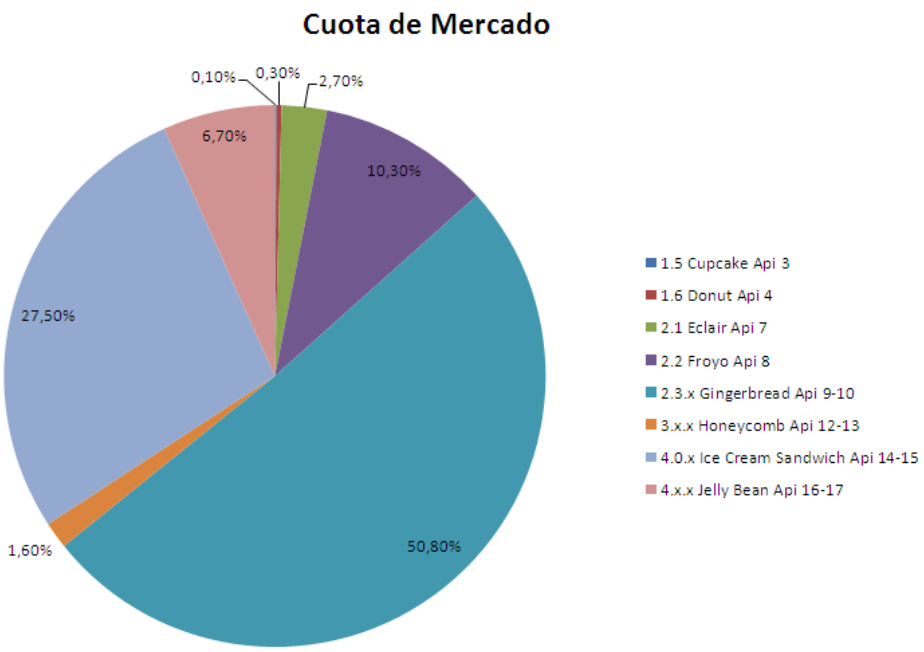



Ilustración 2 – Representación circular

# 2. ANÁLISIS

Análisis de requisitos de la aplicación, en el que se presentan los requisitos funcionales y no funcionales, con sus respectivos casos de uso y siguiendo la metodología de proceso unificado.

## 2.1. Análisis de Requisitos

Este proyecto consta básicamente en hacer una aplicación para móviles con tecnología Android. Va a tener una interfaz gráfica que será la encargada de interactuar con el sistema gestor de bases de datos, el cual guardará, tratará y mostrará toda la información que los usuarios pongan a su disposición.

El proyecto general trata de hacer una tabla de entrenamientos para gimnasios, el cual se ubica dentro del mundo del deporte, salud y bienestar. Se espera que con el uso masivo por parte de las personas y el avance de las nuevas tecnologías (*Smartphone*) éste remplace al papel y lápiz de toda la vida, añadiendo herramientas que permitan ver el progreso durante, antes o después del entrenamiento.

### *2.1.1. Requisitos Funcionales*

1) La aplicación debe ser compatible para dispositivos móviles tipo *Android* versión 2.3.X *Gingerbread* en adelante y tendrá que ser gratuita.

2) La aplicación debe ofrecer una tabla de entrenamiento predeterminada para usuarios noveles en el mundo deportivo y una rutina que pueda ser personalizada, para personas que tengan mayores conocimientos a la hora de crear rutinas de entrenamientos en función de su experiencia.

3) La aplicación debe ser capaz de tratar los datos de manera que estos puedan crearse, modificarse o eliminarse sin causar errores.

4) El programa almacenará los datos en una pequeña base de datos de cada dispositivo móvil que use la *App*. Ésta debe ser consistente y tolerante a fallos.

5) El programa debe ofrecer imágenes alusivas a las rutinas y formas de ejecutar cada ejercicio específico de cada entrenamiento muscular. Las imágenes deben ser separadas por cada grupo muscular y ordenadas alfabéticamente.

6) Los grupos musculares más diferenciados son: Pierna, Pecho, Espalda, Hombro, Bíceps y Tríceps.

7) La aplicación debe ser capaz de diferenciar el lenguaje del dispositivo móvil, ya que por defecto la aplicación deberá estar en inglés, pero con la suficiente autonomía de ponerlo a español, si son para personas de habla hispana. Debe ser bilingüe.

8) Debe mostrar en una ventana el progreso del entrenamiento, a través de gráficas estadísticas que indiquen el progreso medio por cada entrenamiento seleccionado, mostrando fechas y el peso medio adquirido. Debe estar distinguido por grupos musculares.

9) La aplicación debe dar la opción de compartir, ya que esta debe promocionarse mediante los medios disponibles que existen en las redes sociales, como lo son *YouTube*, *Facebook*, *WhatsApp*, etc.

10) También es necesario que los entrenamientos puedan ser compartidos con los usuarios que usen esta aplicación, de tal manera que puedan exportarse e importase las rutinas con otros usuarios.

11) La aplicación tendrá publicidad de *Google AdMob*, ya que se pretende tener beneficios a futuro con la publicad gratuita.

12) También debe proporcionar una pequeña encuesta para ver los resultados de aceptación de esta aplicación, mediante preguntas cortas y sencillas. De esta información se espera hacer una evaluación estadística, para valorar si se puede mejorar más. La encuesta debe enviarse y guardarse en un servidor externo ajeno a la aplicación, que se espera sea gratuito o que no genere muchos costes.
    La encuesta solo se hará una vez por cada usuario y una vez finalizada con éxito debe ocultarse esta opción y mostrar como recompensa otra opción que les permita a los usuarios calcular el Índice de Masa Corporal adecuado y establecido por la organización mundial de la salud.

13) Para ayudar a los usuarios a organizarse a la hora de crear sus tablas, se establece que haya una ventana que permita separar por un estilo de entrenamiento, ya sea Musculación, Tonificación o Resistencia. Y mostrar los entrenamientos creados en una lista, indicando el número de Series, Nombre del Ejercicio y la cantidad de Peso utilizado.

14) Las imágenes deben ser muy livianas y de excelente calidad y entendimiento.

15) La aplicación debe adaptarse al contorno de los diferentes modelos (dimensiones) de dispositivos. No debe haber errores de diseño o de sobre proporción.

16) Facilitar la búsqueda y descarga, para esto ha de crearse una cuenta en *Google Play* y *Google AdMob*, para poder subir la aplicación a *Android Market* y poder dar publicad, de esta manera hacemos que la aplicación sea de primera calidad al ofrecerla a nivel mundial. No se recomienda usar otras páginas para dar este servicio.

17) Permitir actualizaciones a todos los usuarios y que por defecto esta se almacene en la tarjeta SD.

18) Buscar la forma de que la aplicación sea lo menos pesada posible.

### *2.1.2. Requisitos no funcionales*

1) La aplicación trabajará con un volumen de información muy pequeño, así que se espera que la información almacenada sea lo más sencilla y clara posible.

2) El diseño debe estar pensado para personas de una edad comprendida entre los 16 y los 40 años como mucho. También ha de tener en cuenta el nivel de fátiga y cansancio, ya que a la hora de manejar aplicaciones táctiles, los usuarios tienden a ser poco acertados a la hora de navegar por las ventanas y menús.

3) Debe crearse una página web que permita dar publicidad a la aplicación, debe ser sencilla y alojada en un servidor gratuito con su correspondiente nombre, haciendo referencia al nombre de la aplicación. Debe tener un menú de noticias, imágenes que representen los distintos niveles de la aplicación y enlaces de descarga a Google Play.

4) El programa no debe bloquearse, ya que esto repercutirá de forma desastrosa en la imagen del producto.

5) Debe crearse un logo llamativo para que pueda ser fácilmente reconocido en la web y redes sociales.

6) Debe mostrar errores en caso de que se produzcan, informando del error al usuario. Aunque se obliga a que este tipo de cosas no suceda, si ocurren deben ser corregidos lo más pronto posible.

### *2.1.3. Análisis y Diagramas de los Casos de Uso*

#### *2.1.3.1. Metodología de Trabajo.*

Siguiendo la metodología de Proceso Unificado (UP/RUP) los requisitos especificados en el apartado anterior, deben ser analizados y convertidos en Casos de Uso. Los actores que se han detallado son los usuarios finales de la aplicación "**Workout Routines**".

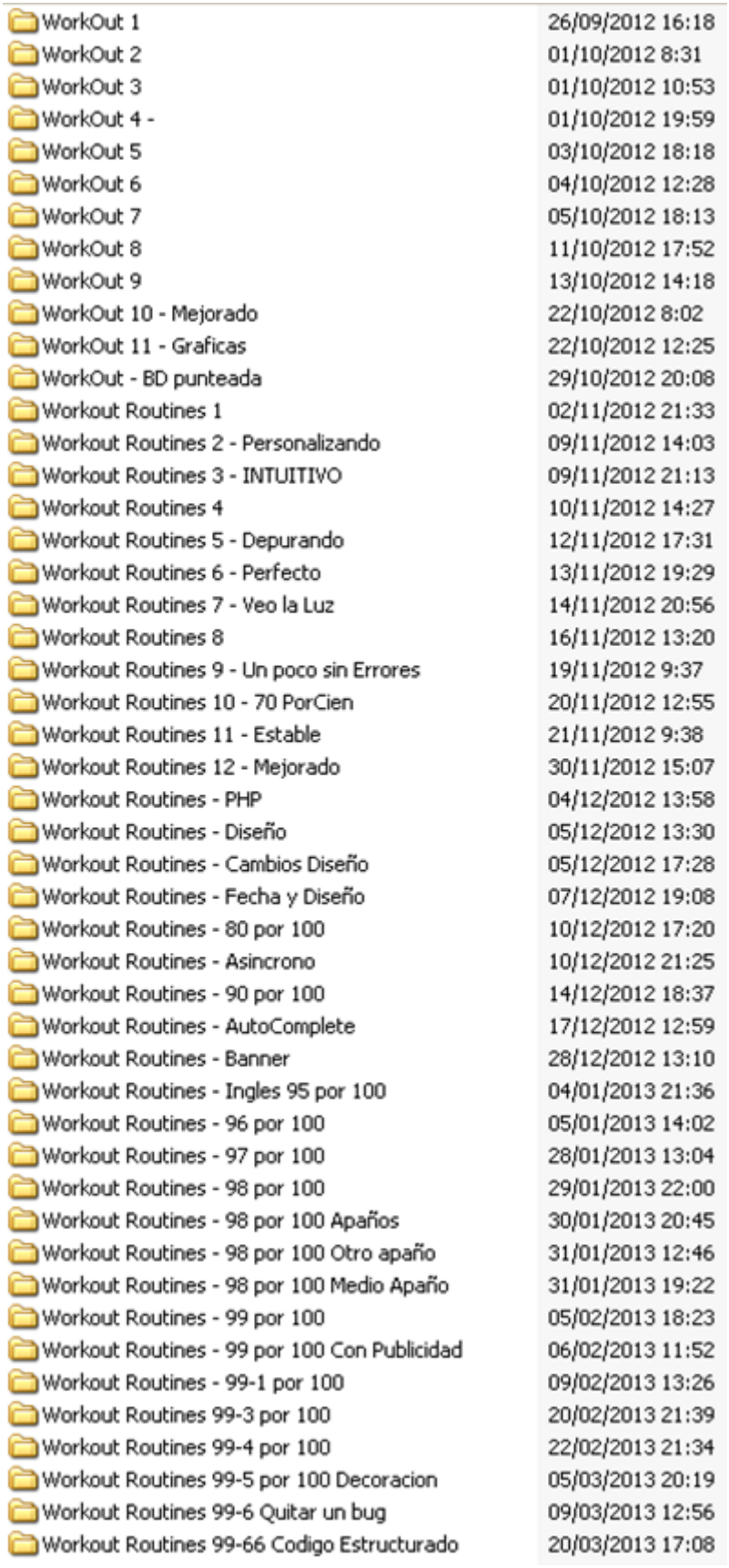
Una característica de esta metodología, es que es iterativa e incremental, dando como resultado un aumento (Mejora) al producto desarrollado, ya sean nuevas funcionalidades o depuración de errores por cada fase de iteración.

En la siguiente ilustración voy a enseñar cómo fue creciendo y evolucionando la aplicación, a medida en que esta iba siendo mejorada en cada iteración y se iban añadiendo nuevas funcionalidades o nuevos diseños a partir de los requisitos.

Cada ciclo del proyecto, por lo general era reemplazado una vez a la semana, dando por concluido que esa etapa había acabado y que todo estaba en condiciones para ser guardado y tratado como la última copia de seguridad (Versión mejorada).

Aunque, habían semanas donde esto no ocurría, porque no se llegaba a la etapa primordial, que era que la funcionalidad a tratar estuviera funcionando correctamente, así que es posible que en vez de ser una semana, fueran dos o casi tres.

La siguiente imagen muestra las versiones de este proyecto, ordenados por fecha de creación y modificación.



| | |
|---|---|
| WorkOut 1 | 26/09/2012 16:18 |
| WorkOut 2 | 01/10/2012 8:31 |
| WorkOut 3 | 01/10/2012 10:53 |
| WorkOut 4 - | 01/10/2012 19:59 |
| WorkOut 5 | 03/10/2012 18:18 |
| WorkOut 6 | 04/10/2012 12:28 |
| WorkOut 7 | 05/10/2012 18:13 |
| WorkOut 8 | 11/10/2012 17:52 |
| WorkOut 9 | 13/10/2012 14:18 |
| WorkOut 10 - Mejorado | 22/10/2012 8:02 |
| WorkOut 11 - Graficas | 22/10/2012 12:25 |
| WorkOut - BD punteada | 29/10/2012 20:08 |
| Workout Routines 1 | 02/11/2012 21:33 |
| Workout Routines 2 - Personalizando | 09/11/2012 14:03 |
| Workout Routines 3 - INTUITIVO | 09/11/2012 21:13 |
| Workout Routines 4 | 10/11/2012 14:27 |
| Workout Routines 5 - Depurando | 12/11/2012 17:31 |
| Workout Routines 6 - Perfecto | 13/11/2012 19:29 |
| Workout Routines 7 - Veo la Luz | 14/11/2012 20:56 |
| Workout Routines 8 | 16/11/2012 13:20 |
| Workout Routines 9 - Un poco sin Errores | 19/11/2012 9:37 |
| Workout Routines 10 - 70 PorCien | 20/11/2012 12:55 |
| Workout Routines 11 - Estable | 21/11/2012 9:38 |
| Workout Routines 12 - Mejorado | 30/11/2012 15:07 |
| Workout Routines - PHP | 04/12/2012 13:58 |
| Workout Routines - Diseño | 05/12/2012 13:30 |
| Workout Routines - Cambios Diseño | 05/12/2012 17:28 |
| Workout Routines - Fecha y Diseño | 07/12/2012 19:08 |
| Workout Routines - 80 por 100 | 10/12/2012 17:20 |
| Workout Routines - Asincrono | 10/12/2012 21:25 |
| Workout Routines - 90 por 100 | 14/12/2012 18:37 |
| Workout Routines - AutoComplete | 17/12/2012 12:59 |
| Workout Routines - Banner | 28/12/2012 13:10 |
| Workout Routines - Ingles 95 por 100 | 04/01/2013 21:36 |
| Workout Routines - 96 por 100 | 05/01/2013 14:02 |
| Workout Routines - 97 por 100 | 28/01/2013 13:04 |
| Workout Routines - 98 por 100 | 29/01/2013 22:00 |
| Workout Routines - 98 por 100 Apaños | 30/01/2013 20:45 |
| Workout Routines - 98 por 100 Otro apaño | 31/01/2013 12:46 |
| Workout Routines - 98 por 100 Medio Apaño | 31/01/2013 19:22 |
| Workout Routines - 99 por 100 | 05/02/2013 18:23 |
| Workout Routines - 99 por 100 Con Publicidad | 06/02/2013 11:52 |
| Workout Routines - 99-1 por 100 | 09/02/2013 13:26 |
| Workout Routines 99-3 por 100 | 20/02/2013 21:39 |
| Workout Routines 99-4 por 100 | 22/02/2013 21:34 |
| Workout Routines 99-5 por 100 Decoracion | 05/03/2013 20:19 |
| Workout Routines 99-6 Quitar un bug | 09/03/2013 12:56 |
| Workout Routines 99-66 Codigo Estructurado | 20/03/2013 17:08 |

**Ilustración 3 - Versiones del proyecto Workout Routines**

### *2.1.3.2. Diagrama de Casos de Uso General de la Aplicación.*

Este modelo de caso de uso, representa de manera generalizada las diferentes herramientas creadas y disponibles en este proyecto.

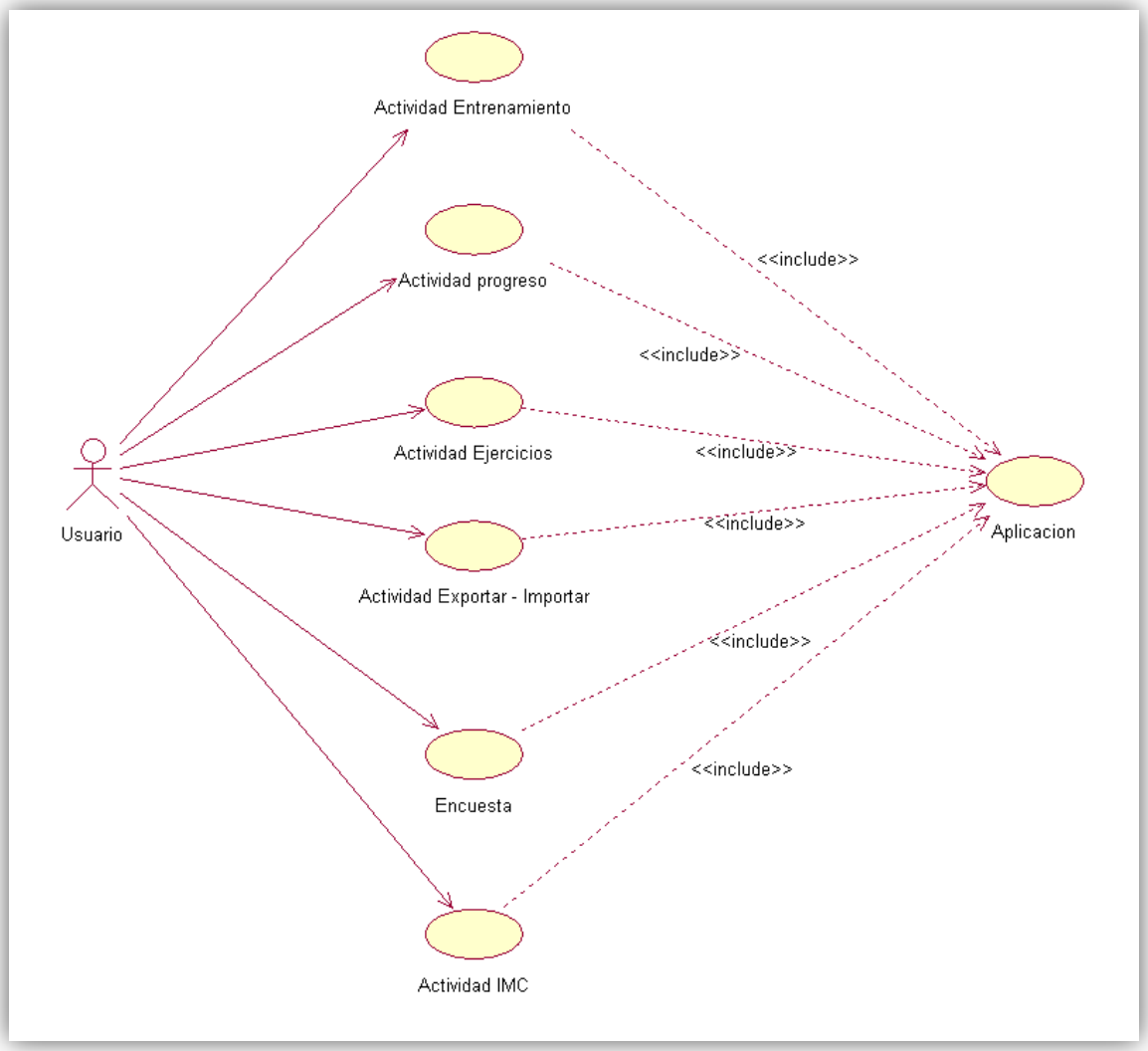



**Ilustración 4- Diagrama de casos de uso general**

### *2.1.3.3. Análisis y Diagrama Caso de Uso: Descarga de la Aplicación*

| Caso de uso: Descarga de la Aplicación | |
|---|---|
| **Descripción** | Descarga de la aplicación |
| **Precondición** | Tener acceso a internet |
| **Camino básico** | 1. El usuario se conecta a la aplicación "*Play Store*"<br>2. El usuario buscará nuestra app "Workout Routines"<br>3. Le dará a descargar y se instalará<br>4. Esta se almacenará en la tarjeta SD por defecto.<br>5. Ejecutar y el usuario ya podrá usar la app |
| **Camino alternativo** | 6. Error al descargar<br>7. No hay suficiente espacio de memoria en el móvil, tanto en la tarjeta SD como en la memoria interna del dispositivo. |
| **Postcondición** | Aplicación instalada correctamente. |

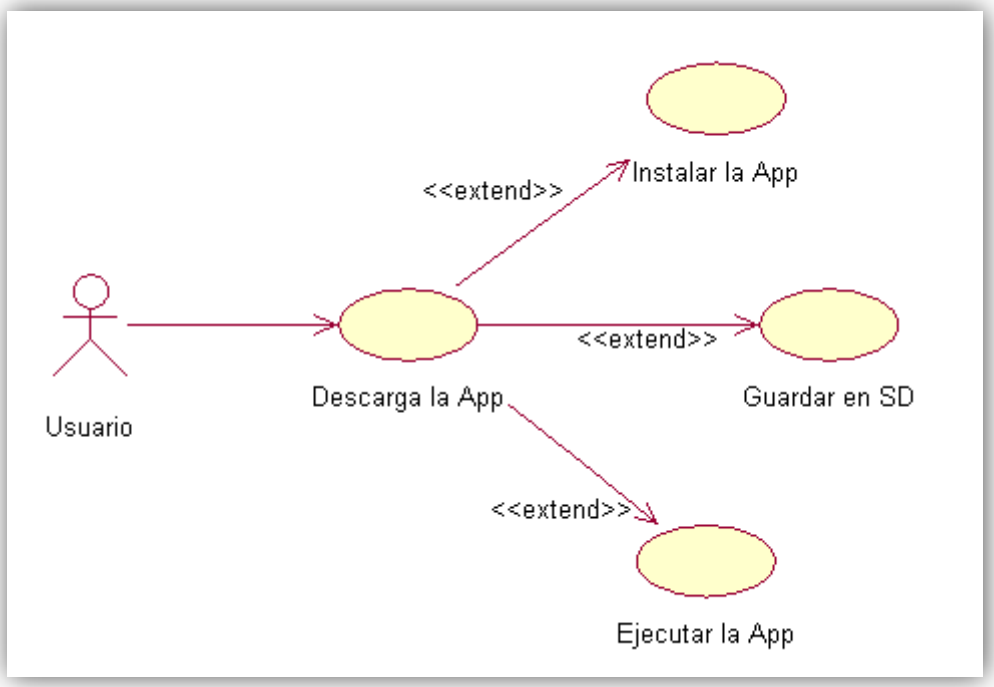



**Ilustración 5- Diagrama caso de uso, descarga de la app**

### *2.1.3.4. Análisis y Diagrama de Caso de Usos: Crear, Editar y Modificar Rutina Personalizada.*

| **Caso de uso: Crear Rutina Personalizada** | |
|---|---|
| **Descripción** | El usuario experto puede crear su rutina de entrenamiento, ayudado de una lista que se autocompleta. |
| **Precondición** | Encontrase en el Menú Entrenamiento |
| **Camino básico** | 1. El usuario pulsará la tecla (Entrenamientos).<br>2. Esta le llevará a una ventana con dos pestañas, donde una será la personalizada.<br>3. El usuario puede decidir dónde crear - guardar su rutina, ya sea para ejercicios de Musculación, Resistencia o Tonificación.<br>4. Al pulsar una de estas tres opciones, se le abrirá una pequeña lista, con un aviso de "ejemplo".<br>5. Al pulsar, se le dará la opción de modificar o crear un nuevo entrenamiento, con sus correspondientes entradas: nombre del ejercicio, numero de series y cantidad de peso usado.<br>6. Pulsar el botón guardar |
| **Camino alternativo** | 7. Seleccionar pestaña predeterminada.<br>8. Salir de la aplicación. |
| **Postcondición** | Ejercicio creado correctamente |

| **Caso de uso: Editar Ejercicio** | |
|---|---|
| **Descripción** | El usuario podrá consultar su rutina que ha ido creando en una lista con sus correspondientes datos (Series, Nombre del Ejercicio y Peso) y podrá modificar o eliminar el entrenamiento seleccionado si así lo desea. |
| **Precondición** | Insertar al menos un dato. |
| **Camino básico** | 1. El usuario pulsa el botón Entrenamientos.<br>2. Este se dirige a la pestaña de Personalizado.<br>3. Pulsará cualquiera de los tres botones, Musculación, Resistencia o Tonificación.<br>4. Se abrirá una ventana nueva que mostrará su tabla creada, la cual podrá seguir durante el entrenamiento.<br>5. Seleccionará en su lista el ejercicio que desee.<br>6. Este abrirá una nueva ventana que le indicará que desea modificar algún dato,<br>7. El usuario hará los cambios pertinentes<br>8. Pulsara luego el botón de modificar.<br>9. Mostrará aviso de que ha sido correcta la actualización |
| **Camino alternativo** | 10. Saldrá de la app.<br>11. Pulsar la pestaña de predeterminada. |
| **Postcondición** | El usuario visualizar correctamente la rutina creada. |

| Caso de uso: Eliminar Ejercicio | |
|---|---|
| **Descripción** | El usuario podrá eliminar los ejercicios creados, seleccionando el que le interese. |
| **Precondición** | Insertar o tener al menos un dato. |
| **Camino básico** | 1. El usuario pulsa el botón Entrenamientos.<br>2. Este se dirige a la pestaña de Personalizado.<br>3. Pulsara cualquiera de los tres botones, Musculación, Resistencia o Tonificación.<br>4. Se abrirá una ventana nueva que mostrará su tabla creada y el cual podrá seguir durante el entrenamiento.<br>5. Seleccionará en su lista el ejercicio que desee.<br>6. Este abrirá una nueva ventana que le indicará que desea modificar algún dato,<br>7. El usuario pulsará el CheckBox de eliminar entrenamiento.<br>8. Pulsará luego el botón de modificar.<br>9. Mostrará aviso de que ha sido eliminado correctamente el ejercicio. |
| **Camino alternativo** | 10. Saldrá de la app.<br>11. Pulsar la pestaña de predeterminada. |
| **Postcondición** | El usuario habrá eliminado el ejercicio seleccionado. |

La siguiente imagen refleja los Modelos de Casos de Uso: Crear, Modificar y Eliminar de manera Generalizada.

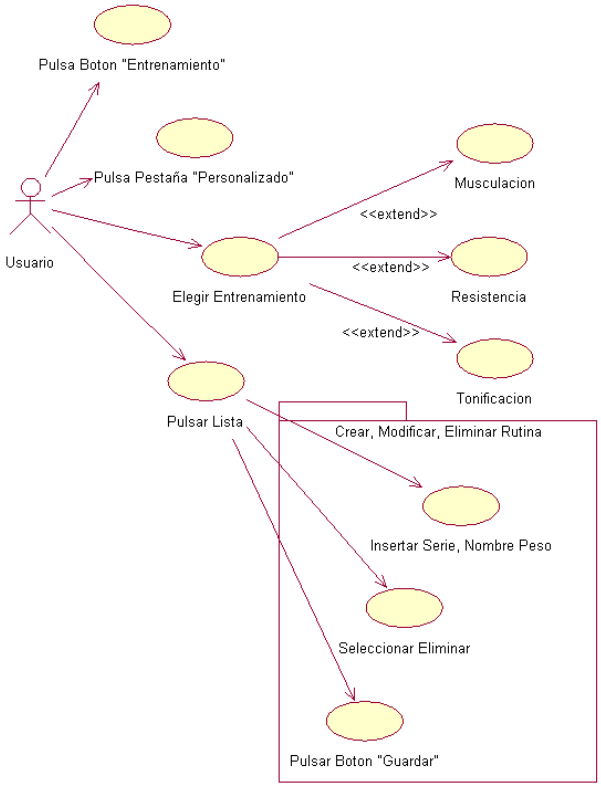



Ilustración 6 - Caso de uso. Creación, modificación y eliminación de ejercicios

### *2.1.3.5. Análisis y Diagrama de Caso de Uso: Visualizar Rutina Predeterminada*

| Caso de uso: Usar "Visualizar" Rutina Predeterminada | |
|---|---|
| **Descripción** | El usuario podrá usar la rutina predeterminada como ejemplo a seguir para crear la rutina predeterminada. |
| **Precondición** | Encontrase en Menú Entrenamiento. |
| **Camino básico** | 1. El usuario selecciona la pestaña predeterminada.<br>2. El usuario sigue parcialmente la lista, ya que es una tabla de entrenamiento creada para que la sigan los usuarios novatos y que sean ellos los que gradúen su peso.<br>3. El usuario sigue el entrenamiento seleccionado hasta finalizar su entrenamiento. |
| **Camino alternativo** | 4. El usuario sale de la aplicación.<br>5. Selecciona la pestaña predeterminada. |
| **Postcondición** | El usuario se hace una idea de cómo se hace una tabla balanceada por carga de trabajo. |

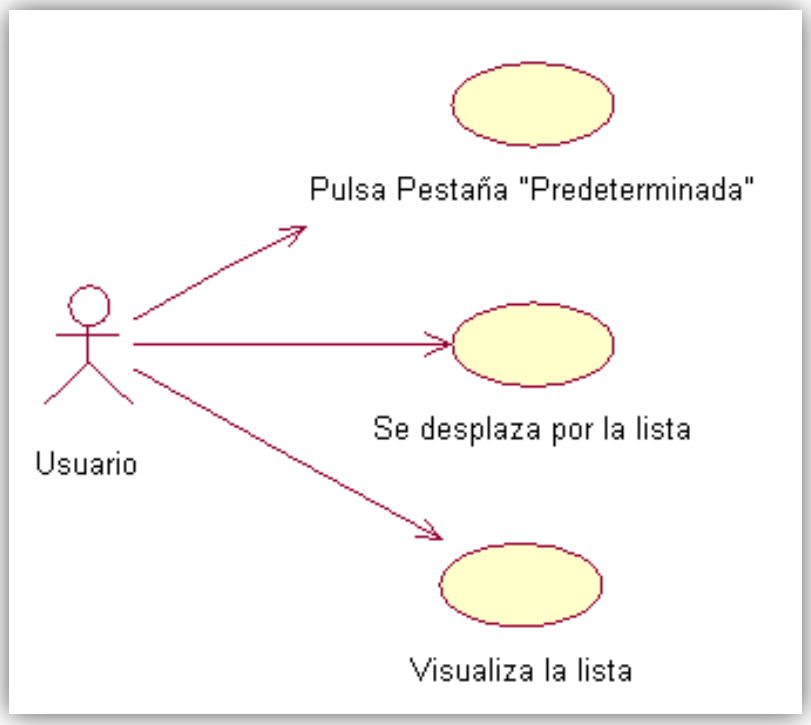



**Ilustración 7 - Caso de uso, Visualizar rutina predeterminada**

### *2.1.3.6. Análisis y Diagrama de Casos de Uso: Uso Gráficas de Progreso*

| Caso de uso: Uso gráficas de Progreso | |
|---|---|
| **Descripción** | Los usuarios pueden ver un progreso medio de sus tres tipos de entrenamientos. |
| **Precondición** | Que haya al menos dos ejercicios creados en la pestaña de personalizado. |
| **Camino básico** | 1. El usuario estando en el Menú inicio pulsara la tecla "Progreso".<br>2. Esta le mostrará una serie de pestañas separadas por seis tipos de grupos musculares, donde se le mostrará cómo ha ido mejorando el usuario cada día.<br>3. El usuario elige el grupo muscular a consultar y mi aplicación dibujará una gráfica estadística a partir de los pesos usados. |
| **Camino alternativo** | 4. El usuario sale de la aplicación. |
| **Postcondición** | |

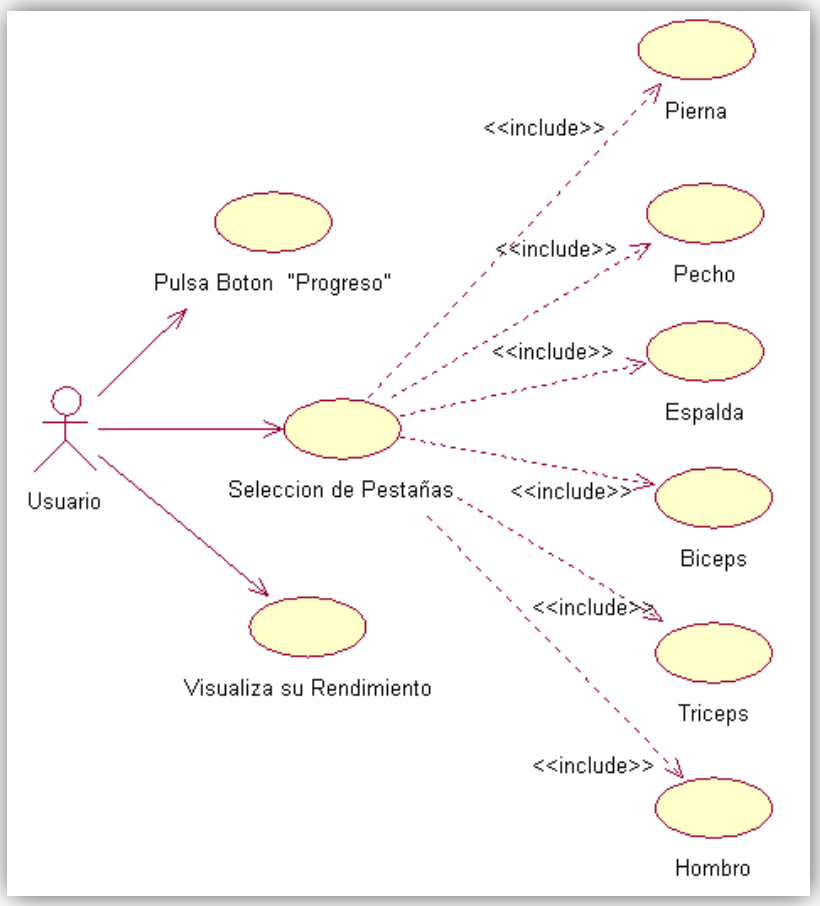



**Ilustración 8 - Caso de uso, Uso de gráficas de progreso**

### *2.1.3.7. Análisis y Diagrama de Caso de Uso: Ejercicios*

| Caso de uso: Ejercicios | |
|---|---|
| **Descripción** | Los usuarios podrán consultar el nombre y la forma correcta de ejecutar cada ejercicio. |
| **Precondición** | Estar en el Menú inicio. |
| **Camino básico** | 1. El usuario pulsa el botón "Ejercicios".<br>2. Esta opción mostrará seis pestañas separadas pos grupos musculares.<br>3. El usuario elige la pestaña deseada.<br>4. Cada pestaña tiene una lista ordenada de ejercicios, con su correspondiente imagen. |
| **Camino alternativo** | 5. El usuario sale de la aplicación |
| **Postcondición** | El usuario ha consultado cómo se hace un ejercicio específico para cada grupo muscular. |

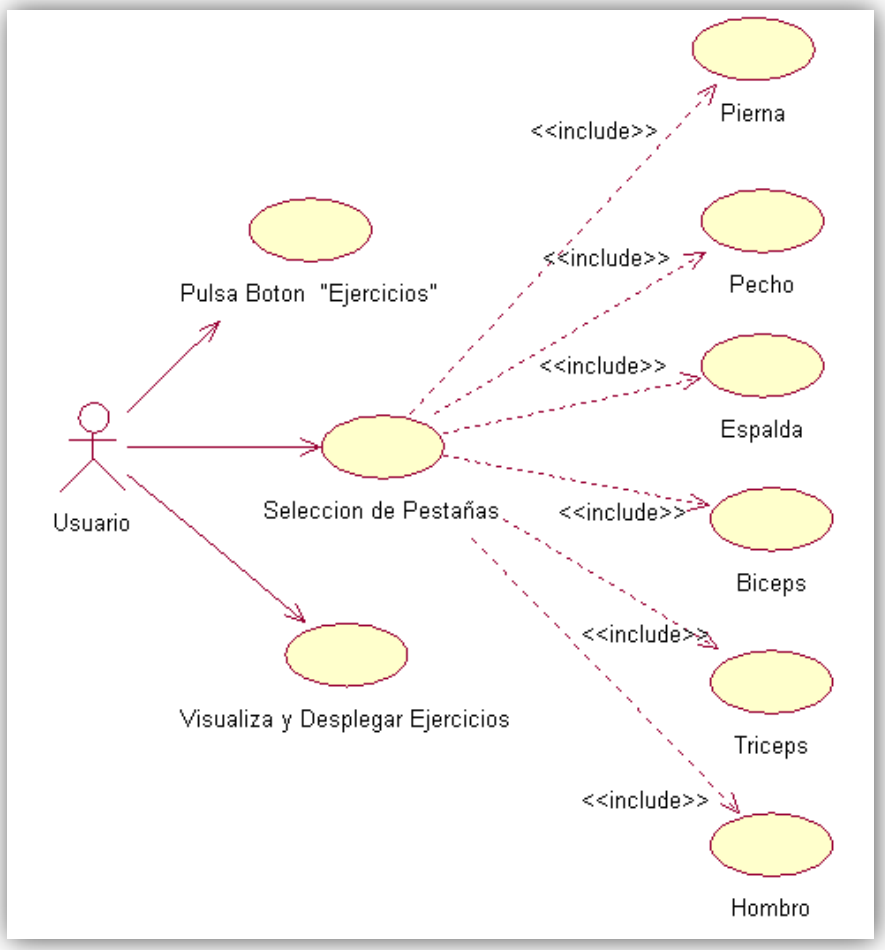



Ilustración 9 - Caso de uso, botón de ejercicios

### *2.1.3.8. Análisis y Diagrama de Casos de Uso: Exportar e Importar Rutina*

| **Caso de uso: Exportar Rutina** | |
|---|---|
| **Descripción** | Los usuarios pueden compartir sus rutinas con usuarios que usen nuestra misma app. |
| **Precondición** | Tener creada al menos una rutina personalizada<br>Tener activo el Bluetooth o el acceso a la red. |
| **Camino básico** | 1. El usuario selecciona el botón "Exportar"<br>2. Este le mostrara el aviso de cómo desea exportar la tabla, ya sea por *Bluetooth*, *Drive*, o *Gmail*.<br>3. El usuario elige la opción que le convenga y activa todas las opciones que necesite.<br>4. Y le da a enviar. |
| **Camino alternativo** | 5. El usuario selecciona salir de la *App*.<br>6. Elige importar la tabla. |
| **Postcondición** | El usuario ha mandado a compartir su rutina con terceras personas. |

| **Caso de uso: Importar Rutina** | |
|---|---|
| **Descripción** | Los usuarios pueden usar rutinas ya creadas por otros usuarios, de esta forma evitan crear una desde cero. Importan la rutina y esta se verá reflejada. |
| **Precondición** | Tener creada al menos una rutina personalizada<br>Tener activo el *Bluetooth* o el acceso a la red.<br>Usar un gestor de archivos para móviles *Android*.<br>Que la rutina exportada esté en la Ubicación correcta. |
| **Camino básico** | 1. El usuario pulsará la tecla importar y ésta automáticamente inserta los nuevos datos en su BD. |
| **Camino alternativo** | 2. Error, el archivo no está en la ruta especifica.<br>3. Error al insertar datos. |
| **Postcondición** | El usuario ya podrá consultar su tabla nueva en entrenamientos personalizados. |

La siguiente gráfica de casos de uso, muestra los análisis de casos de uso, Exportar e Importar Rutina, ya que de alguna manera, una depende de la otra.

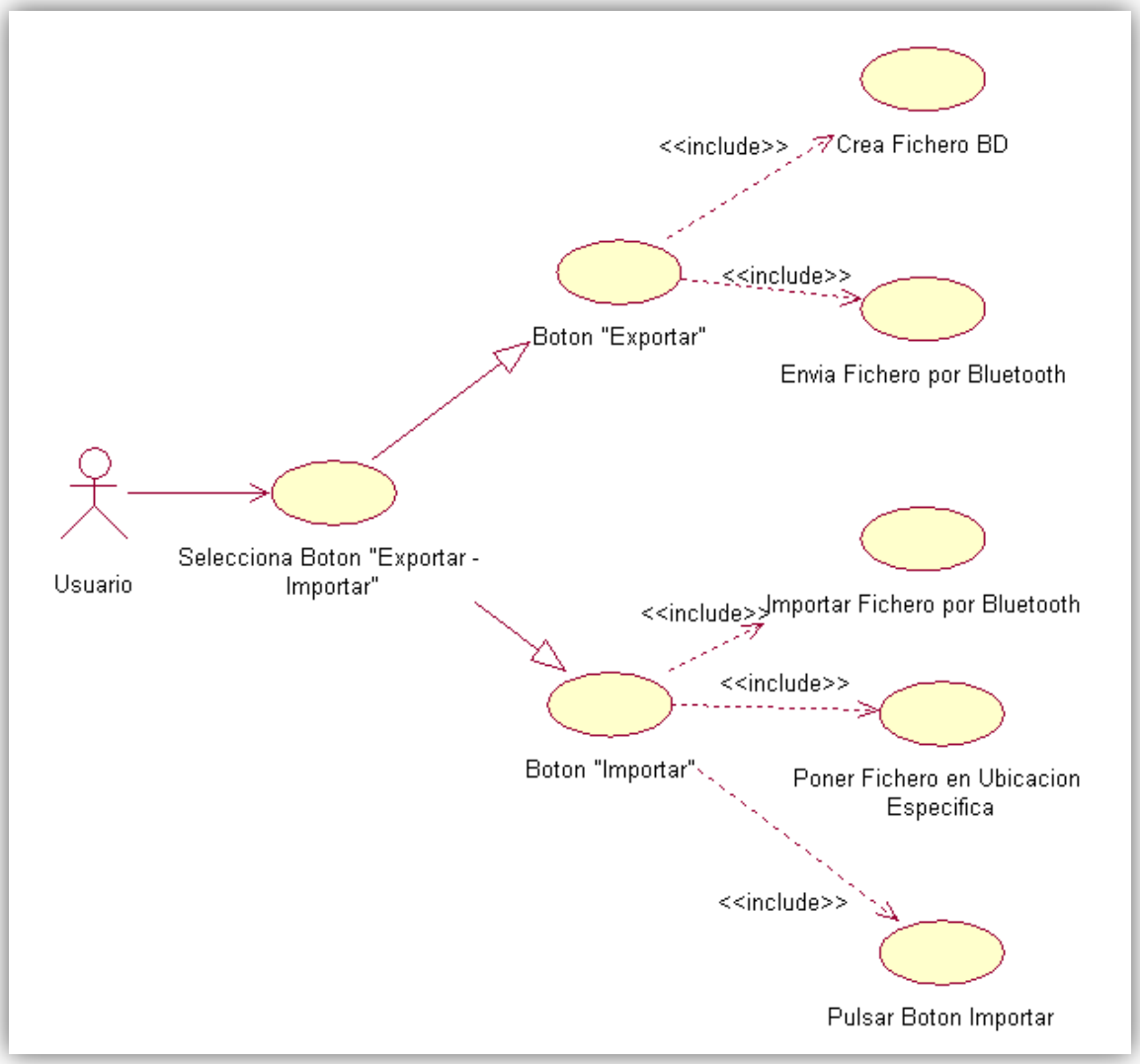



**Ilustración 10 - Caso de uso, Exportar e Importar rutina**

### *2.1.3.9. Análisis y Diagrama de Casos de Uso: Encuesta*

| Caso de uso: Encuesta | |
|---|---|
| **Descripción** | Esta pequeña encuesta sirve para identificar el grado de satisfacción y uso de esta aplicación por parte de los usuarios. |
| **Precondición** | Tener acceso a internet.<br>Estar en el menú inicio. |
| **Camino básico** | 1. Pulsar el botón de encuesta.<br>2. El usuario hará la encuesta, sin dejarse una pregunta sin contestar.<br>3. Le dará a enviar.<br>4. Éste informará de que ha sido enviada con éxito.<br>5. El botón se ocultará y dejará paso al botón IMC. |
| **Camino alternativo** | 6. Mostrará mensaje de error si no tiene conexión a internet.<br>7. Indicará si no se ha rellenado todo el formulario.<br>8. El usuario sale de la ventana encuesta. |
| **Postcondición** | Se finaliza la encuesta con éxito y se ocultará esta opción. |

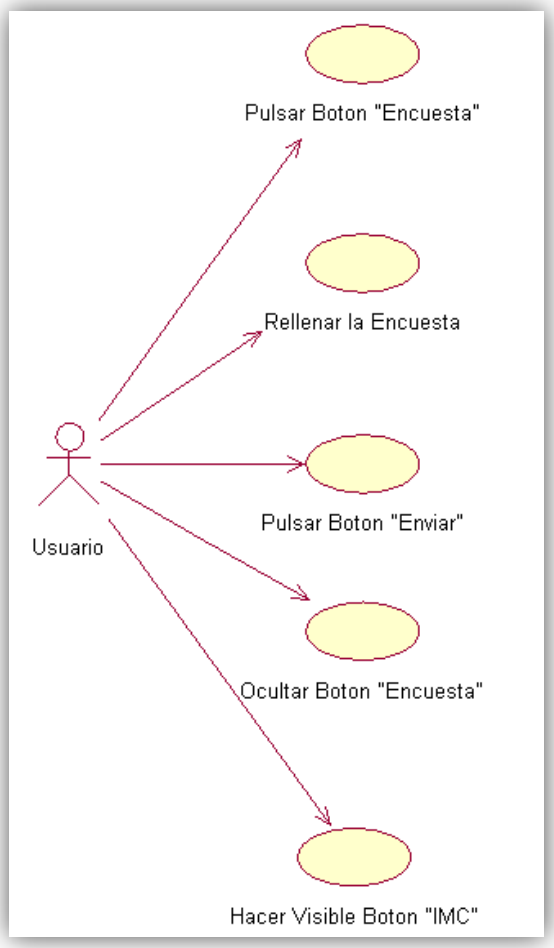



Ilustración 11 - Caso de uso, Encuesta

### *2.1.3.10. Análisis y Diagrama de Casos de Uso: Calculador IMC*

| Caso de uso: Calculador IMC | |
|---|---|
| **Descripción** | Esta herramienta es capaz de calcular el índice de masa corporal de las personas. |
| **Precondición** | Haber enviado la encuesta con éxito. |
| **Camino básico** | 1. El usuario pulsa el botón IMC<br>2. Éste mostrará dos campos a rellenar. Campo Peso y Estatura.<br>3. El usuario inserta los datos y pulsa el botón calcular.<br>4. Éste generará un coeficiente de IMC que el usuario podrá medir con una tabla dada. |
| **Camino alternativo** | 5. El usuario sale de la app. |
| **Postcondición** | El usuario conocerá su índice de masa corporal y con base a eso podrá tomar decisiones acerca de la dieta a seguir. |

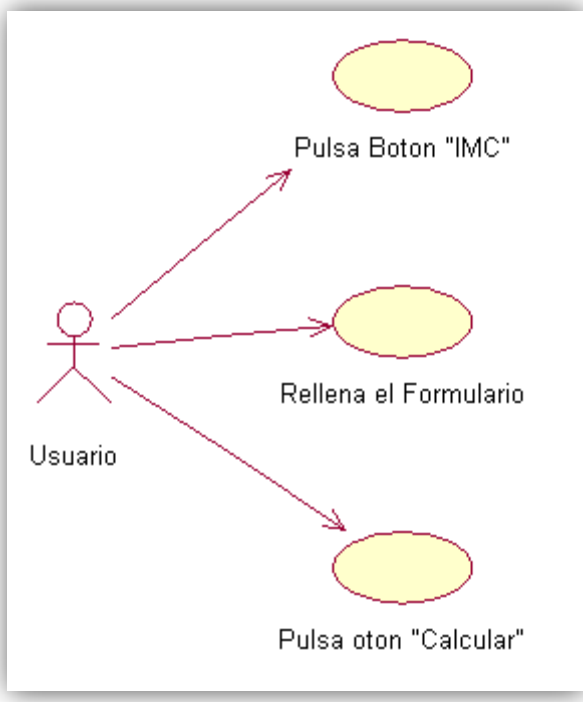



**Ilustración 12- Caso de uso, Índice de masa corporal**

### *2.1.3.11. Análisis y Diagrama de Casos de Uso: Compartir*

| Caso de uso: Compartir | |
|---|---|
| **Descripción** | Esta herramienta permite compartir y dar a conocer la *App* en las Redes Sociales, enviando información acerca de la aplicación, *URL*. |
| **Precondición** | |
| **Camino básico** | 1. El usuario pulsa el botón Compartir<br>2. Elige una opción, de las tantas que le aparecerán.<br>3. Compartirá y aceptará. |
| **Camino alternativo** | |
| **Postcondición** | El usuario dará a conocer la aplicación a sus contactos de teléfono, redes sociales, etc. |

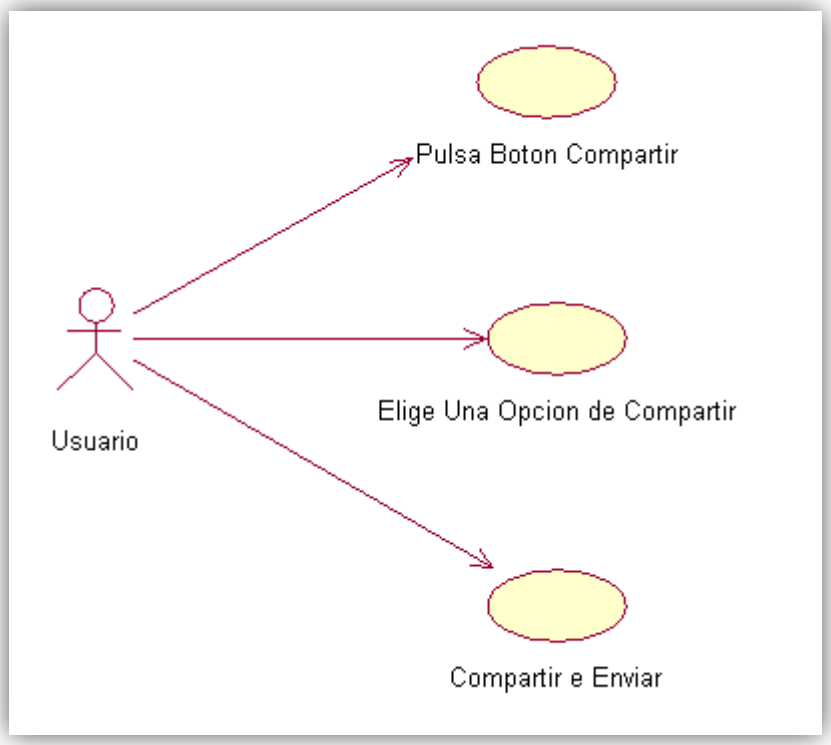



**Ilustración 13 - Caso de uso, Boton compartir app**

### *2.1.4. Requisitos de Hardware*

El requisito es que deben ser para dispositivos táctiles, como móviles o tablets.
Han de disponer de 4 megas libres, en la tarjeta SD o en la memoria del dispositivo en cuestión.

No es necesario disponer de servidores propios, ya que la página web y la BD externa estarán alojados en un servidor gratuito externo.

Además, para el almacenamiento de la aplicación, ésta estará alojada en los servidores de *Google Play*, haciendo fácil la búsqueda y descarga.

# 3. DISEÑO

En este capítulo del diseño de la aplicación, voy a representar los diagramas de secuencia correspondientes a los casos de uso y también los diagramas de clases que componen mis clases Java.

## 3.1. Diagramas de Secuencias

A continuación se describen los diagramas de secuencia los cuales se inician en el momento en el que el usuario accede a la aplicación, mostrando una interacción con nuestras herramientas a través del tiempo.

### *3.1.1. Diagrama de secuencia general*

El siguiente Diagrama De Secuencia muestra de manera general como el usuario interactúa con la aplicación Android.

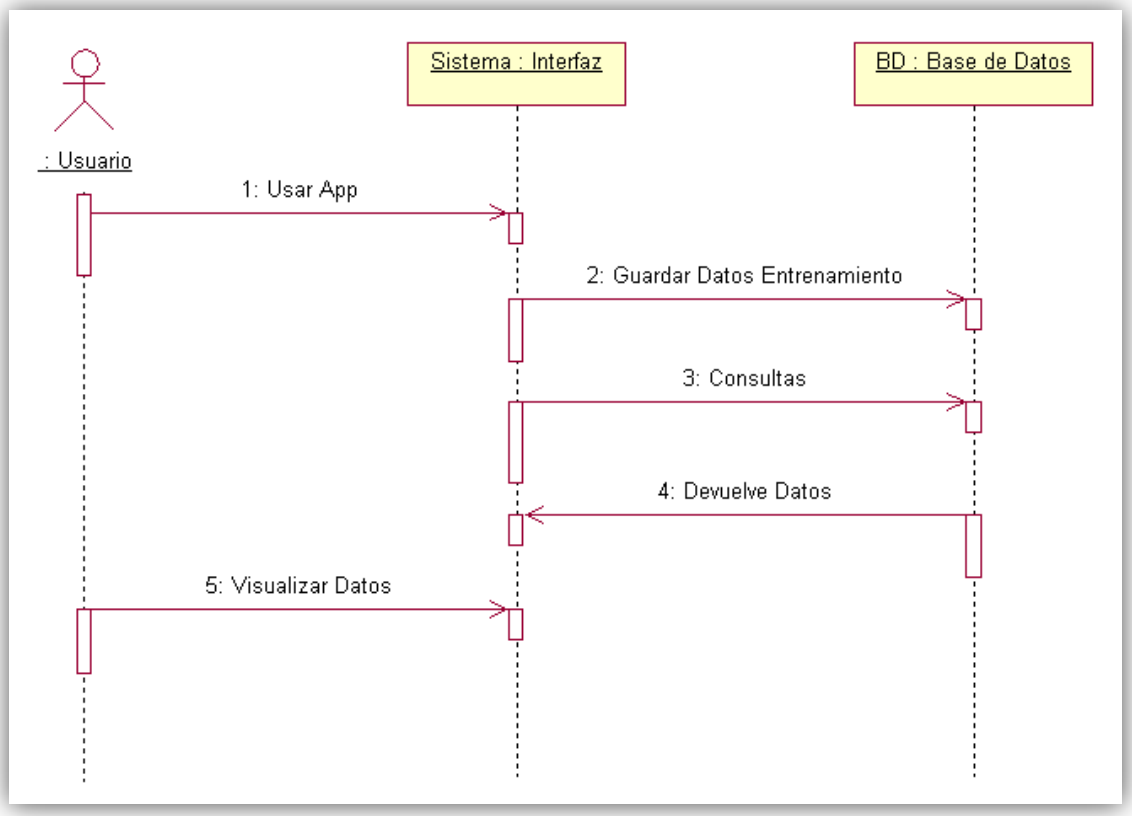



Ilustración 14 - Diagrama de secuencia, Interacción Cliente con la App

### *3.1.2. Los usuarios insertan datos en la App.*

En esta gráfica se muestra como un usuario es capaz de agregar una nueva rutina a la aplicación.

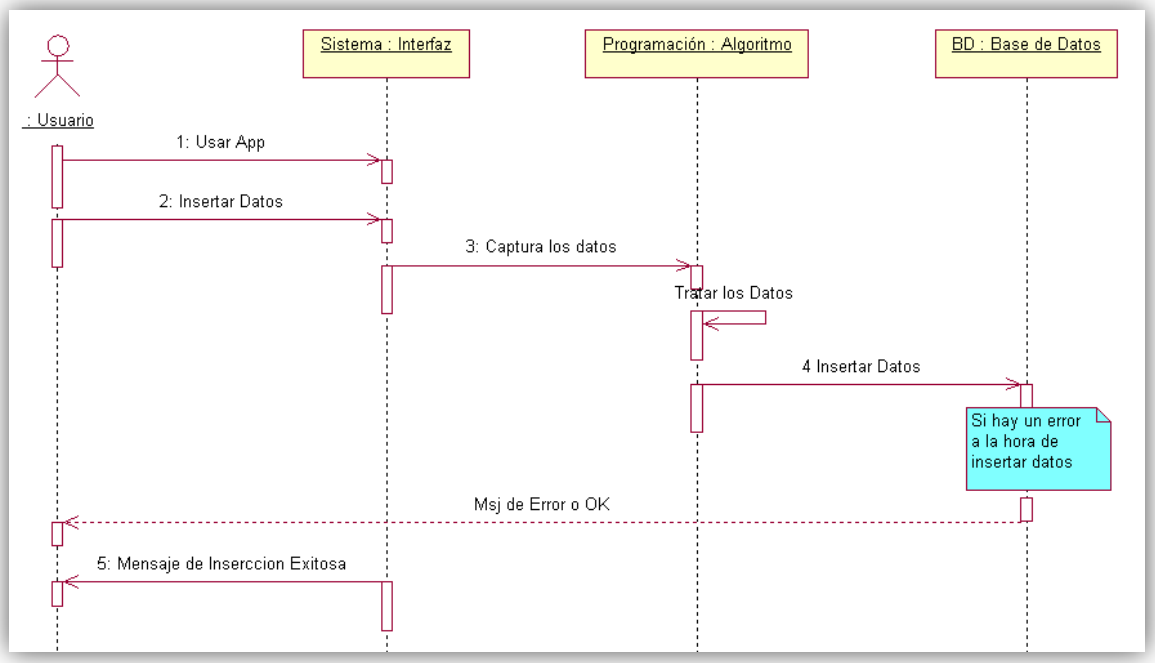



**Ilustración 15 - Diagrama de secuencia, Clientes insertan datos en la App**

### *3.1.3. Los usuarios Editan los datos del Entrenamiento*

En esta gráfica enseño como un usuario puede editar una rutina que previamente ha sido creada en su aplicación.

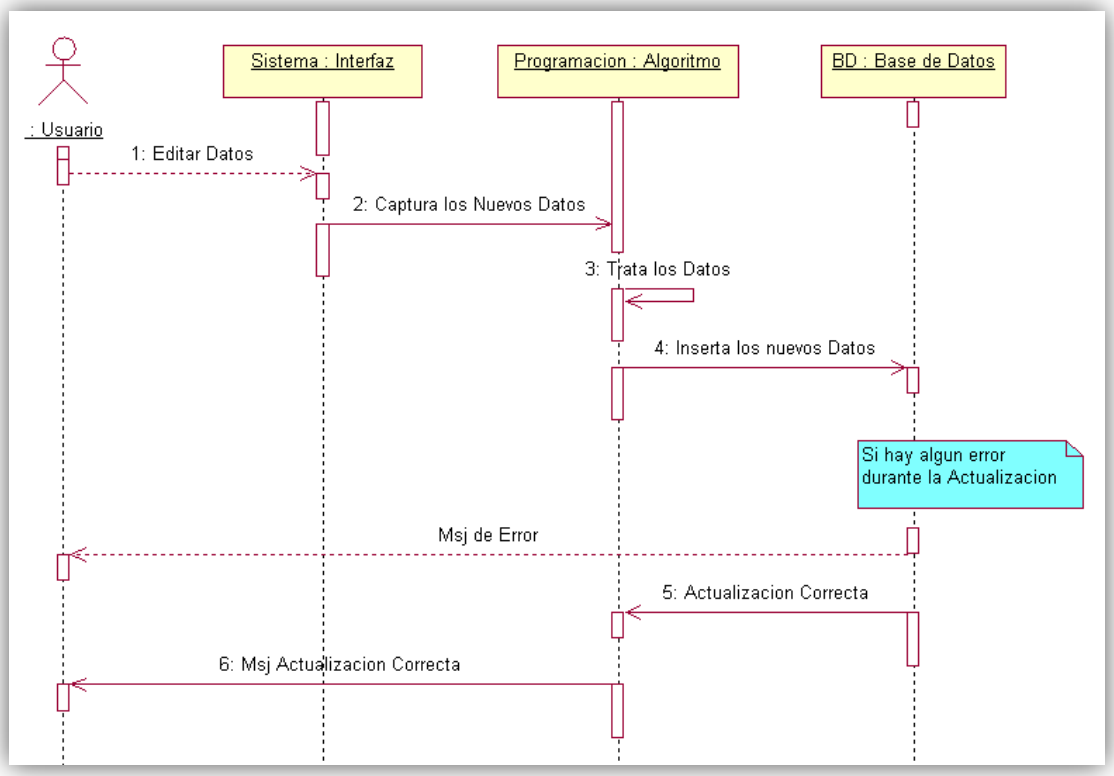



**Ilustración 16 - Diagrama de secuencia, Cliente edita el ejercicio seleccionado**

### *3.1.4. Los usuarios eliminan un entrenamiento*

En este apartado, enseño como un cliente puede eliminar un ejercicio que previamente existía en sus rutinas (Ejercicios).

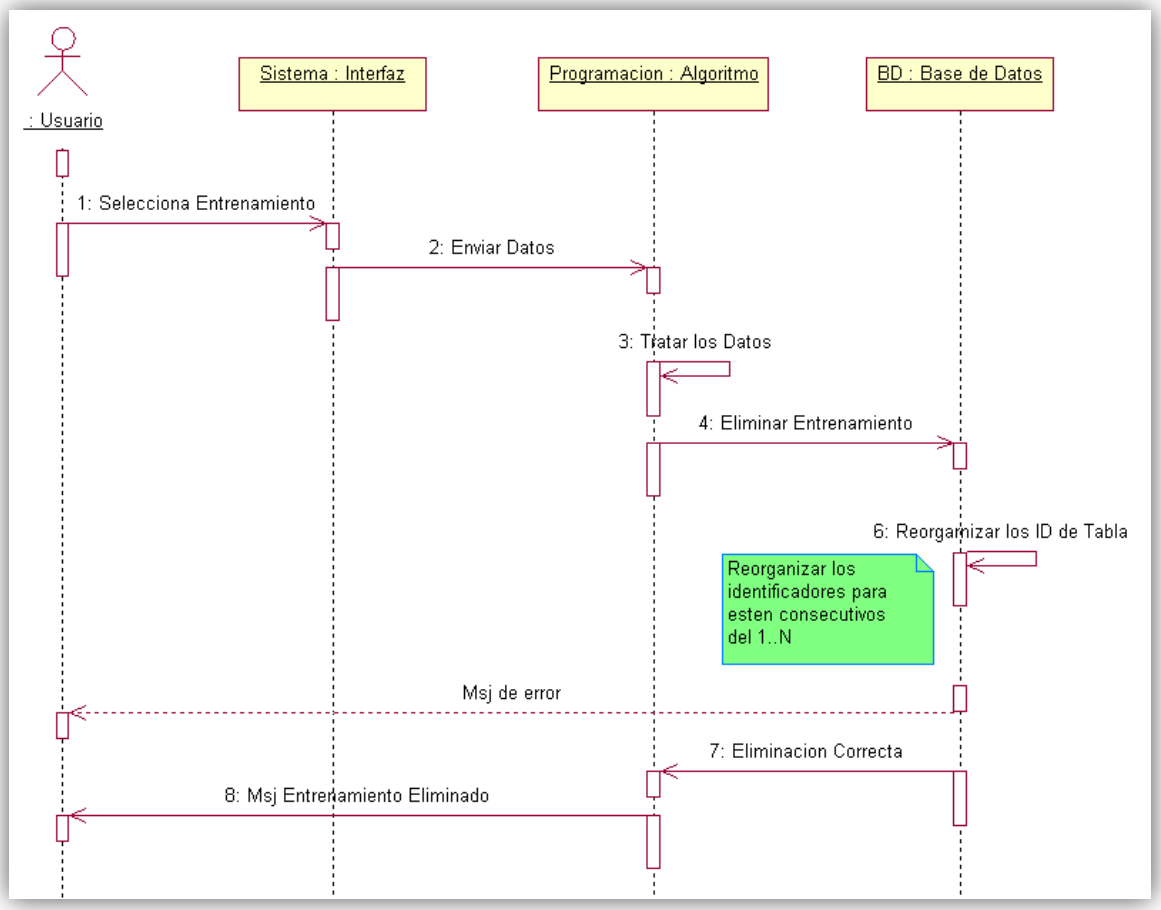



**Ilustración 17 - Diagrama de secuencia, Cliente elimina un ejercicio de la App**

### *3.1.5. Los usuarios miran su progreso*

En este punto, detallo como un deportista puede mirar su progreso a través de una gráfica estadística, la cual le indicará su mejora correspondiente en una línea de tiempo con respecto al peso ganado por sus entrenamientos diarios.

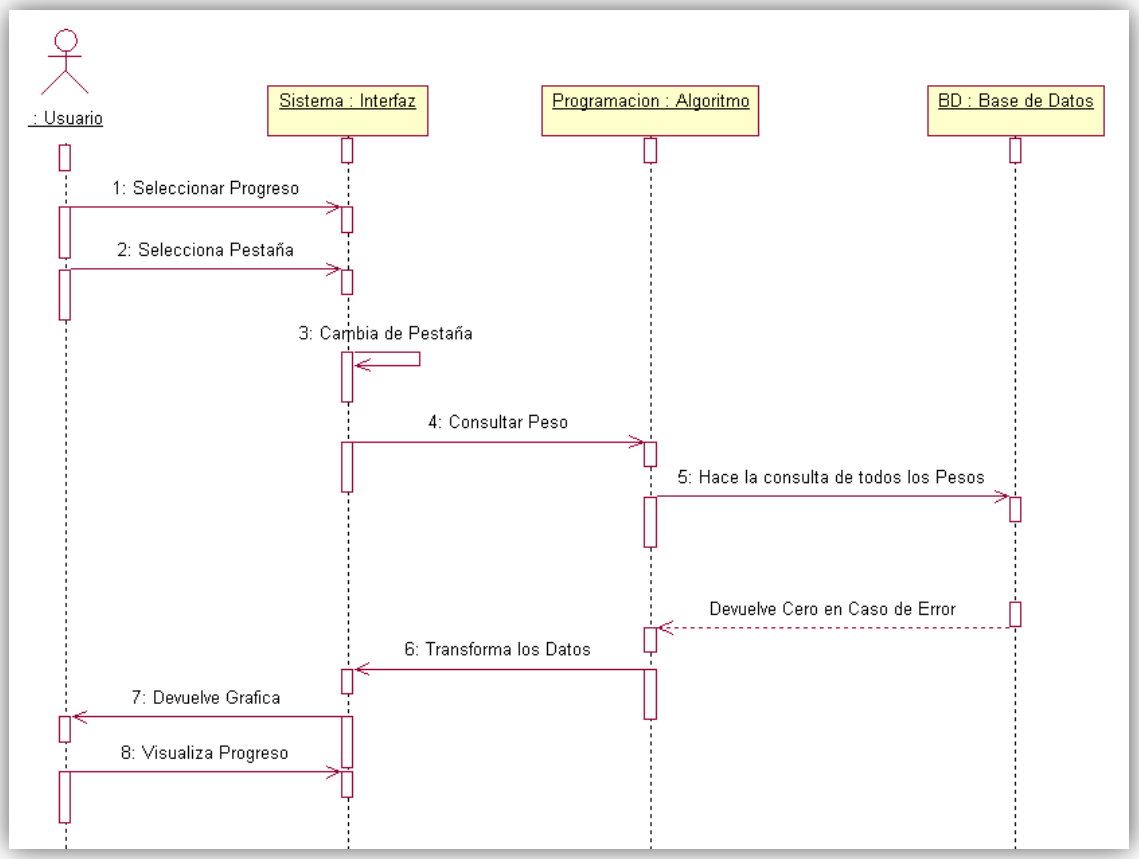



**Ilustración 18 - Diagrama de secuencia, Cliente visualiza su progreso**

### *3.1.6. Los usuarios consultan los ejercicios que de su interés.*

En este diagrama de secuencia, cada usuario puede consultar los ejercicios existentes a cada grupo muscular específico y de esta forma enseñar cómo se debe ejecutar el ejercicio y también cómo se le conoce en el mundo deportivo.

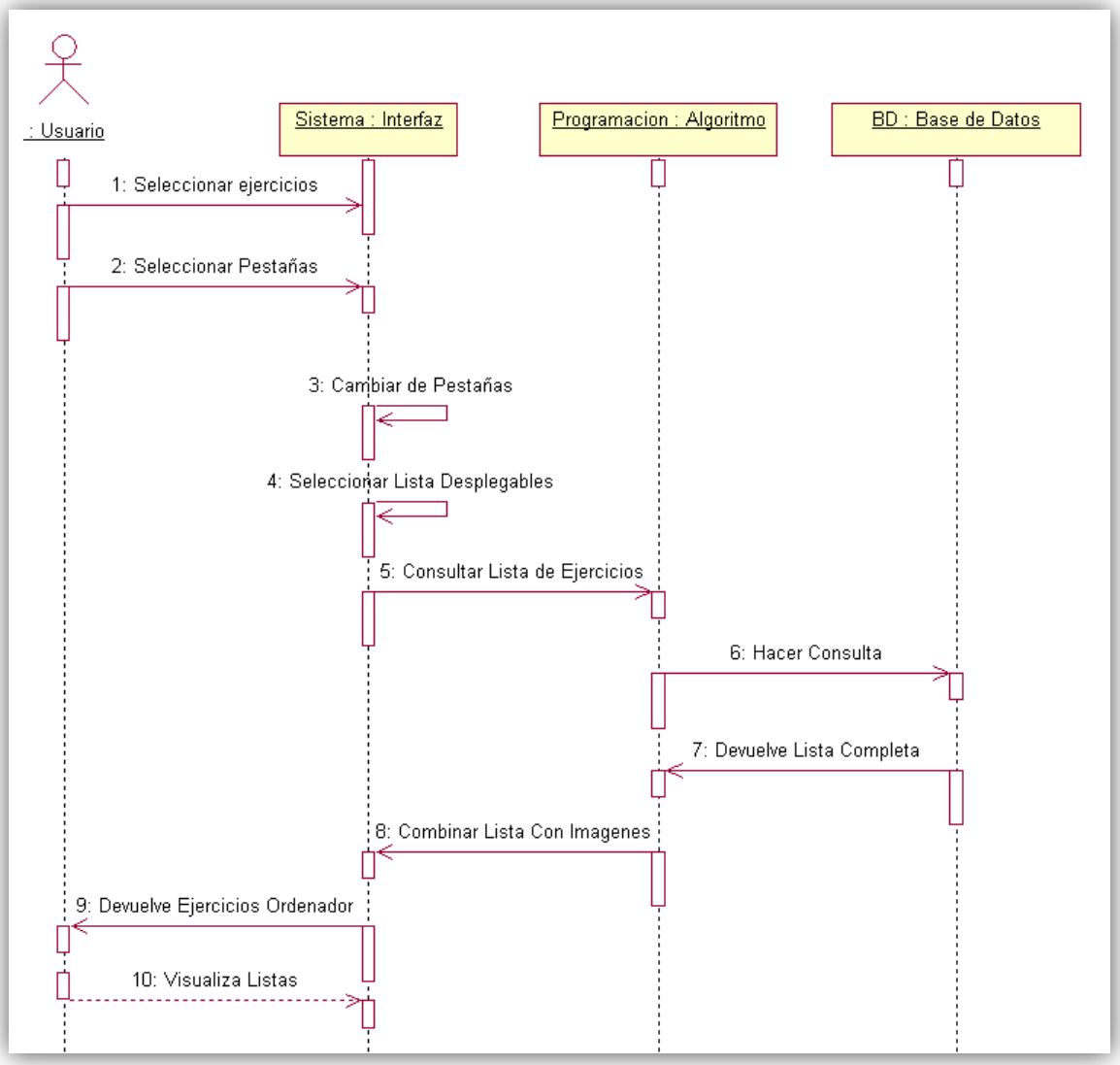



**Ilustración 19 - Diagrama de secuencia, Consultar formas en que se ejecuta un ejercicio**

### *3.1.7. Los usuarios seleccionan exportar sus entrenamientos.*

Cada usuario es capaz de exportar su entrenamiento creado si así lo desea, aunque con la condición de que previamente haya creado una rutina.

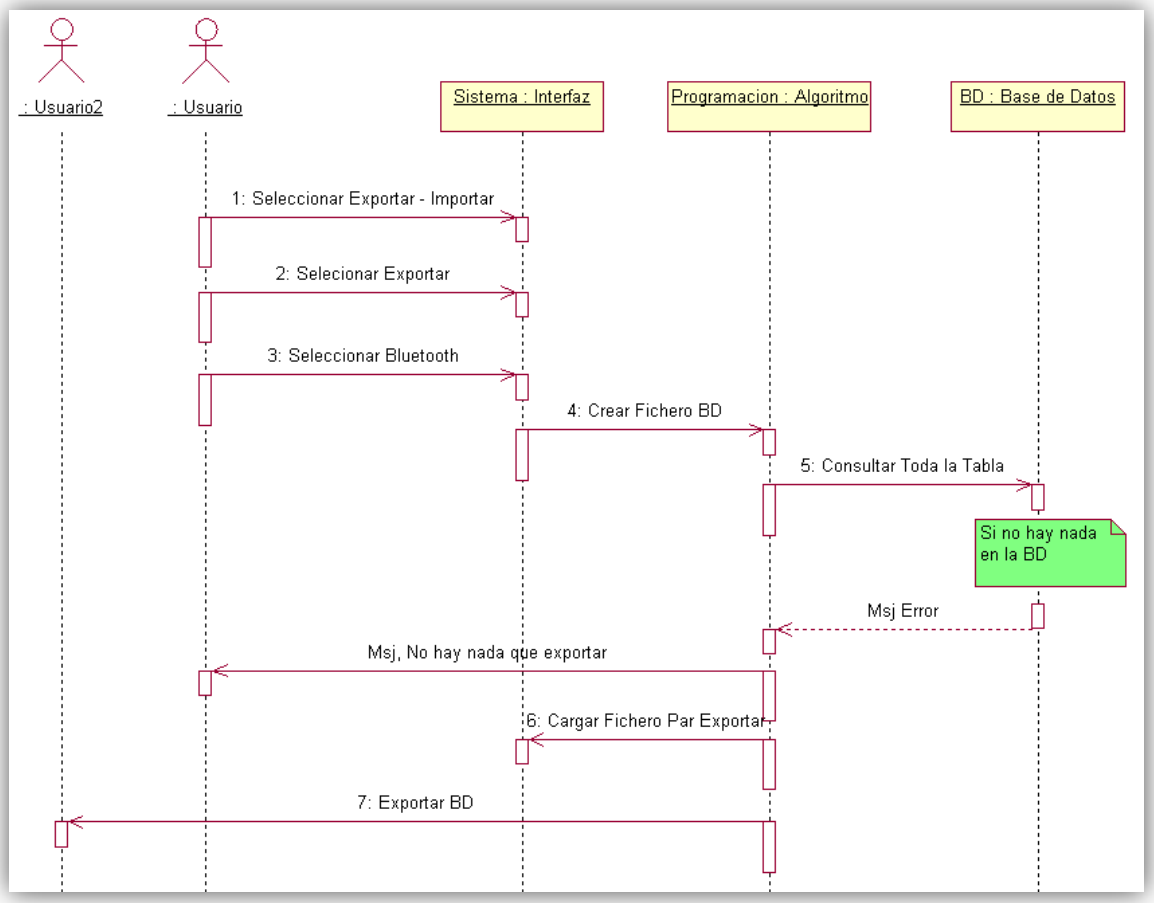



Ilustración 20 - Diagrama de secuencia, Exportar entrenamiento

### *3.1.8. Los usuarios seleccionan Importar.*

Esta opción sirve para importar una tabla, previamente adquirida por medio de esta *App*.

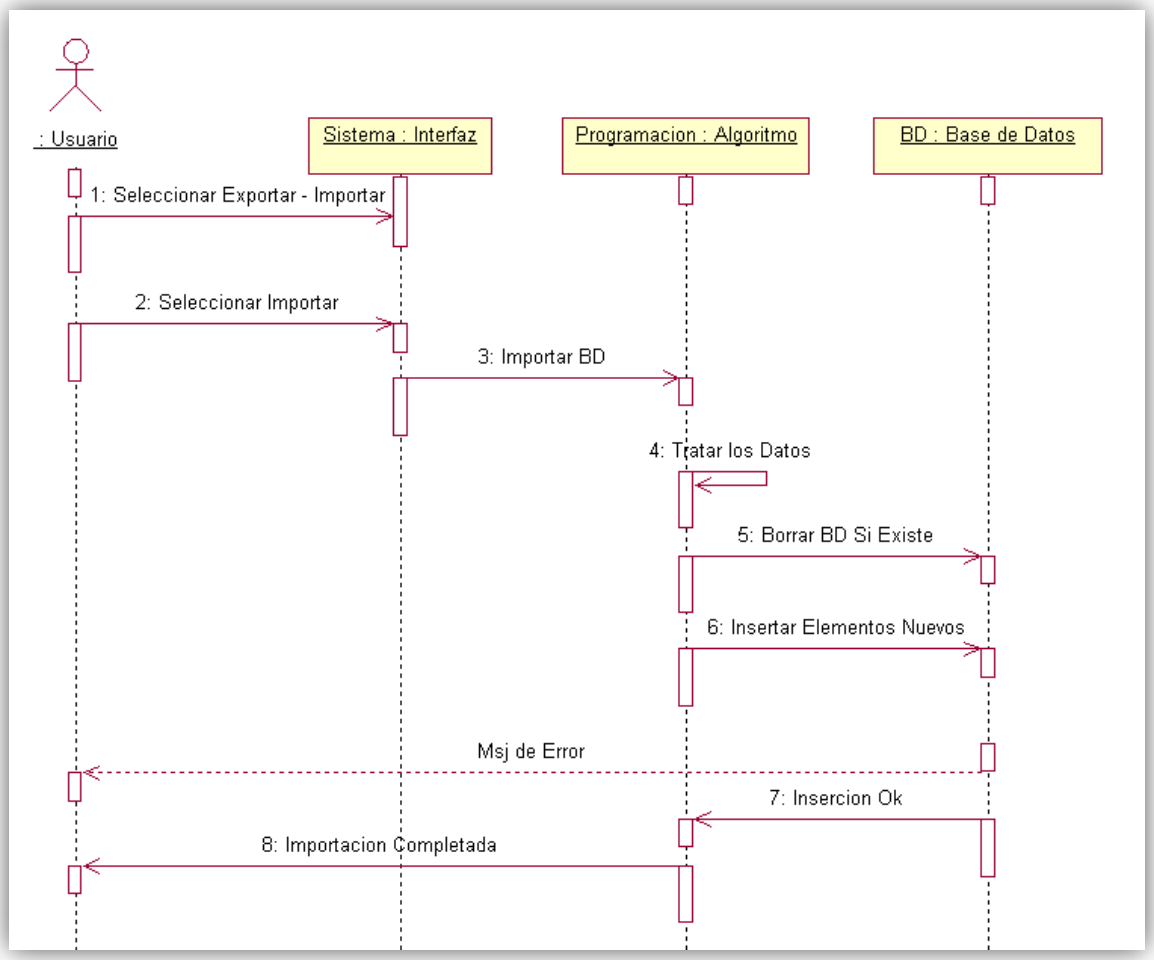



**Ilustración 21 - Diagrama de secuencia, Importar entrenamiento**

### *3.1.9. Los usuarios hacen la encuesta y los envía*

En este aparatado, los usuarios hacen una pequeña encuesta. La cual me servirá como referencia para obtener posibles modificaciones o introducir nuevas ideas.

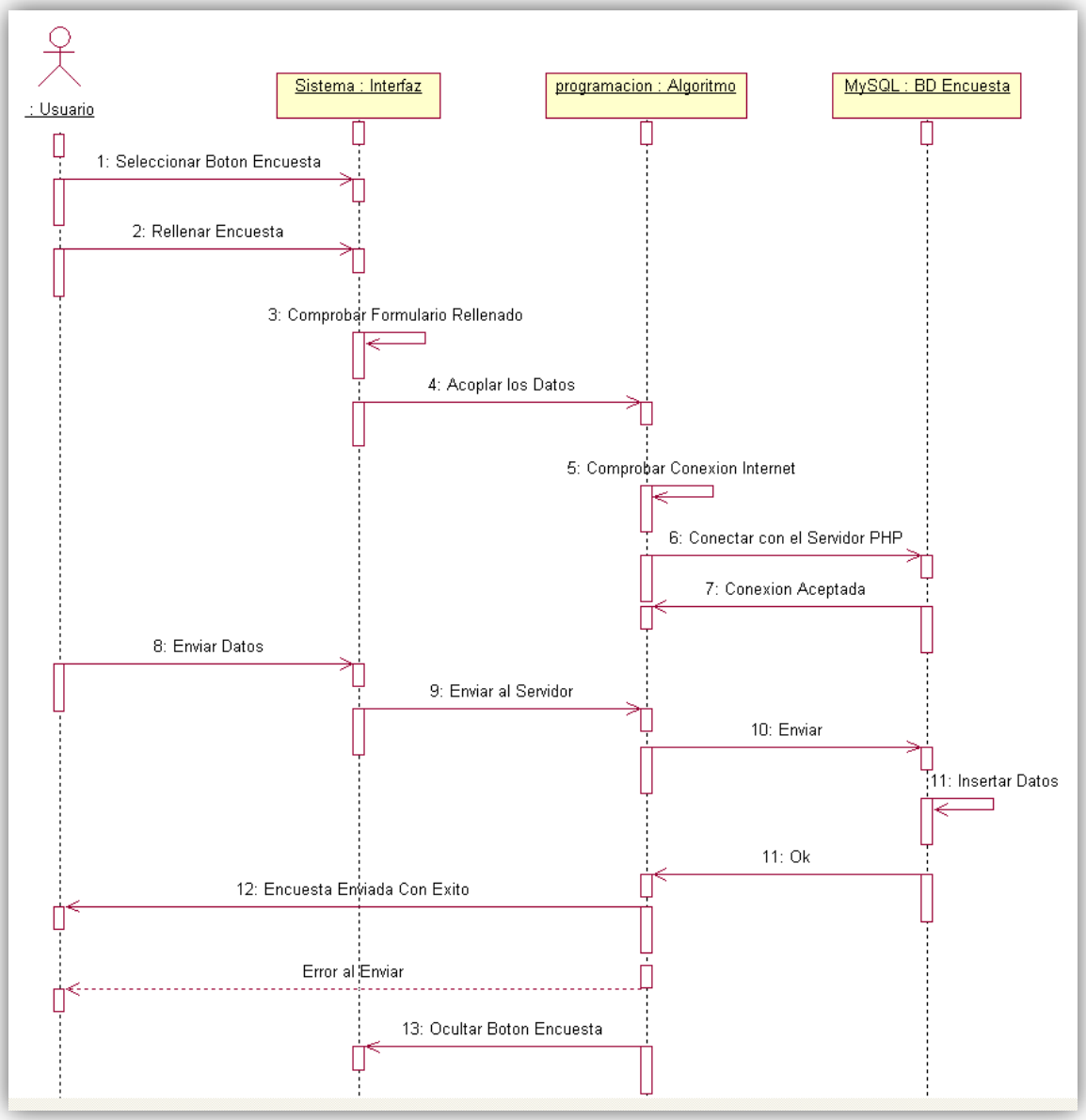



**Ilustración 22 - Diagrama de secuencia, Hacer y enviar encuesta**

### *3.1.10. Los Usuarios pulsan el botón IMC.*

Esta opción sirve para que los usuarios puedan calcular su índice de masa corporal y así puedan comprobar el estado de su salud.

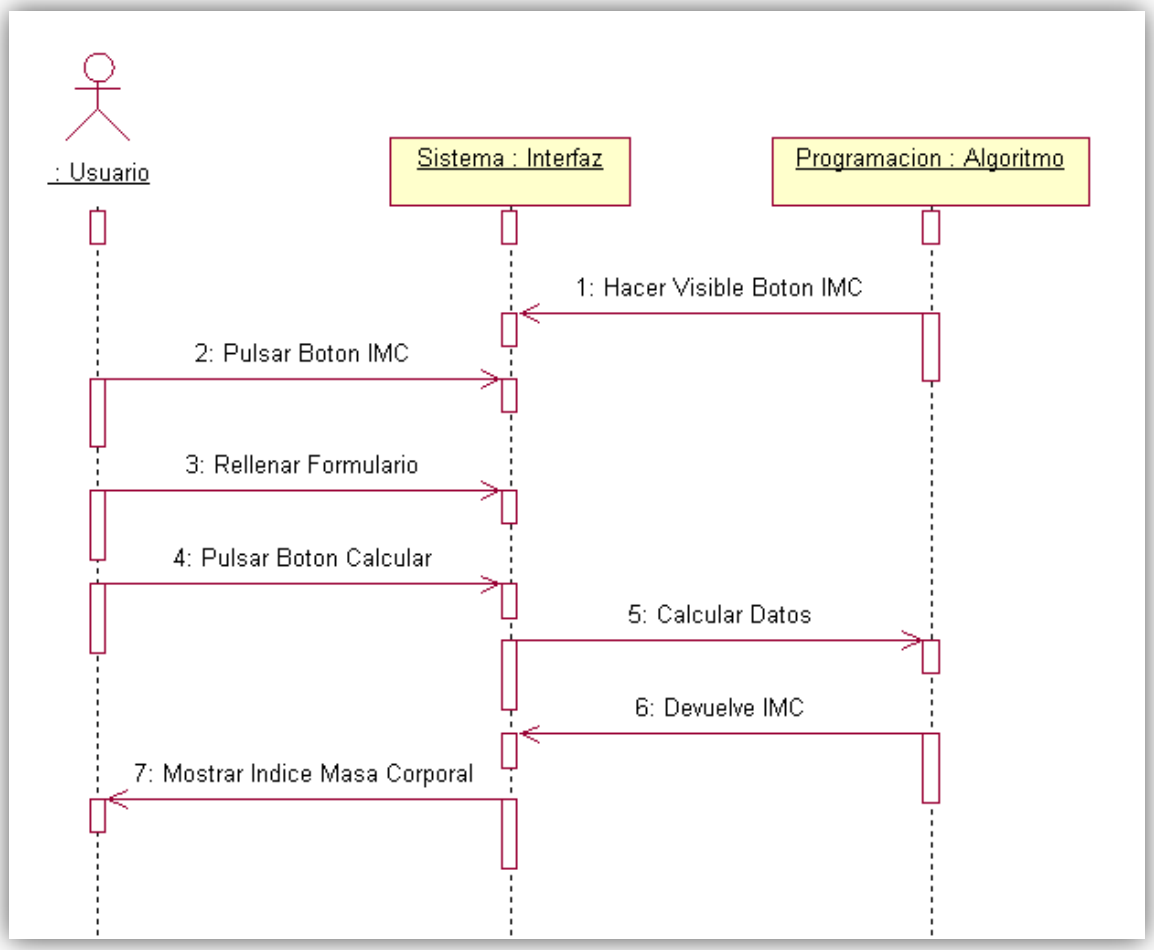



**Ilustración 23 - Diagrama de secuencia, Calcular el Índice de masa corporal**

### *3.1.11. Los usuarios seleccionan el botón compartir*

Esta opción sirve para publicar esta *App* con los amigos de los usuarios y de esta manera ganar publicidad mediante las redes sociales.

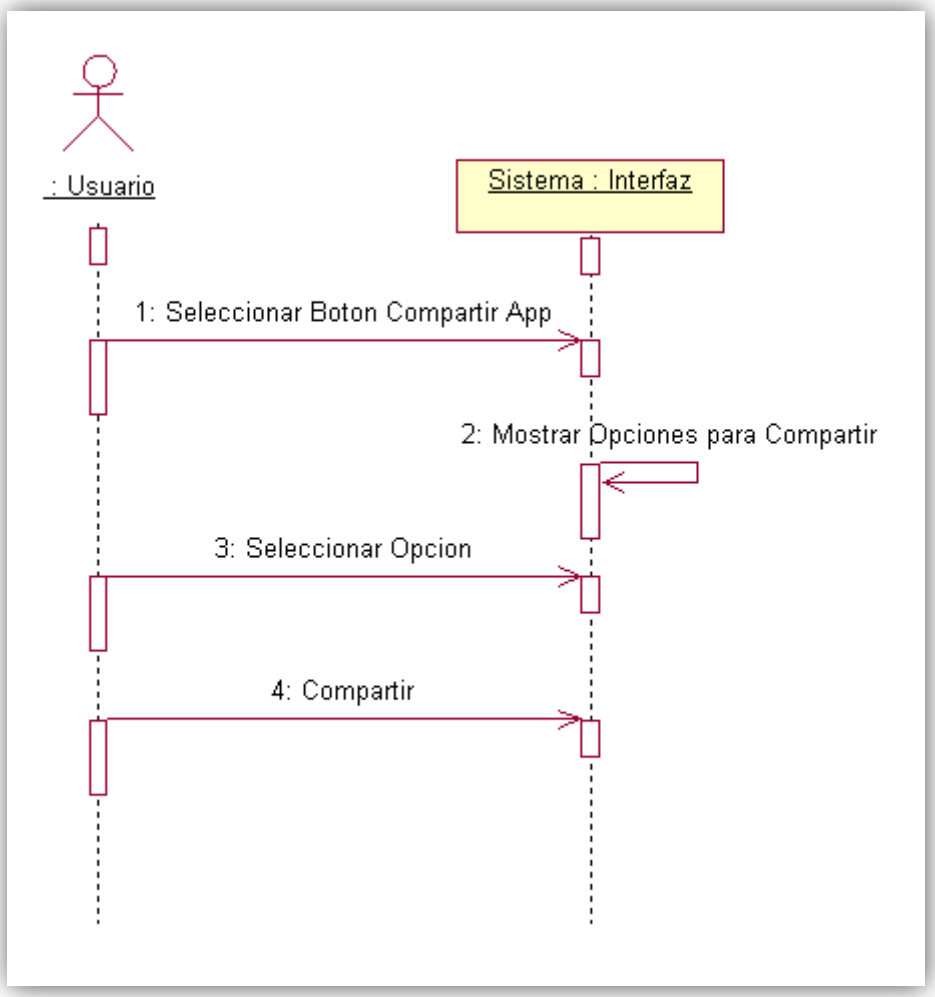



**Ilustración 24 - Diagrama de secuencia, Compartir App en las redes sociales**

## 3.2. Diagrama de Clases

En este punto, voy a mostrar los diagramas de clases que componen mi estructura de Clases y paquetes de programación en Java.

### *3.2.1 Diagrama de Clases de Paquetes Java*

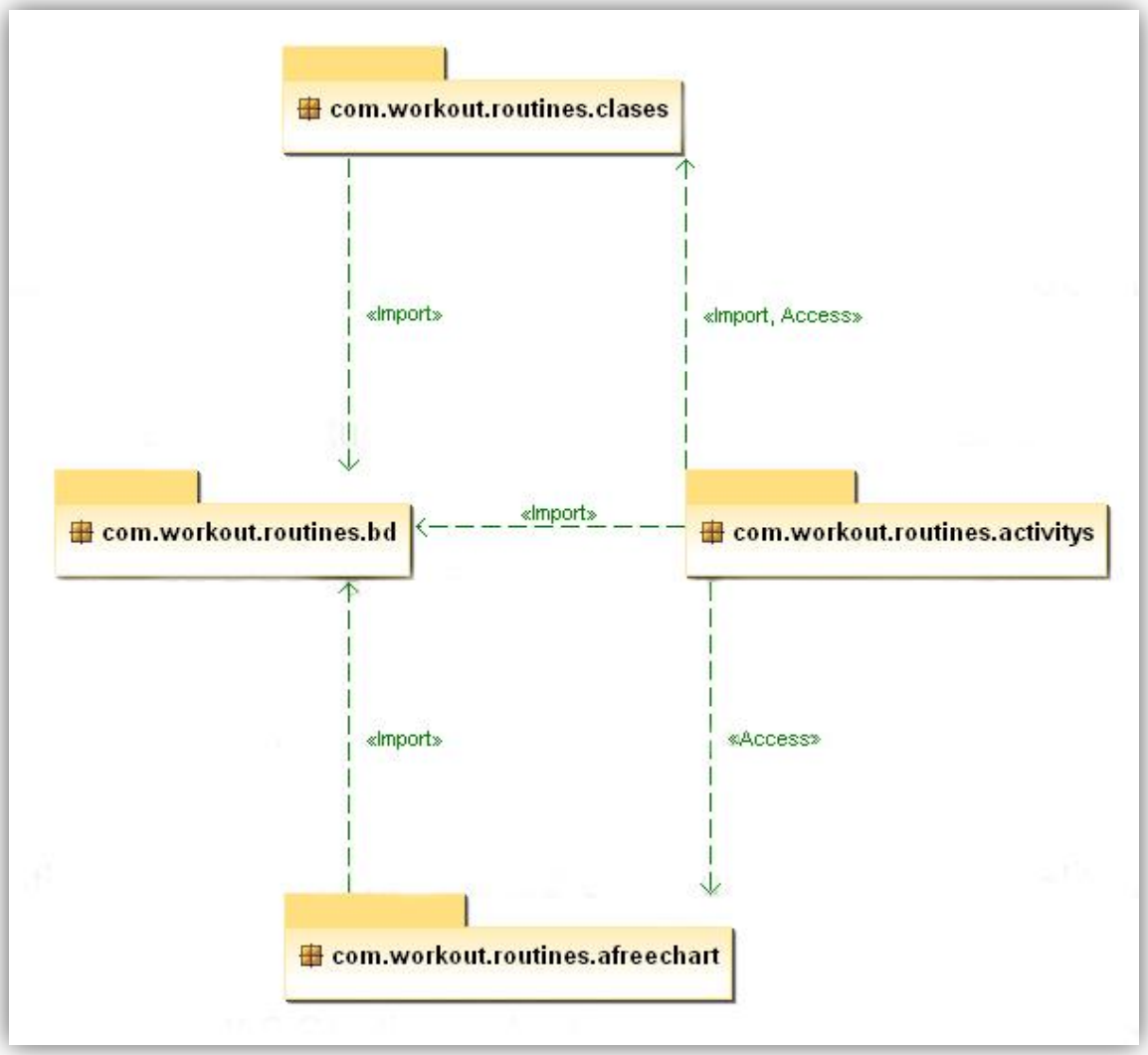



Ilustración 25 - Diagrama de clases de paquetes Java

### 3.2.2. Diagrama de Clases, Paquete Activitys

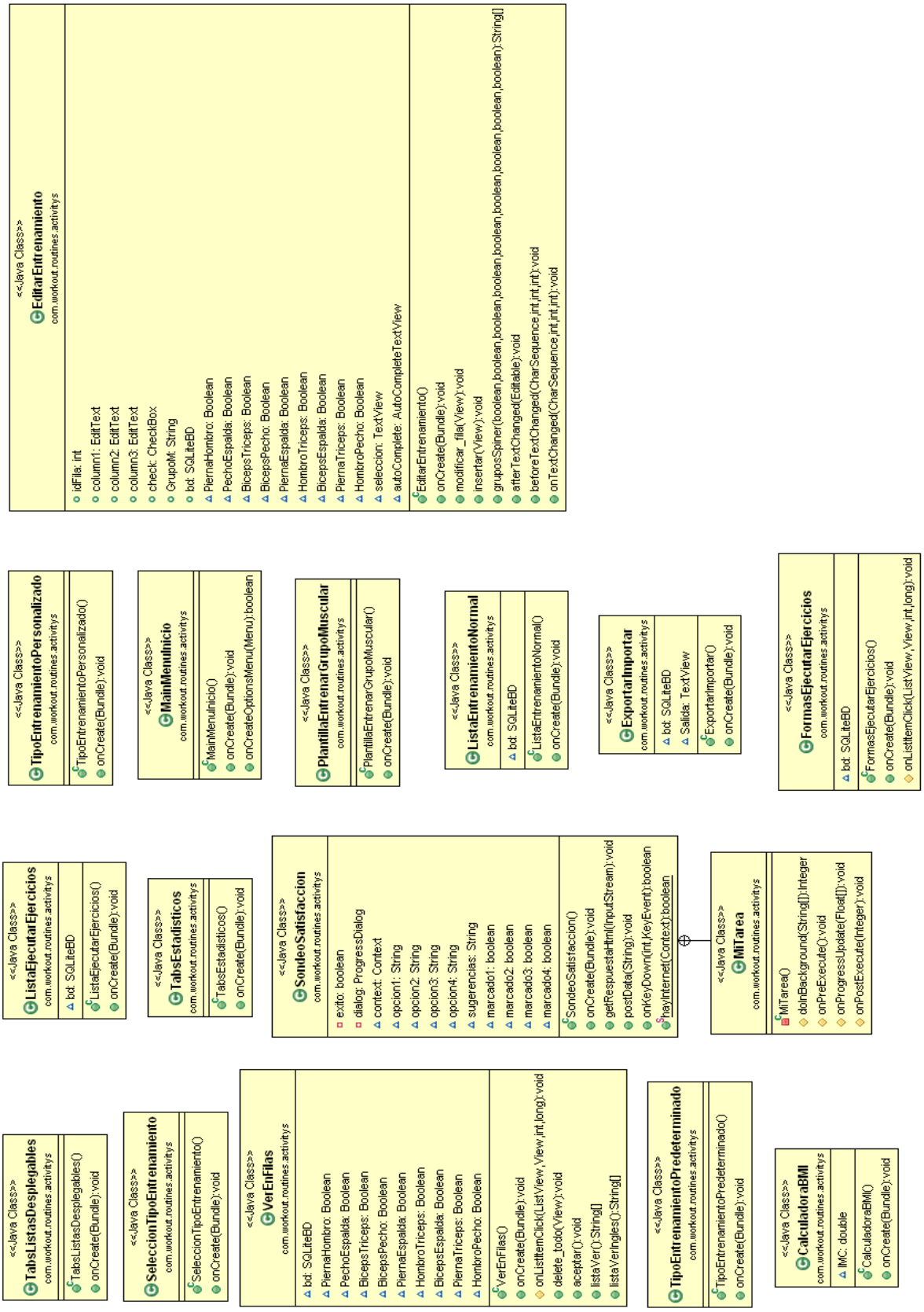



Ilustración 26 - Diagrama de clases, Clases del paquete Activitys

### 3.2.3. Diagrama de Clases, Paquete AFreeChart

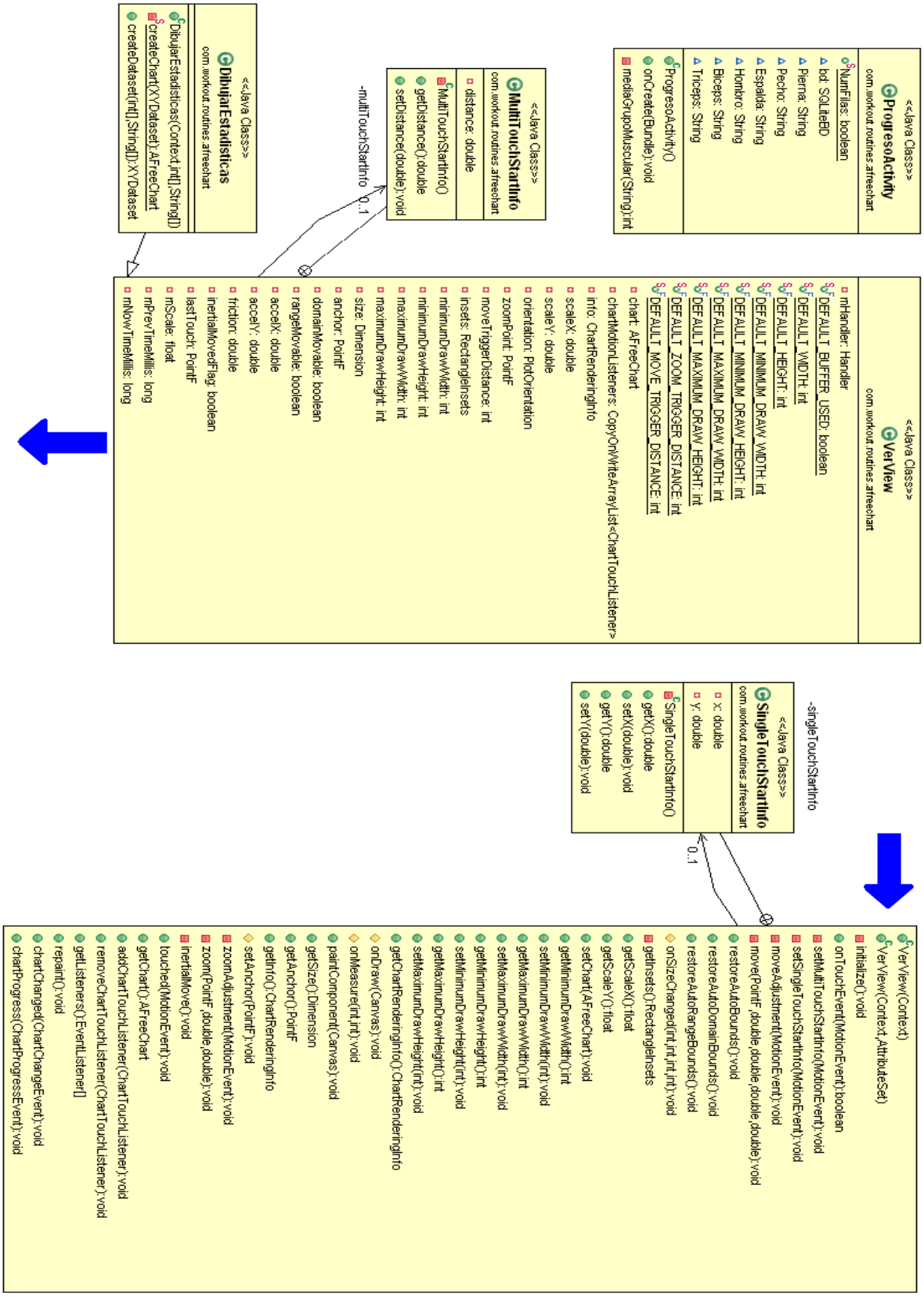



Ilustración 27 - Diagrama de clases, Clases del paquete AFreeChart

### 3.2.4. Diagrama de Clases, BD SQLiteBD

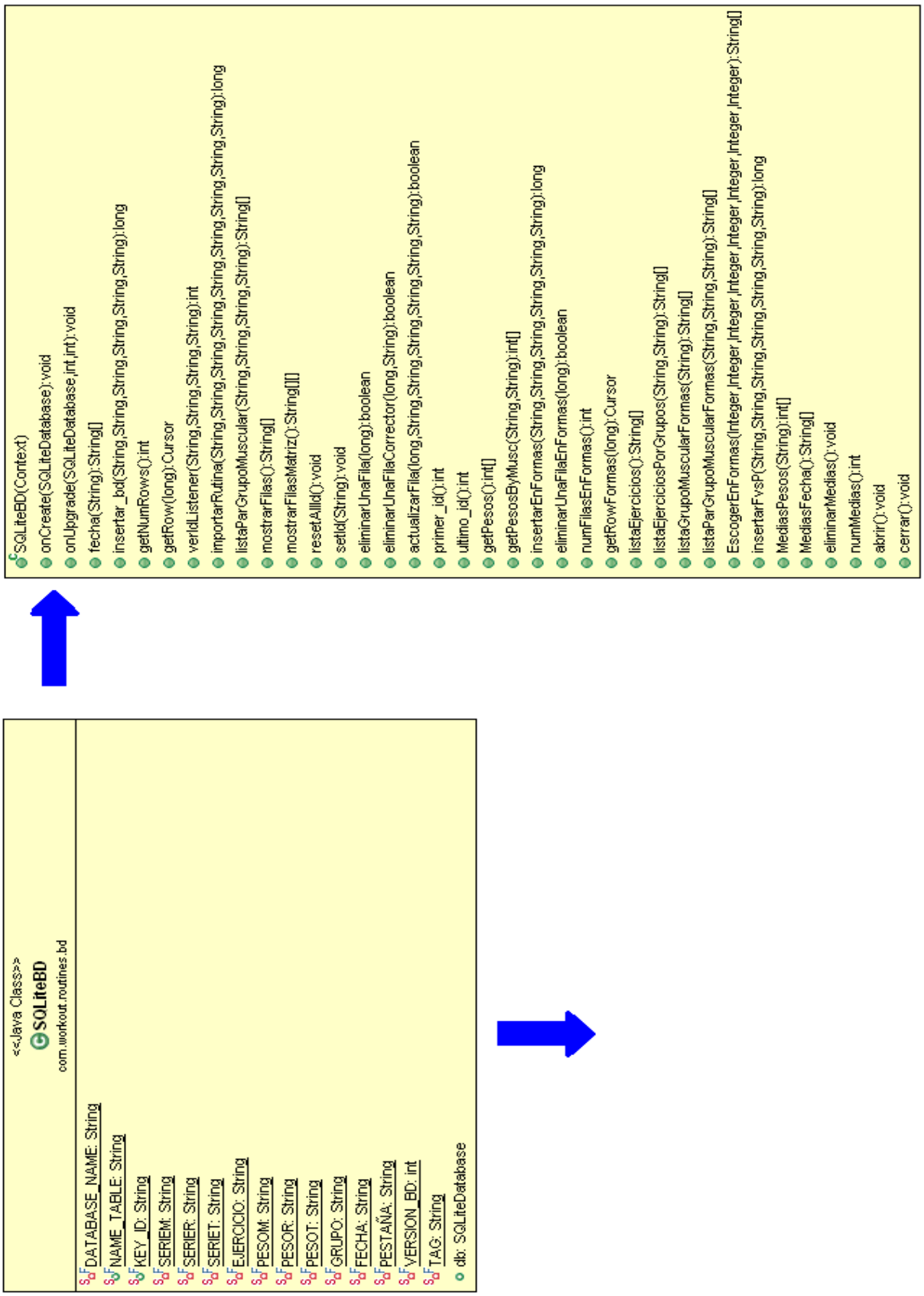



**Ilustración 28 - Diagrama de clases, Clases del paquete BD, SQLiteBD**

### 3.2.5. Diagrama Clases, Paquete de Clases – Clases

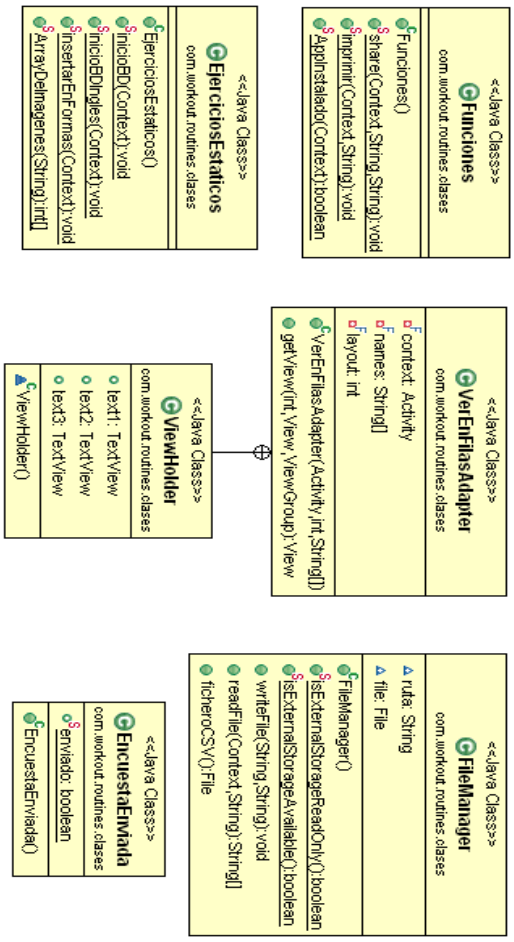



Ilustración 29 - Diagrama de clases, Paquete Clases

### 3.2.6. Diagrama de Clases General De Toda la App

En este capítulo voy a mostrar el diagrama general de clases que componen todo el proyecto de esta aplicación. Las he divido en tres partes, para mostrar con más detalle y mayor calidad las partes que la componen.

### 3.2.6.1. Parte 1

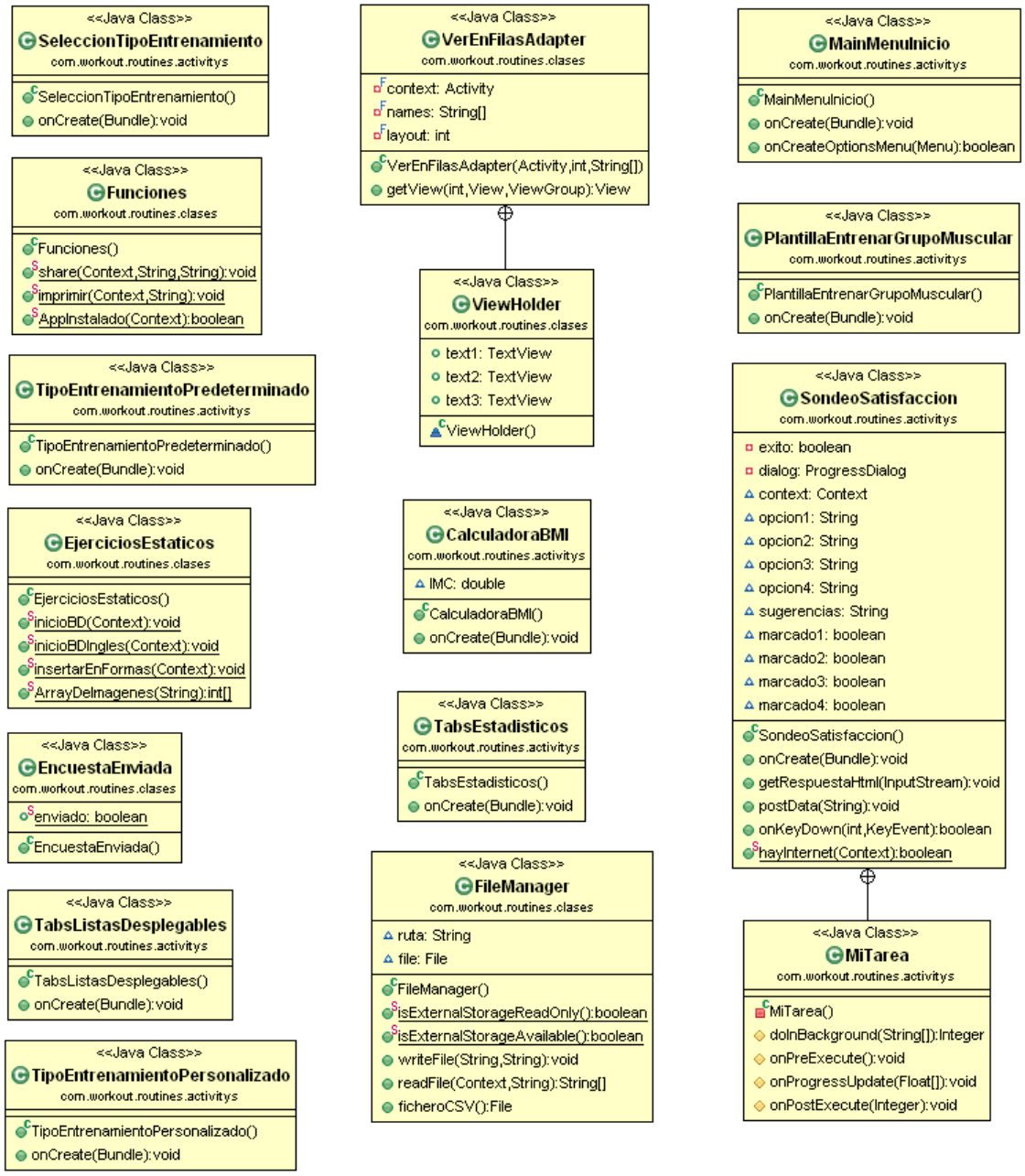



Ilustración 30 - Diagrama de clases general, parte 1

### 3.2.6.2. Parte 2

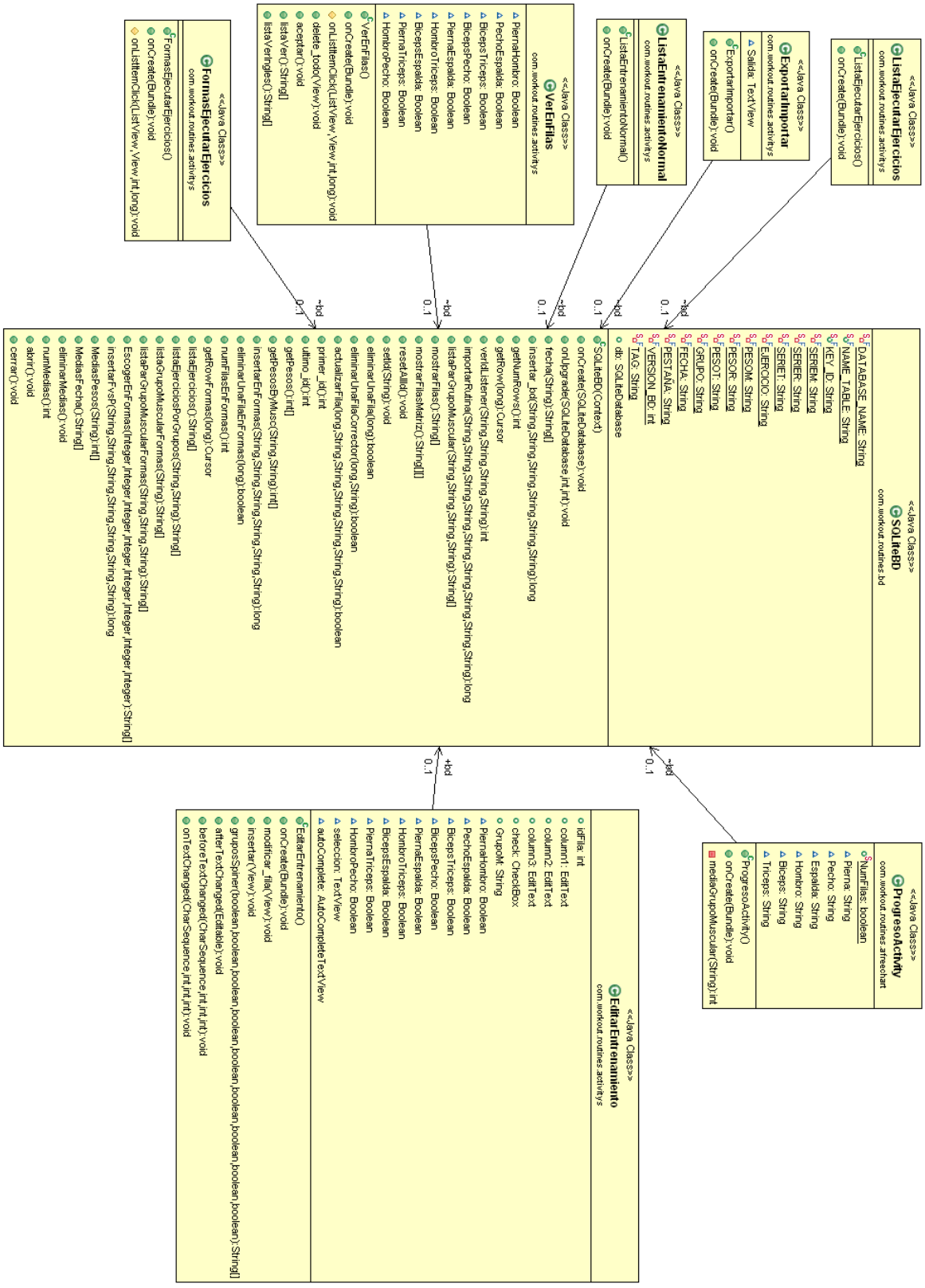


Ilustración 31 - Diagrama de clases general, parte 2

### 3.2.6.3. Parte 3

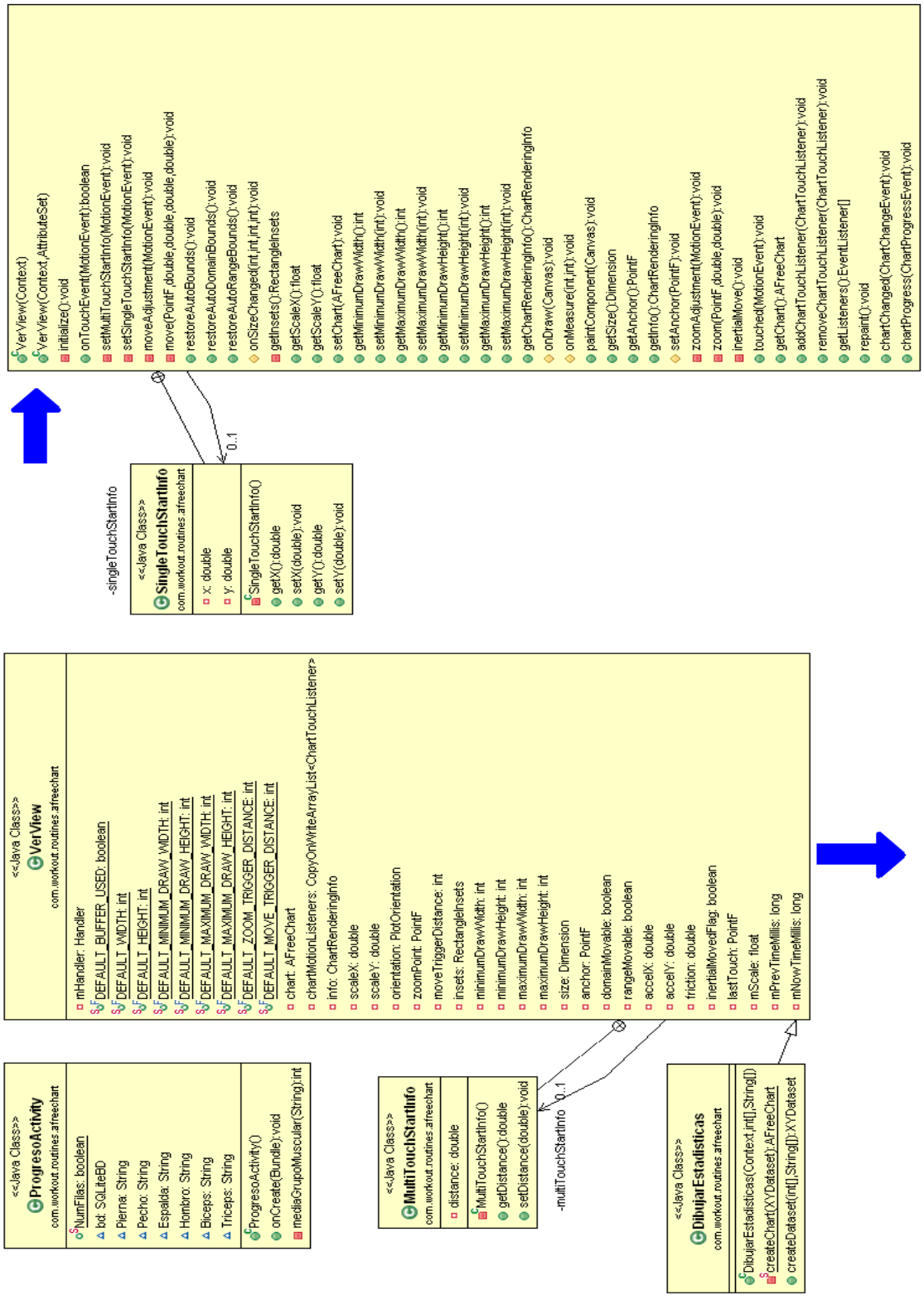



Ilustración 32 - Diagrama de clases general, parte 3

# 4. IMPLEMENTACIÓN

En este capítulo detallaré como ha sido el proceso de desarrollo e implementación de esta *App*, para llegar a obtener el producto final "Workout Routines".

## 4.1. Plataforma de Desarrollo

La plataforma seleccionada para este proyecto ha sido *Android* de *Google* por las siguientes ventajas:

- *Google* tiene una licencia de software y desarrollo libre para poder programar en *Android*. Esto permite a los desarrolladores usar sus librerías, para crear aplicaciones en dispositivos con sistema operativo *Android* y luego poder publicarlas o distribuirlas a todas las personas que lo deseen.

- Es la tecnología más usada junto con las de *Apple* y *BlackBerry*, que son de pago.

- El *SDK (Software Development Kit)* es muy completo y está disponible para cualquier desarrollador que desee descargarlo.

- Existen numerosas ayudas para comenzar a crear aplicaciones en Android, desde las *API* completas con todas las clases y paquetes, hasta herramientas de programación y un perfecto emulador para poder realizar pruebas.

- Conocimiento aprendido sobre el lenguaje de programación Java, aunque Android sólo utiliza la sintaxis y la semántica de Java, pero no incorpora en su totalidad las bibliotecas de clases de *Java* y *APIs*.

- Una gran comunidad de desarrolladores *Android*, lo cual permite acceder a muchísima información existente en la web.

- Servidor propio de *Google* para subir y descargar tus aplicaciones.

En concreto, la plataforma que utilizo para desarrollar es *Android 4.0 – Api 14*, por lo que utilizo un dispositivo móvil que usa esta versión, la cual me viene bien para hacer pruebas de ejecución. Aunque tengo un rango definido de compilación para dar compatibilidad para diferentes versiones, que va del *API 10* al *API 16*.

Por otra parte, utilizo un servidor de *Hosting* gratuito, para dar publicidad a la aplicación mediante el acceso a su propia página web, también este servicio me sirve para enviar las consultas de las encuestas de satisfacción a un servidor *PHP* con *MySQL*.

En la siguiente imagen, detallo las características que me ofrece Hostinger (www.hostinger.es)

| | |
|---|---|
| Carga del Servidor: | 4.35 |
| Carga del Servidr MySQL: | 0.15 |
| Apache Versión: | Apache/2.2.14 |
| PHP Versión: | 5.2 |
| MySQL Versión: | 5.1.61 |
| Activado: | 2012-12-03 16:06:55 |
| Estado: | ✔ Activo |
| Plan de Hosting: | Gratis |

Ilustración 33 - Características del servidor externo

El nombre que le he dado a la página, es el nombre del proyecto Android, ya que de esta forma es más fácil reconocer la página por el nombre de la *App*. La siguiente *URL* es el enlace a la página y presentación del proyecto: http://www.workoutroutines.hol.es/


Ilustración 34 - Pagina Web de la App

El diseño de la página web, es una platilla web sacada de la siguiente página http://all-free-download.com/free-website-templates/ ya que ofrece una gran variedad y la mayoría son gratuitas. Lo único que hay que hacer es rediseñar y acoplarlo a las necesidades requeridas.

## 4.2. Configuración y Creación de Proyectos Android

En este apartado voy a explicar la creación y configuración de los programas, ficheros y librerías necesarias para crear aplicaciones Android.

### *4.2.1. Descargar Eclipse*

Para descargar *Eclipse*, necesitamos ir a la siguiente URL de descarga: http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops4/R-4.2.1-201209141800/eclipse-SDK-4.2.1-win32.zip

Una vez descargado, hay que descomprimir el fichero y abrir por primera vez la aplicación.

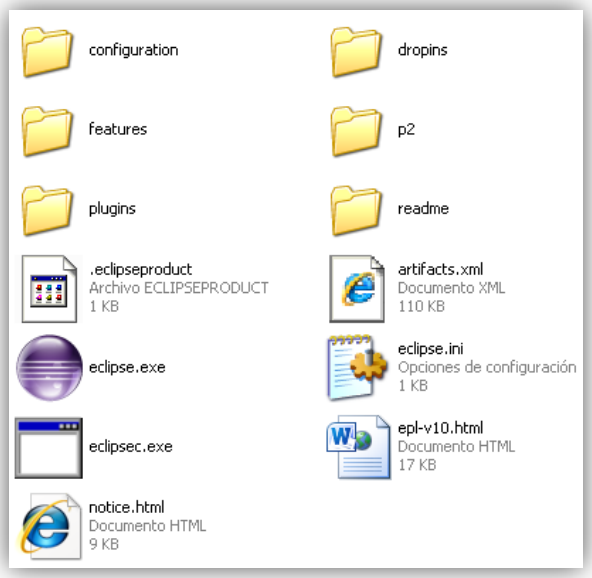



Ilustración 35 - Directorios y archivos que componen el IDE Eclipse

Lo siguiente que se carga son los módulos que hacen funcionar *Eclipse* y mientras nos muestra una imagen de bienvenida de la versión del Eclipse que estamos usando.

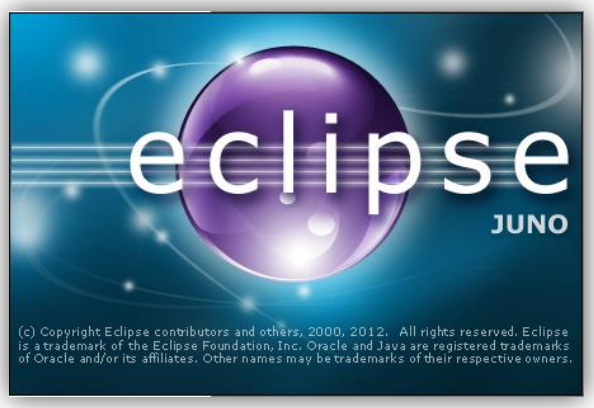


Ilustración 36 - IDE Eclipse versión JUNO

Luego hay que configurar el espacio de trabajo, ya que aquí es donde se van a aguardar nuestros proyectos *Android* o *Java*. Por defecto dejamos la que nos recomienda Eclipse.

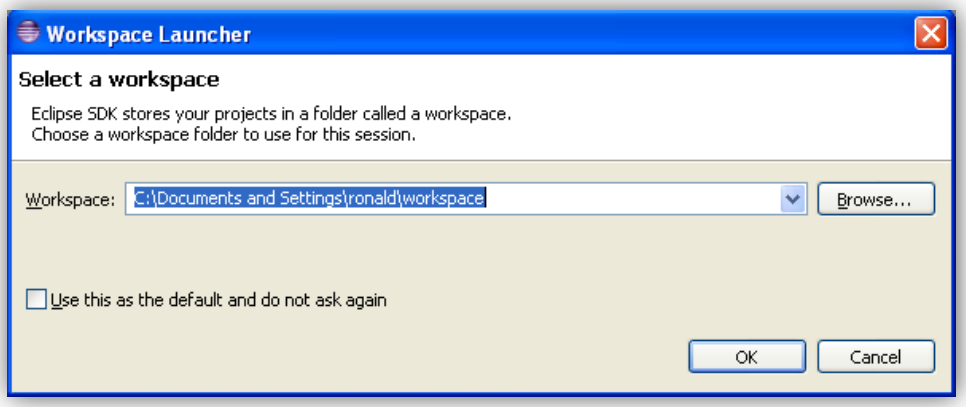



Ilustración 37 - Ruta en donde se guardaran nuestros proyectos

Y nos muestra nuestro espacio de trabajo.

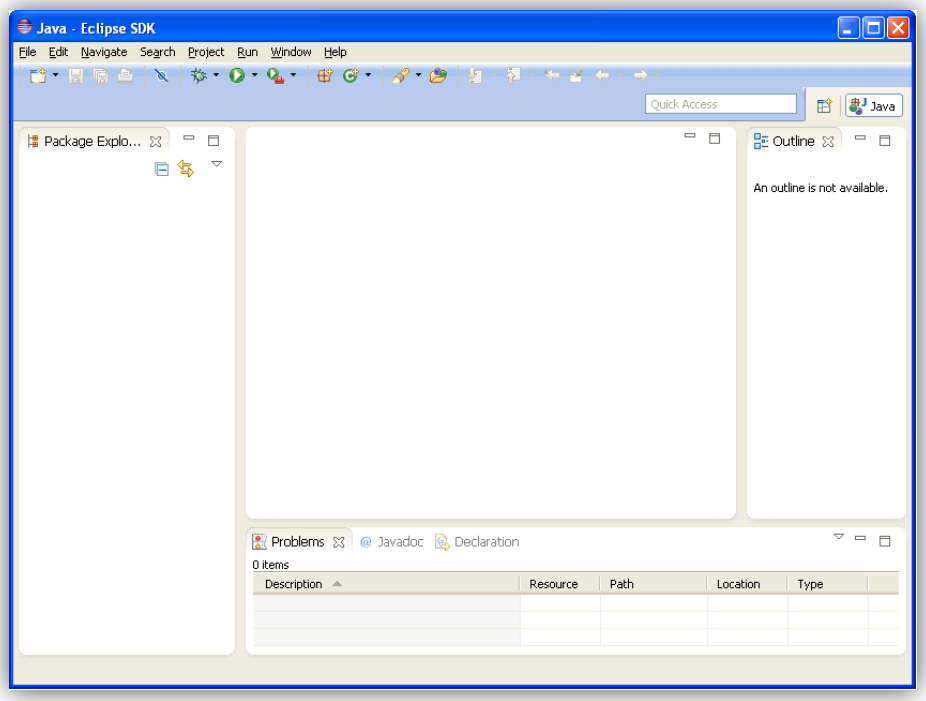



Ilustración 38 - Espacio de desarrollo en Eclipse

### 4.2.2. Descargar SDK Android

Una forma fácil de hacerlo es ir a la barra de herramientas: ***Help*** / ***Install New Software***. E insertar la siguiente URL. : https://dl-ssl.google.com/android/eclipse/

La *url* fue sacada de la documentación de *Android Developer*: http://developer.android.com/sdk/installing/installing-adt.html

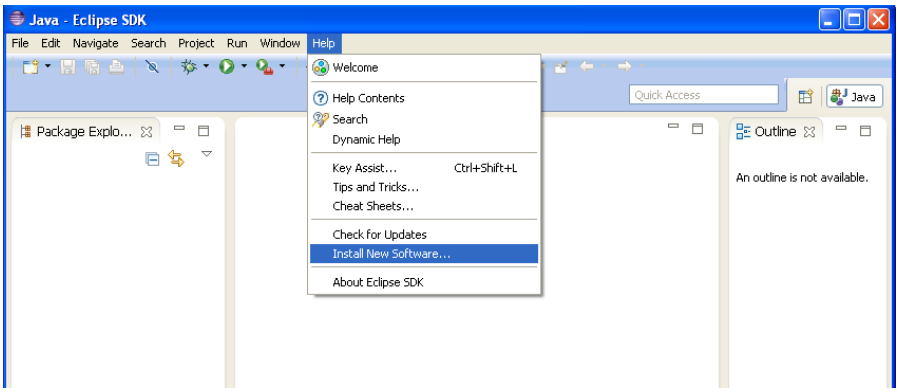



Ilustración 39 - Selección de Instalación de una nueva API

Lo siguiente es pulsar el botón agregar

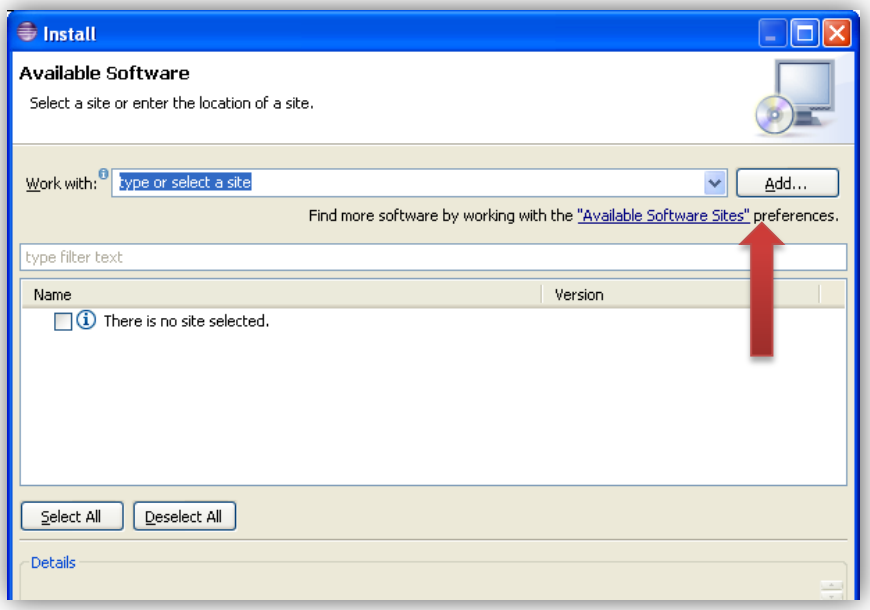



Ilustración 40 - Pulsar botón agregar

Después de pulsar el botón agregar, ponemos la URL y damos a Ok.

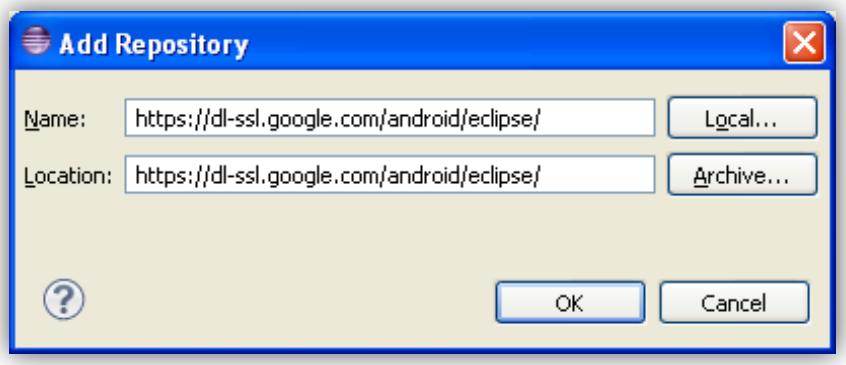



Ilustración 41 - Agregar la dirección de descarga de Android

Seleccionamos todas las opciones y damos a *Next*.

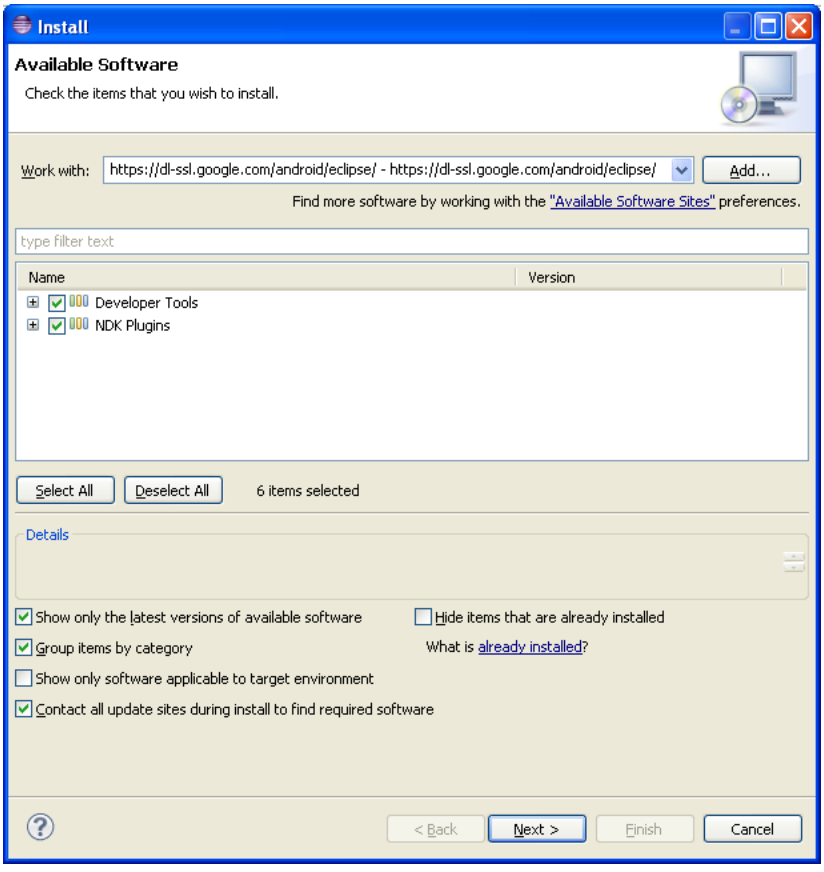



Ilustración 42 - Seleccionar todo y continuar

Esta ventana nos muestra los que vamos a instalar y continuamos con la instalación.

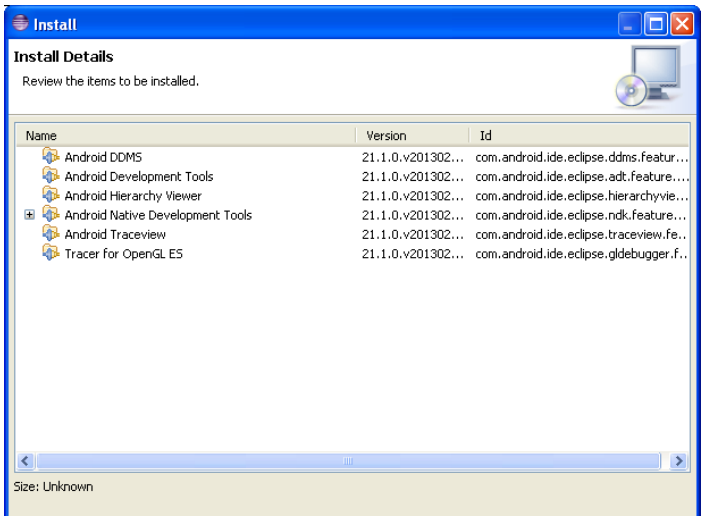



Ilustración 43 - Next

Lo siguiente es el acuerdo de la licencia. Aceptamos y continuamos.

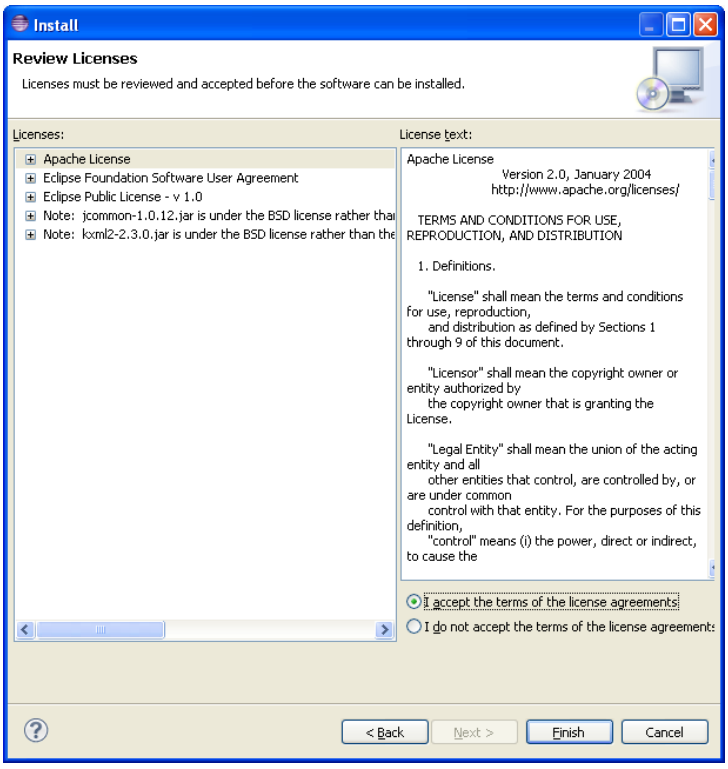



Ilustración 44 - Aceptar el acuerdo de licencia

Se nos cargara una ventana con el progreso de la instalación.

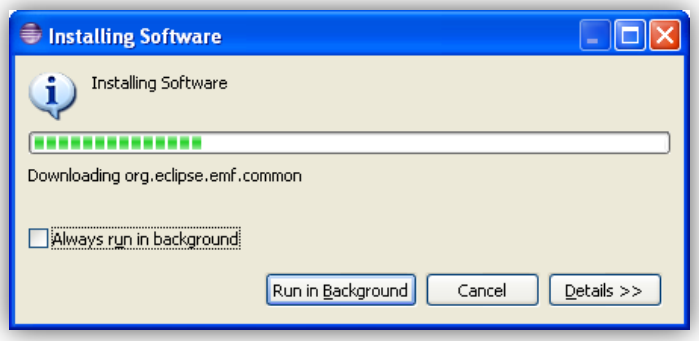



Ilustración 45 - Progreso de descargar e instalación

Lo siguiente que nos sale es un aviso de advertencia de instalación de Software malintencionado, damos a Ok, ya que es *Google*.

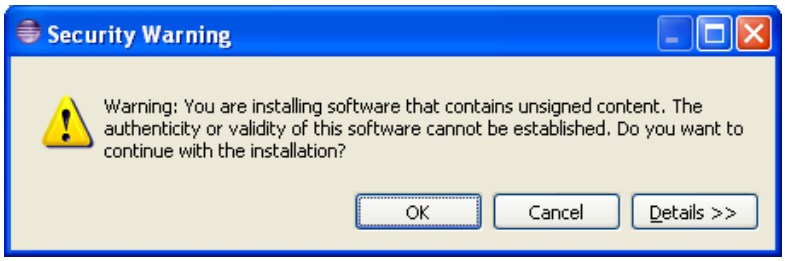



Ilustración 46 - Aceptar y continuar

La alerta de seguridad del *Firewall*, nos sale esta opción ya que hay darle permisos a *Eclipse* para que este pueda funcionar correctamente en nuestra sistema operativo. Así que pulsamos el botón desbloquear.

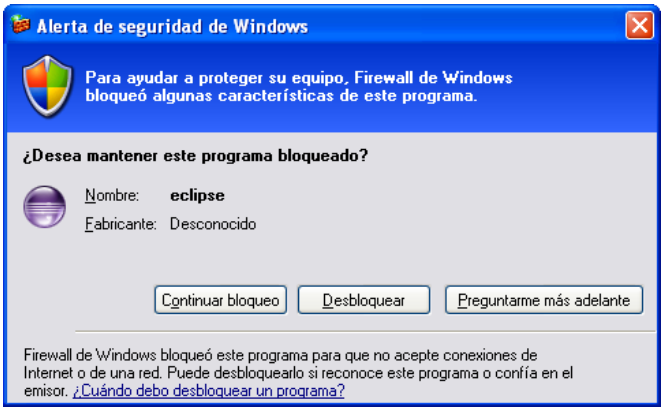



Ilustración 47 - Desbloquear Eclipse en el Firewall

Una vez finalizada la instalación, damos a reiniciar a *Eclipse*.

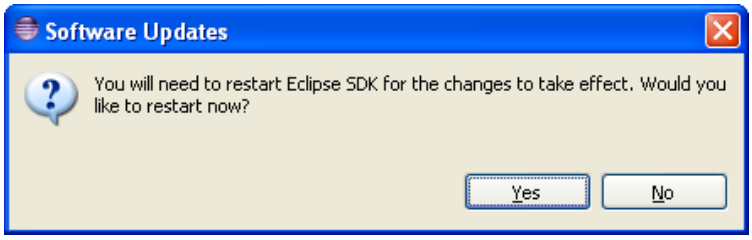



Ilustración 48 - Reiniciar Eclipse

En la siguiente imagen mostramos dos nuevos botones, estos son ***Android SDK Manager*** y el segundo ***Android Virtual Device Manager***. Al tener estos dos botones, nos indica que hemos instalado correctamente el *SDK* de *Android*, y esto significa que ya podemos crear nuestras aplicaciones *Android*.

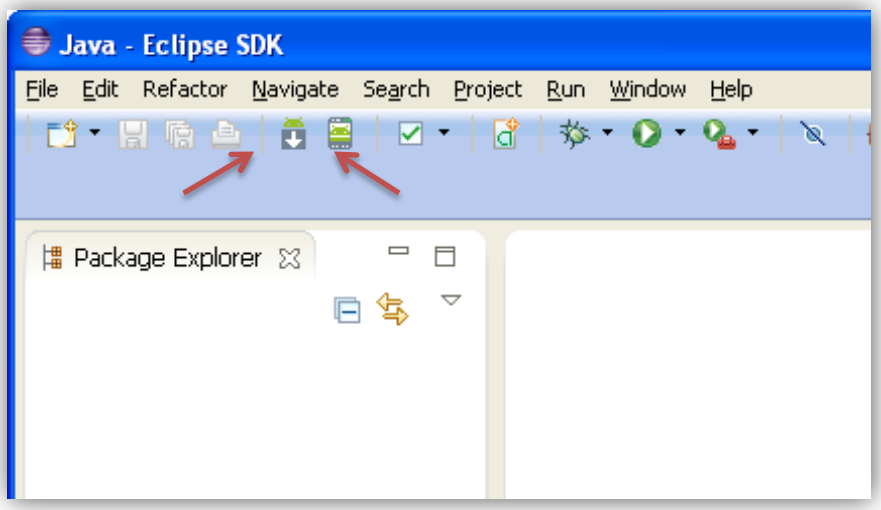



Ilustración 49 - Configuración Android finalizada correctamente

### 4.2.3. Crear Proyecto "Workout Routines"

Para crear el proyecto, vamos a: ***File/New/Other***

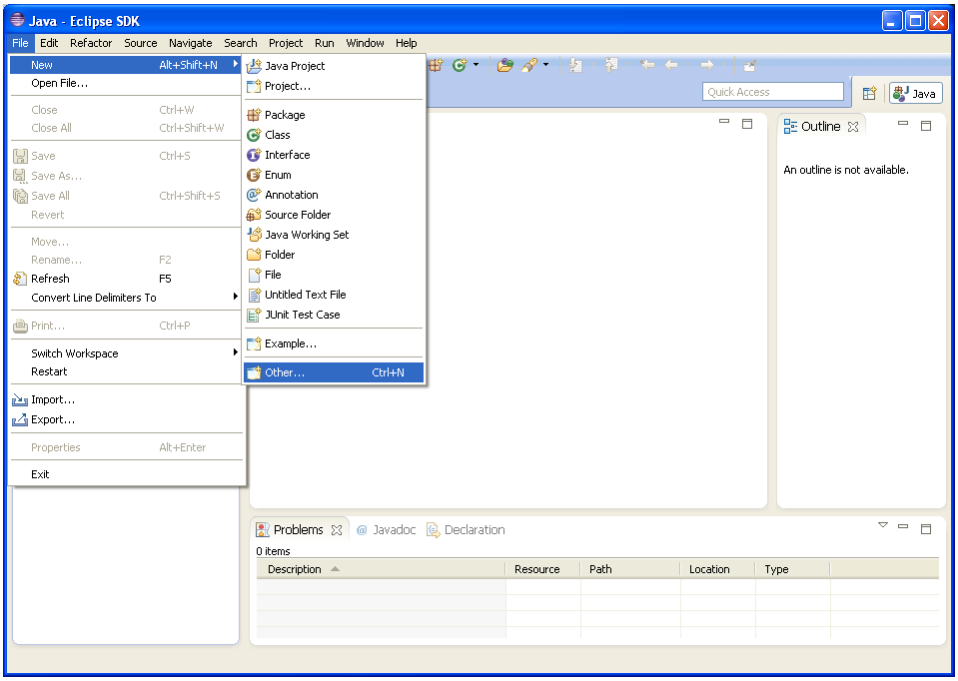



Ilustración 50 - Creación de un proyecto Android

En la pestaña vamos a la carpeta y seleccionamos *Android Application Project*.

De esta manara crearemos nuestro primer proyecto Android.

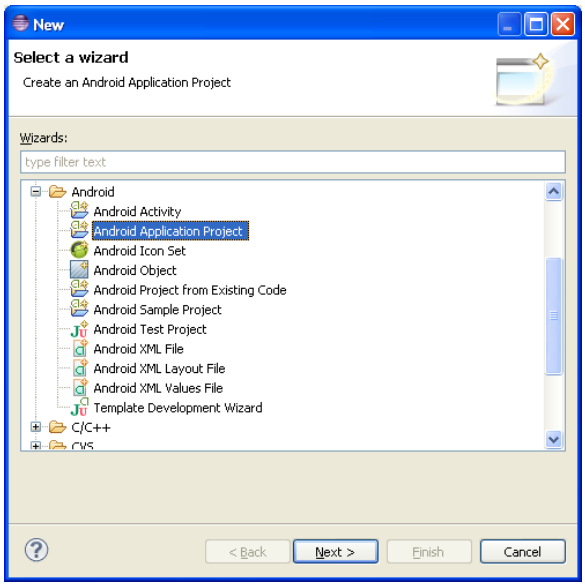



Ilustración 51 - Seleccionamos un proyecto Android

Ponemos un nombre al proyecto y configuramos las *API* necesarias para compilar. Para esto, seleccionamos que nuestra *API* para compilar es la 14 pero que estará en un rango de la *API 8* a la *16*. Esto sirve para ganar compatibilidad con distintos dispositivos móviles.

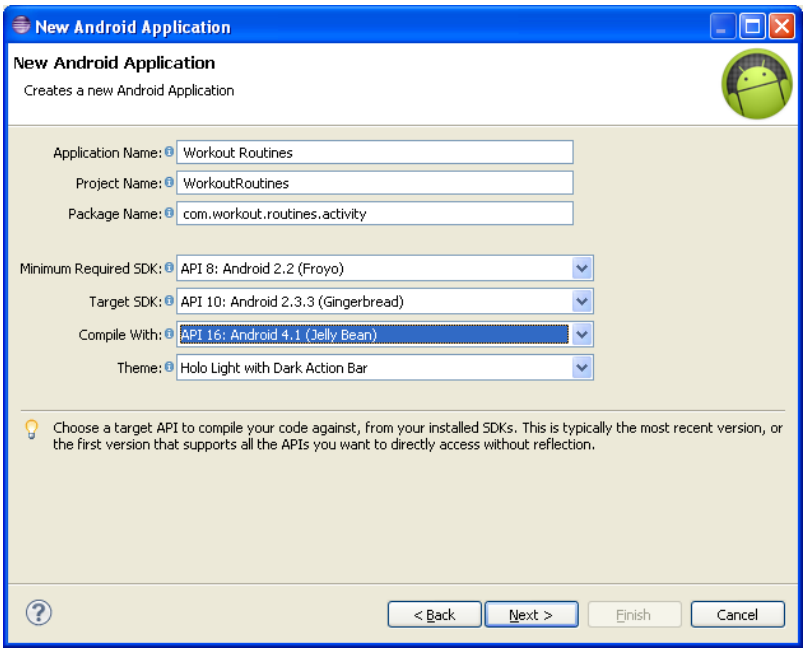



Ilustración 52 - Damos un nombre al proyecto Android

En este apartado seleccionamos el icono a mostrar en el móvil, seleccionamos por defecto el que está, que luego podremos cambiarlo manualmente.

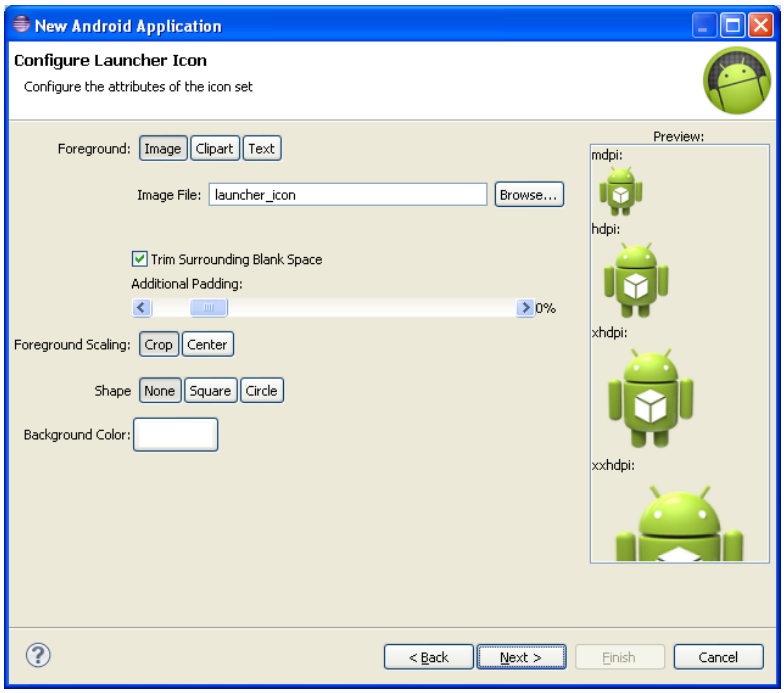



Ilustración 53 - Selección del icono que lanzara la App

En este apartado seleccionamos el nombre de nuestra clase principal y del *XML* que muestra todo el diseño de nuestra *App*. Una vez hecho esto damos a finalizar.

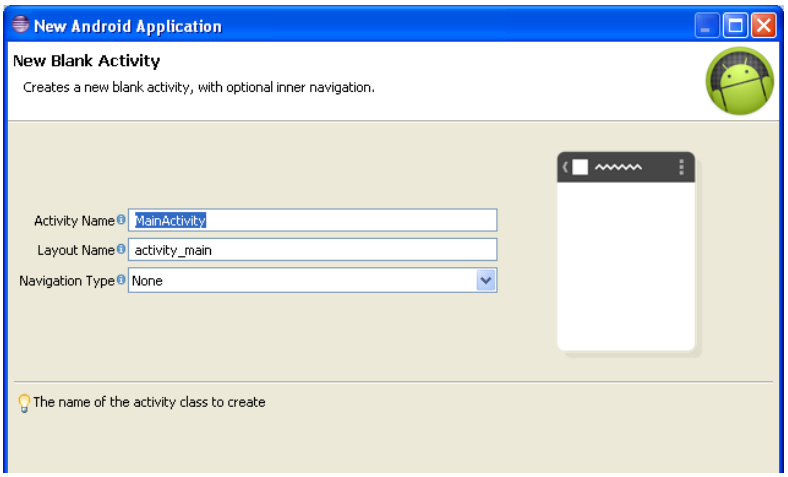



Ilustración 54 - Seleccionamos el nombre de nuestra clase principal

Una vez hecho todo lo anterior, nos queda el siguiente espacio de trabajo, el cual podemos ir mejorando y colocando cosas nuevas actividades o funciones que se adapten a nuestras necesidades y requisitos.

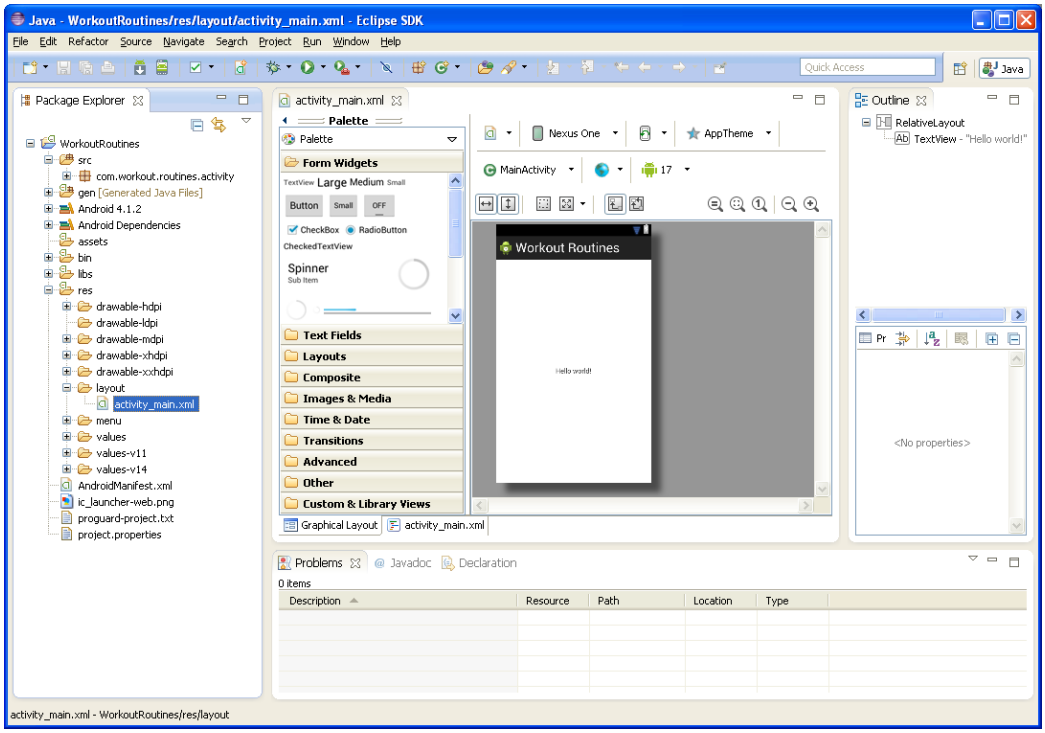



Ilustración 55 - Primer proyecto Android creado en Eclipse

### 4.2.4. Arquitectura de un proyecto Android

***ARCHIVOS:***

1. ***AndroidManifest.xml:*** es el archivo de configuración central de la aplicación, en el detallaremos el rango de compilación de nuestras *App*, las actividades a usar y los permisos que hay que otorgar al móvil para que pueda trabajar sin ningún problema.
2. ***default.properties:*** es un archivo, que no debe ser editado, que utiliza *Eclipse* y el *plugin ADT*.
3. ***proguard.cfg:*** hay que editar el archivo para configurar la optimización del código y la ofuscación cuando se lleve a cabo la publicación de la aplicación.

***DIRECTORIOS:***

1. **src:** dentro se encuentra el paquete creado y se incluye todo el código fuente de la aplicación. Por defecto, se halla el archivo *Activity* que se ha creado en el proceso de creación del proyecto.
2. **gen:** dentro se encuentra el paquete creado y dentro está el archivo ***R.java*** para el manejo de recursos. No se debe editar.
3. **assets:** dentro se encuentran los archivos de recursos, que no sean compilados, necesarios para el proyecto.
4. **res:** se manejan todos los recursos de la aplicación. Incluye animaciones, gráficos, archivos de diseños, datos, archivos raw, etc.

    - **drawable-xxx:** es donde van los iconos gráficos de la aplicación, en diferentes tamaños.
    - **layout:** dentro se encuentra el archivo main.xml, el archivo de recursos gráficos para organizar los controles en la pantalla principal de la aplicación
    - **values:** dentro se encuentra el archivo strings.xml, el archivo de recursos para definir los String que se utilicen

***AndroidManifest.xml:***

1. **Manifest:** se utiliza para configurar las características generales de la aplicación
2. **Application:** se utiliza para definir detalles de la aplicación. En la parte de *Application Nodes* se incluyen todas las actividades que la aplicación puede realizar y, por defecto, aparece la generada en la parte de generación del proyecto. Si se quiere poder debuggear la aplicación, es necesario poner *Debuggable* a *true*.
3. **Permissions:** sirve para otorgar permisos en caso de tener que acceder al teléfono para leer datos
4. **Instrumentation:** sirve para utilizar las clases del *plugin SDK* de testeo.

### 4.2.5. Creación del Dispositivo Virtual

Esta configuración es muy importante, ya que nos permite emular cualquier móvil y de distinta versión. Sin esto, no podremos hacer las pertinentes pruebas de ejecución, si no se tiene un móvil Android configurado para depurar.

Vamos al Botón a la barra de herramientas de Android y pulsamos el botón "***Android Virtual Device Manager***"

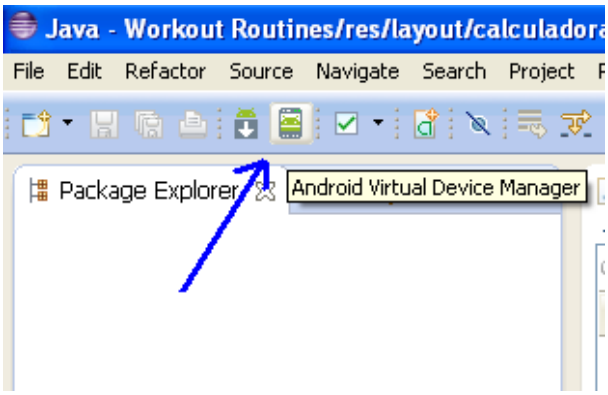



Ilustración 56 - Seleccionar botón Android Virtual Device Manager

Nos abrirá una ventana en blanco la primera vez, pero en esta imagen tengo unas versiones emuladas para las pruebas pertinentes. Así que para crear un nuevo dispositivo virtual, le damos al botón *New*

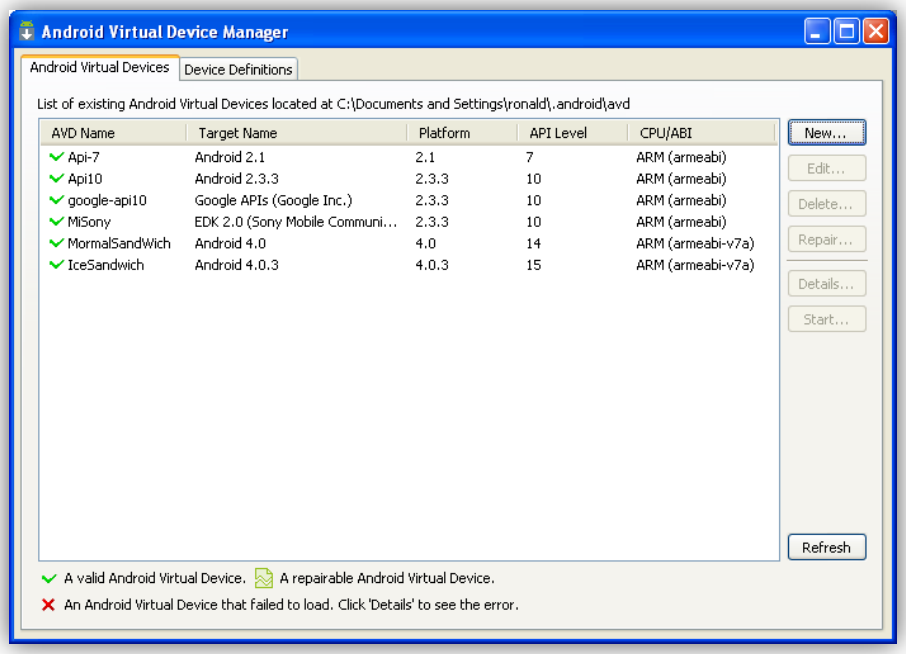



Ilustración 57 - Consola de Android Virtual Device Manager

Lo segundo es dar un Nombre al dispositivo, con su *Target* Correspondiente y el tamaño de la pantalla a emular. Por último damos al botón Ok.

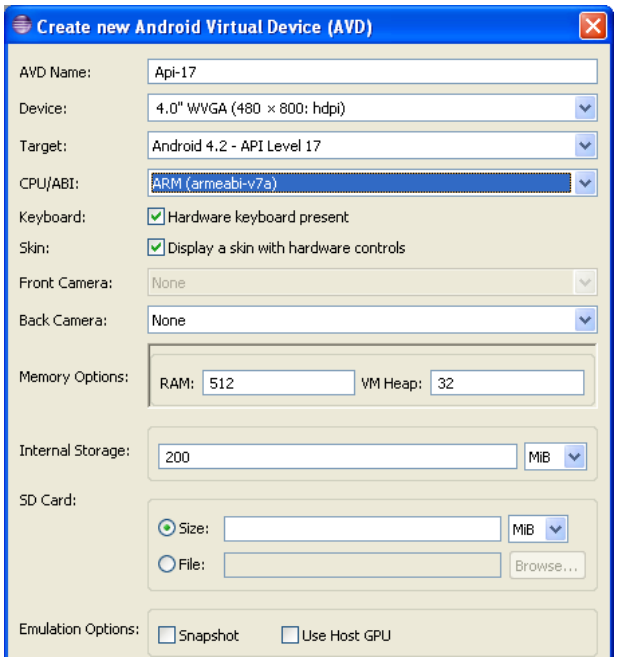



Ilustración 58 - Creación de un nuevo dispositivo virtual

Una vez finalizado esto, iremos al Proyecto *Android*, haciendo clic derecho con el ratón y en *Run as*, buscamos la opción: *1. Android Application*.

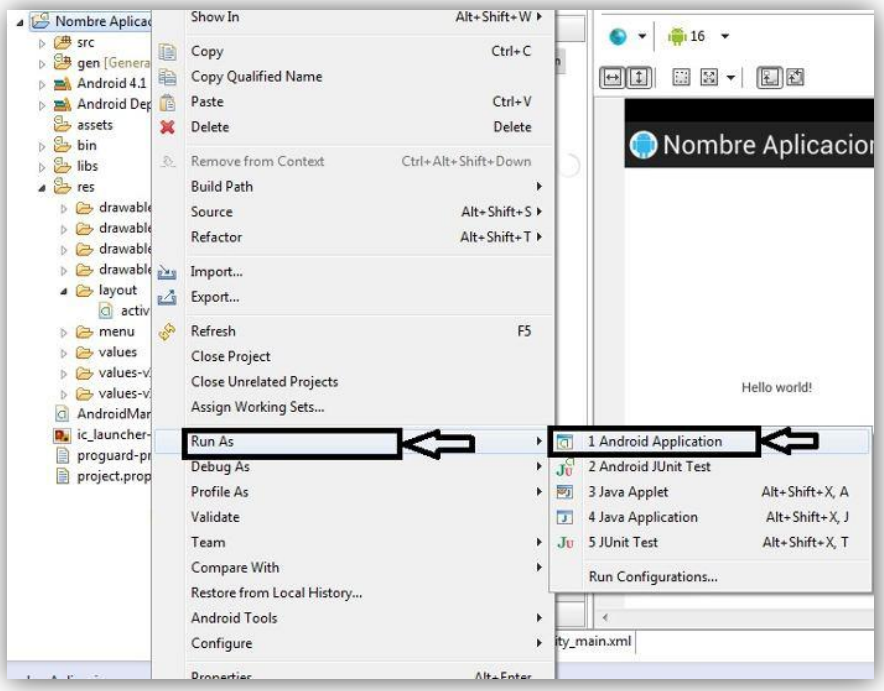



Ilustración 59 - Ejecutar el proyecto con el dispositivo virtual creado

Esto nos generará un Dispositivo Virtual, en el cual podremos emular cualquier Móvil, con su correspondiente versión de Android, sin necesidad de tener uno físico.

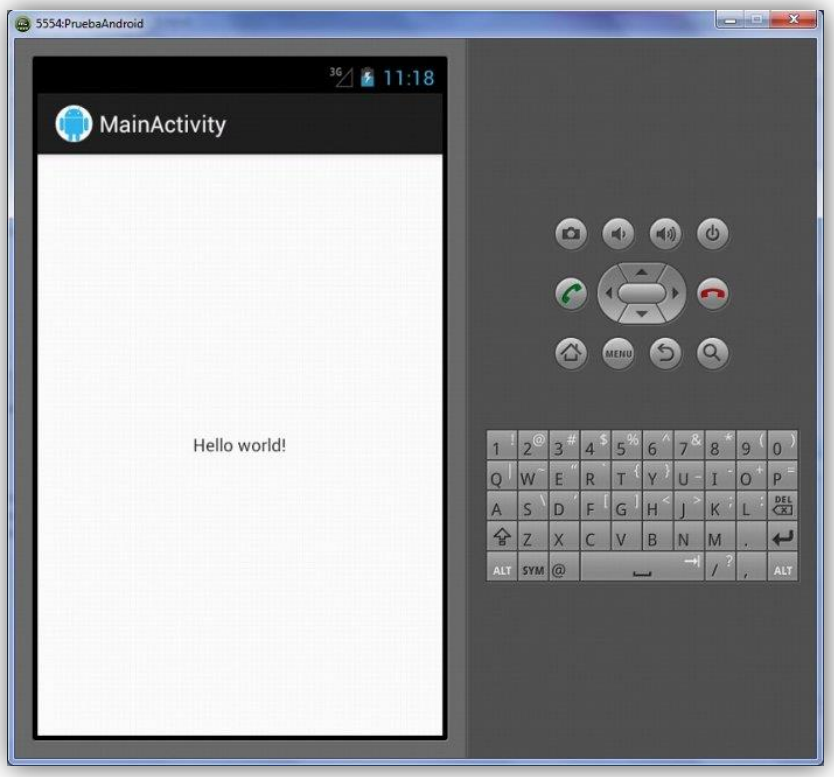



**Ilustración 60 - Dispositivo virtual emulado con nuestro proyecto**

### 4.2.6. Configuración Móvil Android Para Depuración con Eclipse

Antes que nada, hay que instalar el *Driver* del dispositivo móvil, el cual nos permite interactuar con el ordenador. Cada controlador varía dependiendo de la marca del móvil.

Lo primero que debemos hacer es configurar el móvil en Modo Desarrollador; para esto debemos ir a *Ajustes* del dispositivo móvil.


Ilustración 61 - Seleccionar ajustes

En ajustes, vamos a la opción de *Seguridad* y allí vamos a *Orígenes Desconocidos* y activamos el CheckBox. Esta opción nos permite instalar aplicaciones que no pertenezcan a *Android Market*.


Ilustración 62 - Aceptar instalación de orígenes desconocidos

Por último, debemos activar la Depuración USB, para esto vamos a *Opciones del Desarrollador*, y una vez allí dentro activamos la opción de *Depuración USB*.

Esta opción es la más importante, ya que Eclipse depende de ella para compilar y ejecutar todos los programas *Android*.


Ilustración 63 - Seleccionar depuración USB

Cabe decir, que estas opciones suelen cambiar del dispositivo móvil, ya sea por la marca o versión del software, pero la idea sigue siendo la misma.

El segundo paso es ajustar Eclipse para que permitan detectar dispositivos móviles, para esto, tenemos que descargarnos el complemento Extra que trae el SDK Manager, éste se llama "*Google USB Driver*". El cual nos permite reconocer el dispositivo móvil en Eclipse.

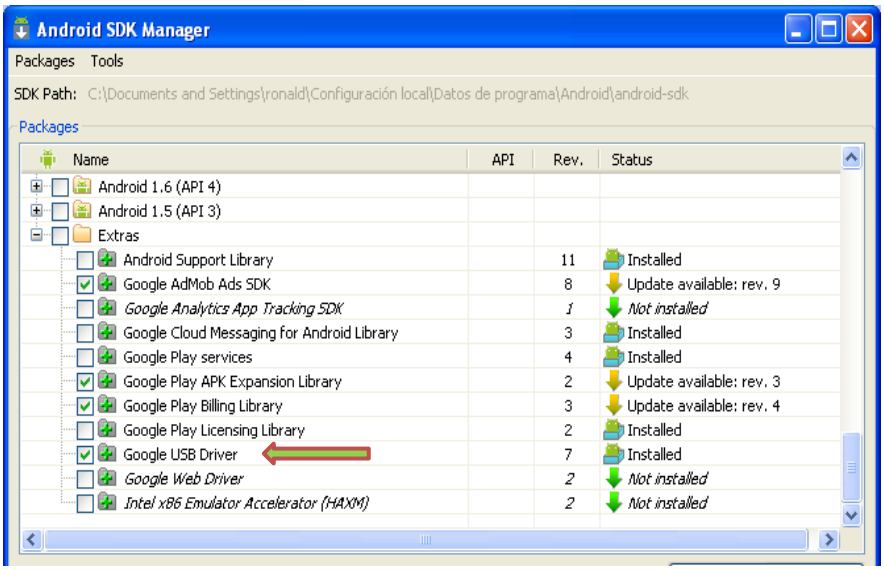



Ilustración 64 - Instalar Google USB Driver

Por último queda configurar Eclipse, por defecto lo ponemos en Automático. De esta manera *Eclipse* buscará y reconocerá si está conectado el Móvil al Ordenador, para que dé a lugar a compilar y ejecutar directamente en el móvil sin necesidad de usar un emulador.

Si por cualquier motivo no está enchufado el móvil, Eclipse lanzara por defecto un emulador Virtual.

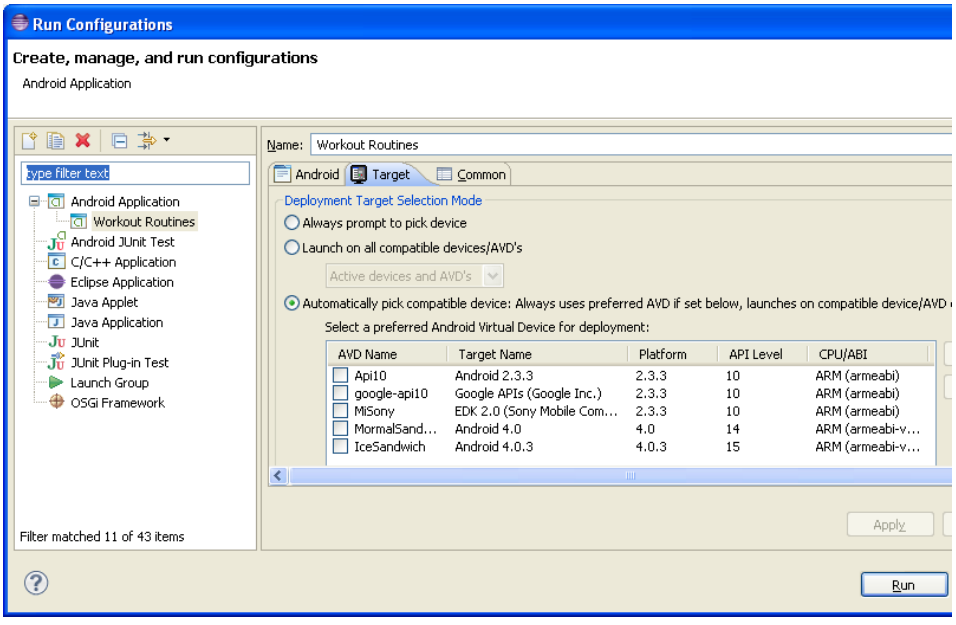



Ilustración 65 - Configuración del Target de Android

## 4.3. Bocetos Manuales

Para hacer los bocetos manualmente y que me permitan diseñar de manera correcta las interfaces, he usado una platilla que tiene las dimensiones de mi dispositivo móvil.

De esta manera aprovecho todos los espacios sin dejarme nada y con la facilidad de poder crear, rediseñar o borrar esquemas que luego llevare a su pertinente producción.

La platilla general tiene 9 mini platillas a escala real de mi dispositivo móvil. La Ilustración 66 muestra un ejemplo de una platilla en Blanco.

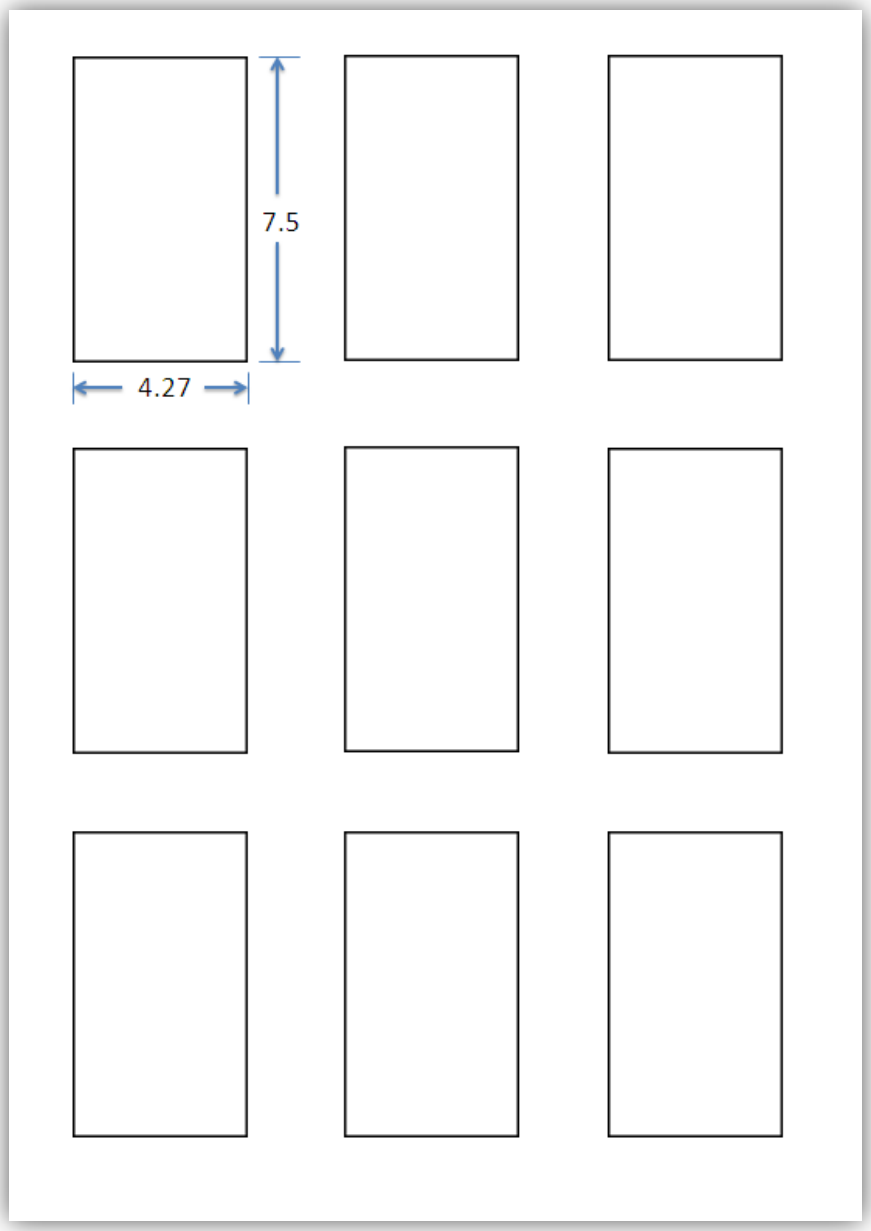



Ilustración 66 - Plantillas para los bocetos

### 4.3.1. Bocetos Iniciales.

A continuación muestro la primera fase de diseño que he realizado para la propuesta de este proyecto. Se ha diseñado una primera aproximación para mostrar la funcionalidad principal de la aplicación. Aquí especifico las primeras funcionalidades de la aplicación, definiendo una serie de botones, imágenes y actividades. Esta primera imagen corresponde con el boceto inicial, el cual me llevaría a generar más ideas sobre esta base (Escaneos Originales).

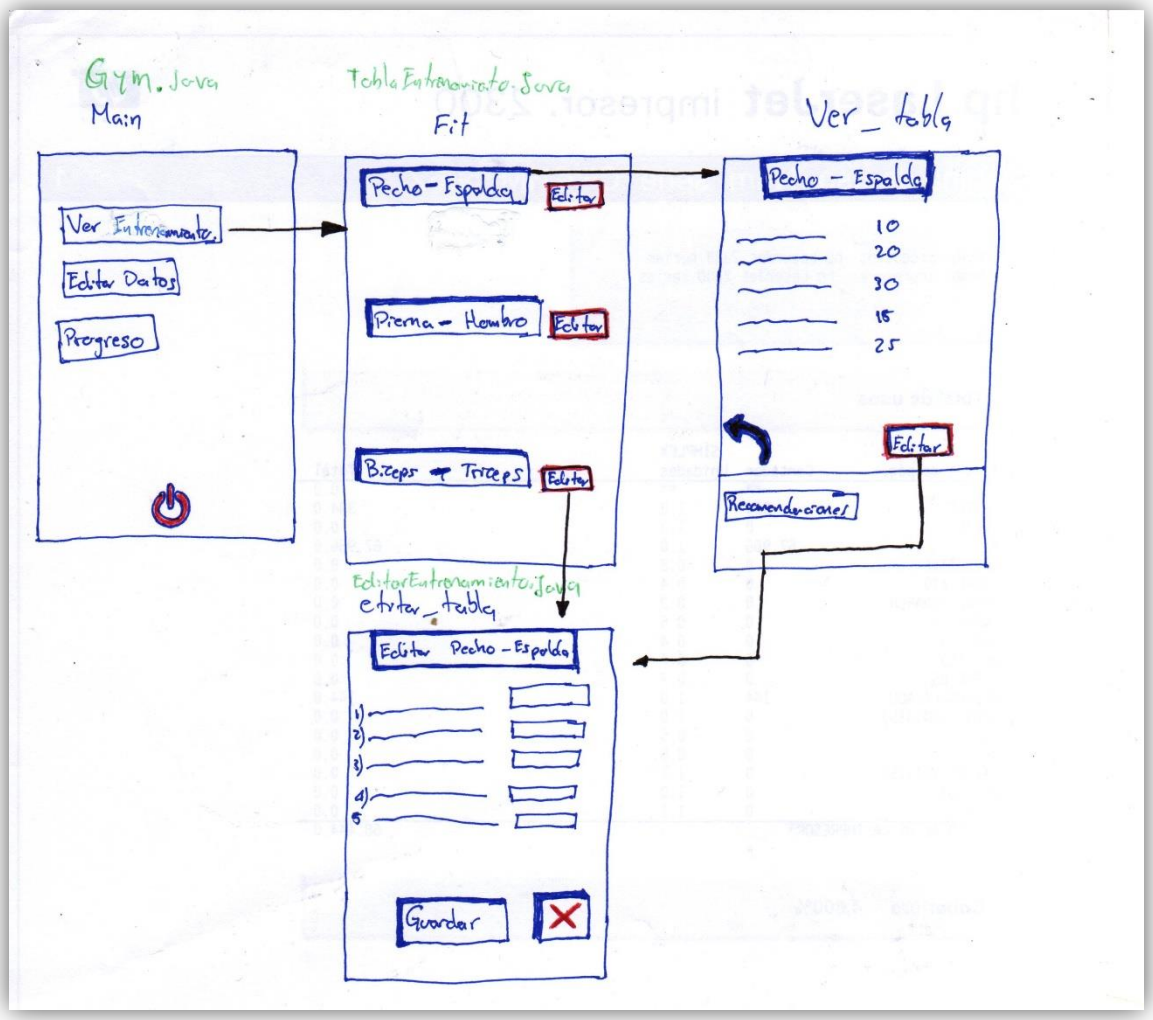



Ilustración 67 - Boceto inicial para el proyecto

Esta corresponde con la primera idea de cómo sería en sus inicios la BD de la app. La cual sería mejorada en cada fase.

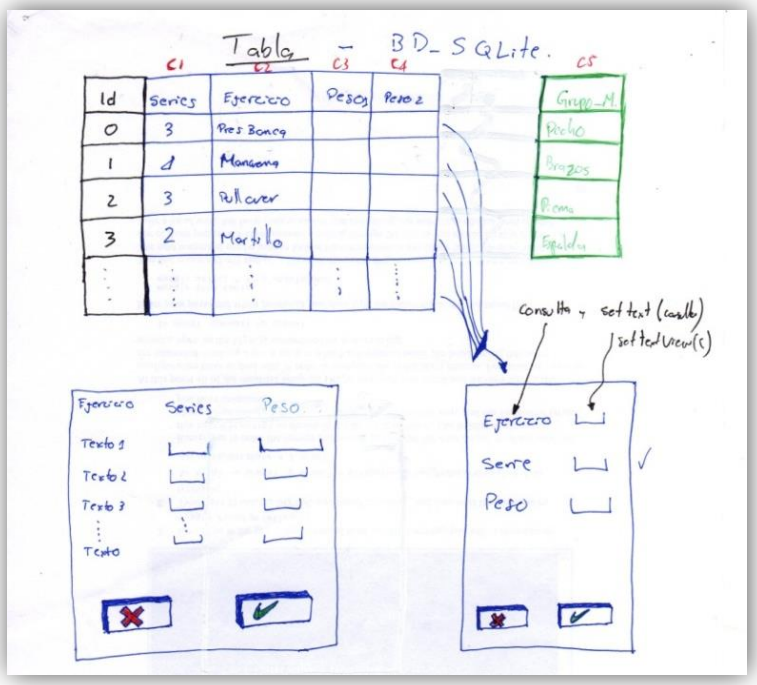



Ilustración 68 - Idea inicial de la BD en el dispositivo móvil

Esta sería una mejora del diseño, pero no la definitiva.

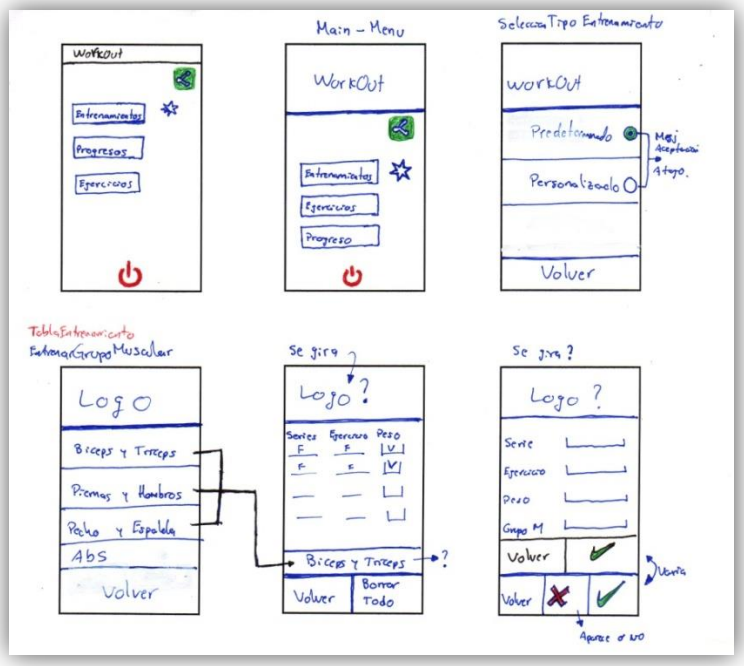



Ilustración 69 - Modelo intermedio del proyecto

Por terminar con el diseño y programación de la *App*, se optó por este diseño, ya que es más práctico y más intuitivo. Este boceto no refleja el diseño final, pero es la base en la cual me apoyado para realizar este proyecto.

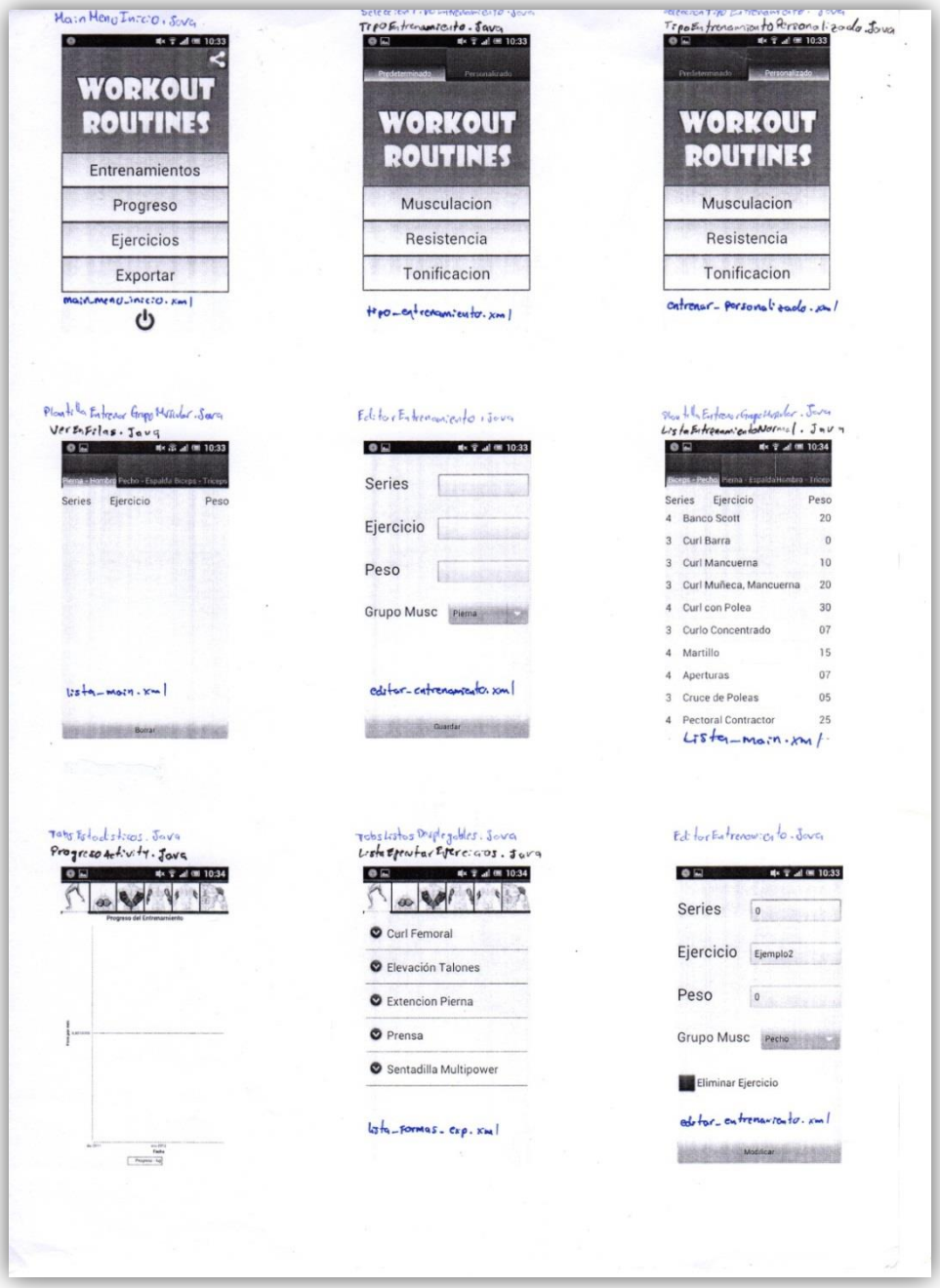



**Ilustración 70 - Plantilla base para desarrollar la App**

## 4.4. Diseño Interfaz de Usuario

En este apartado, voy a explicar el desarrollo de las interfaces de este proyecto, como son los botones, las imágenes, las ventanas, etc.

### *4.4.1. Diseño Icono de la Aplicación.*

Para hacer visible nuestra aplicación, ya sea en los dispositivos móviles o en *Google Play*, hay que diseñar un logo con las siguientes características que recomienda *Google*:

- Archivo *PNG* de alta resolución (con alpha) de 32 bits.
- (tamaño máximo: 1.024 KB) 512 x 512
- Se recomienda tener un sombreado en los bordes, para hacer más visible el logo en dispositivos que utilicen fondos Blancos o muy Negros.

Como se puede apreciar en la Imagen, esta cumple con los requisitos y está acorde con lo que quiere reflejar nuestra aplicación. Es de color verde con degradado amarillo para hacer más visible e inconfundible.

Ilustración 71 - Logo de la App, Launcher

### 4.4.2. Diseño Logo de la aplicación.

El logo de esta aplicación, realizado con *Photoshop*, ayuda a recordar a los usuarios cómo se llama la aplicación que están usando en sus días de entrenamiento. Es una forma de inducir a la publicidad.


Ilustración 72 - Logotipo con el nombre de la App

### 4.4.3. Diseño Botón Compartir.

Para este diseño se opta por un botón con fondo verde, con el símbolo que mejor lo define. Se diseñaron dos botones, ya que uno se utiliza para cuando no se pulsa y el otro para cuando se pulsa, básicamente para reflejar dinamismo a la hora de su utilización.

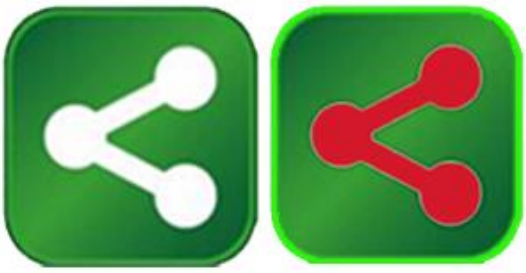


Ilustración 73 - Diseño botón compartir

A la hora de pulsar este botón, llama a una función que es la encargada de gestionar todas las herramientas que nos permiten compartir hacia las redes sociales y de comunicación, estas utilidades varían del número de aplicaciones instaladas en cada dispositivo. Este método cambia el Idioma del mensaje dependiendo del lenguaje establecido en el teléfono, aunque la estructura básica es la misma.

Android por defecto nos crea su propia interfaz gráfica, solo hay que poner el mensaje que se desea comunicar. La siguiente imagen muestra cómo se implementa la herramienta de compartir.

```
// Creamos el msj a compartir en la Red Social
Button bshare = (Button) findViewById(R.id.compartir);
bshare.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        //Accion Enviar
        final Intent intent = new Intent(Intent.ACTION_SEND);

         /*****Seleccion del Idioma del telefono ******/
        String idioma = getResources().getConfiguration().locale.getDisplayName().trim();
        if(idioma.equals("español (España)")){
            intent.setType("text/plain"); // Enviar a todas las App Disponibles
            intent.putExtra(Intent.EXTRA_SUBJECT, "Descarga Workout Routines");
            intent.putExtra(Intent.EXTRA_TEXT, "Visita mi pagina web:  " +
                    "http://www.workoutroutines.hol.es/");
        }else{
            intent.setType("text/plain"); // Enviar a todos
            intent.putExtra(Intent.EXTRA_SUBJECT, "Download Workout Routines");
            intent.putExtra(Intent.EXTRA_TEXT, "Visit my website:  " +
                    "http://www.workoutroutines.hol.es/");
        }

        //Titulo de la presentación, => R.string.app_name
        startActivity(Intent.createChooser(intent, getString(R.string.app_name)));
    }
});
```

Ilustración 74 - Código botón compartir

La siguiente imagen, muestra una línea de código que es importante, ya que dependiendo del orden de las palabras, podremos compartir con todas las aplicaciones o solo para unas cuantas. Por defecto se pone **"text/plain",** ya que así nos aseguramos que abarque todo, si el orden fuese al revés, solo compartirá con *Bluetooth* o *Gmail*.

```
intent.setType("text/plain"); // Enviar a todas las App Disponibles
```

La siguiente imagen muestra las opciones para compartir, una en español y la otra en Ingles.

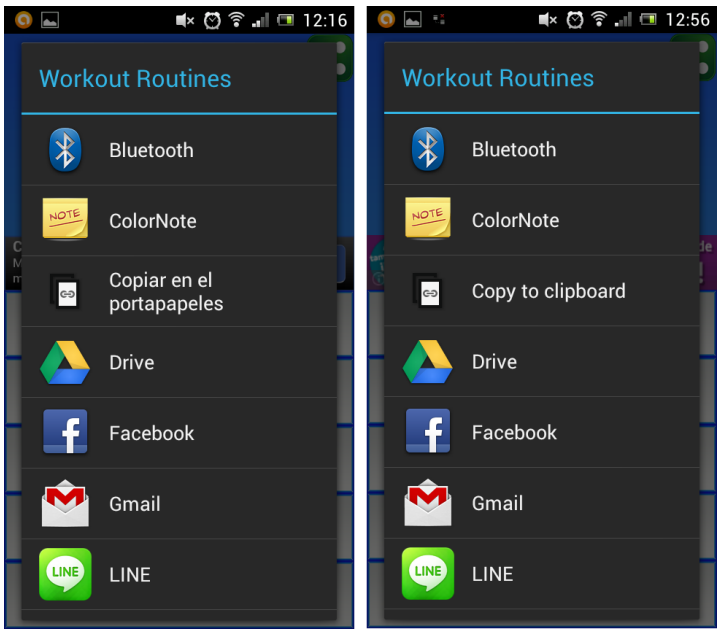



Ilustración 75 - Opciones de compartir, en inglés o español

### 4.4.4. Diseño de los Botones del Menú Principal

En un primer diseño, se planteó la idea de hacer varios botones, con texto incluido en la imagen del botón, pero esto al final no resultó ser muy cómodo, ya que si se quiere modificar para que sea Multilenguaje, habrá que diseñar más botones y con la carga de tener que cambiarlos y añadirlos de forma casi manual, con el agravante de que el tamaño de la *App* iba aumentando por cada imagen agregada.


Ilustración 76 - Primer modelo de los botones del menú principal

Así que se diseñó un solo botón, el cual tomaría diferentes valores, parecido al diseño anterior, solo que sin texto, ya que se espera que lo tome de manera automática a partir de una lista de nombres, que pueden estar en diferentes Idiomas, en nuestro caso solo será inglés y español.

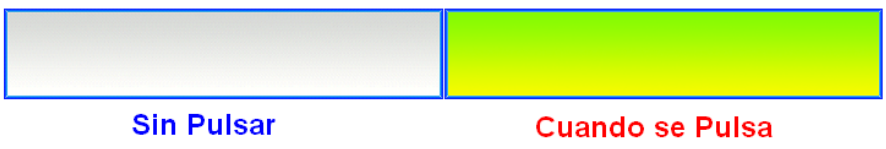



Ilustración 77 - Modelo final de los botones de la App

Para hacer que se vea que hay una interacción mientras se pulsa una tecla, lo que se hizo fue que mientras el usuario pulse una tecla, ésta se cambie por otra imagen de la misma forma pero con distinto fondo.

Para hacer esto hay que crear un fichero XML, al cual llamamos **botones_animados.xml**, en letras minúsculas, porque si no es así, dará error.

El siguiente código muestra cómo asignar las imágenes, que se encuentran en las carpetas *drawable*.

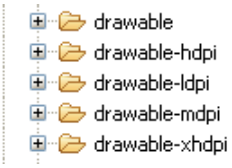



Ilustración 78 - Directorios para almacenar las imágenes de la App

Cabe recordar, que Android busca de manera autónoma, solo basta con dejar las imágenes en alguna de estas carpetas, solo hay que hacer referencia al nombre de las dos imágenes (no hace falta poner extensión) y dejar indicado qué imagen cambia de estado, dejando este valor a Verdadero.

botones_animados.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <selector xmlns:android="http://schemas.android.com/apk/res/android">
3     <item android:drawable="@drawable/plantilla2"
4           android:state_pressed="true" />
5     <item android:drawable="@drawable/plantilla1" />
6 </selector>
```

Ilustración 79 - Diseño XML de los botones animados

### *4.2.4.1. Asignar imagen a un botón.*

Creamos un botón ya sea por XML o por el Palette de Diseño, con las propiedades y nombres que se necesiten para cada aplicación. Lo único que hay que hacer es poner la siguiente línea de código `android:background="@anim/botones_animados"` dentro de la definición de un botón Android tal como muestra la siguiente imagen.

```
<Button
    android:id="@+id/bmi"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:background="@anim/botones_animados"
    android:text="@string/imc"
    android:textSize="30sp" />
```

Ilustración 80 - Asignación de imágenes a un botón especifico

### 4.4.5. Programación y Diseño General de la Actividad Principal.

Para hacer el diseño se empleó Layouts Relativos, ya que así nos aseguramos que haya compatibilidad con los diferentes tamaños de pantallas que existen en el mercado, ya que tienen que ser compatibles con *Tablets* o Móviles pequeños.

Dentro del *Relative*, se empleó un *LinearLayout* en orientación vertical que contiene botones y otros *Layouts* para agrupar más elementos, tales como:

1. Logo.
2. Publicidad.
3. Entrenamiento.
4. Progreso.
5. Ejercicios.
6. Exportar – Importar y otro.
7. LinearLayout, que a su vez contiene dos botones, los cuales son:
    a. Encuesta e IMC.
8. Y por fuera, el Botón de Compartir, ya que interesa tenerlo en una esquina, donde sea más fácil verlo y reconocerlo.

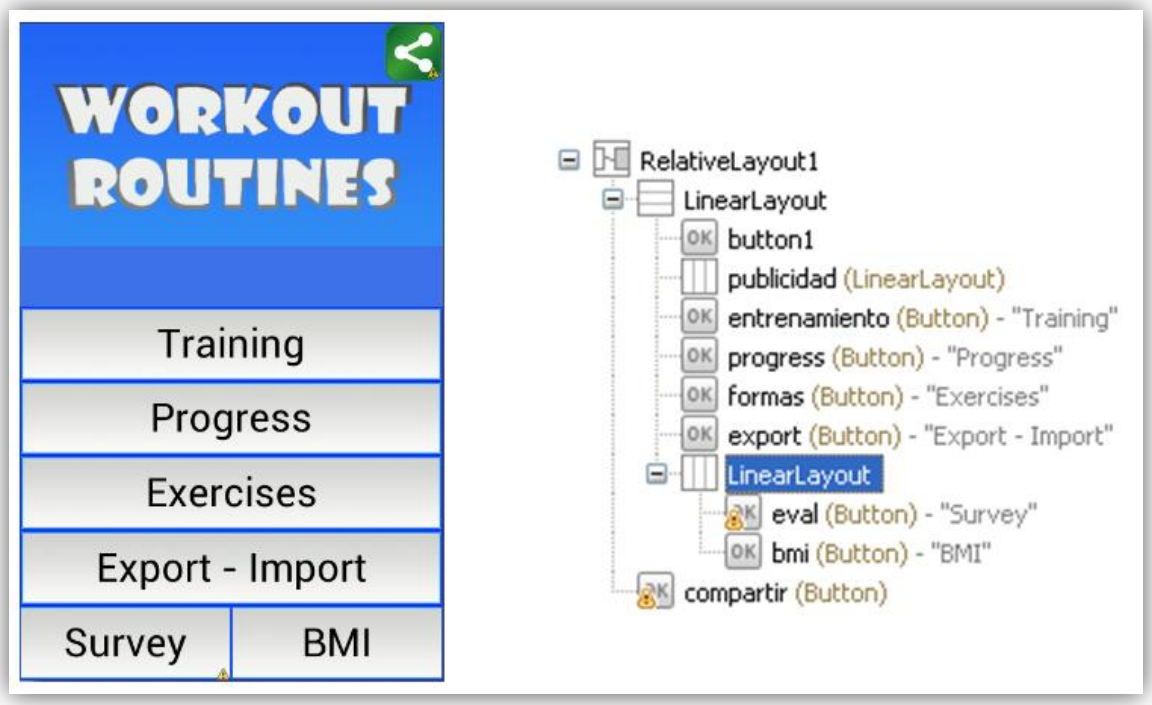



Ilustración 81 - Diseño final del menú Inicio

Gracias a la ayuda del Palette de Diseño, ésto se hace mucho más fácil a la hora de agrupar, espaciar y redimensionar.

#### 4.4.5.1. Peso y Dimensiones de la Actividad Principal.

Uno de los problemas a la hora de diseñar para la Interfaz de Android, es saber manejar los tamaños de las Imágenes, Botones, Textos, Áreas, etc., ya que por defecto estos complementos toman tamaños predefinidos y hay que cambiarlos para que se parezcan al diseño elegido y que se adapten a los diferentes modelos y resoluciones de móviles y tabletas.

Usando la propiedad ***android:layout_weight***, puedo especificar una relación de tamaño entre diferentes *Views* de un *layout*, orientando más que nada el espacio que queremos que ocupen todos los componentes respecto a otros, aunque por sí sola no hace mucho y hay que combinarla con otras dos propiedades, que son: ***android:layout_ width*** y ***android:layout_ height*.** Éstas definen el ancho y el alto de un contenedor y/o *widget* respectivamente.

A continuación muestro un ejemplo sencillo, el cual me ayudará a especificar mejor en mi diseño final.

Ejemplo original: http://androideity.com/2012/06/01/ui-fluidas-y-la-propiedad-weight-en-android/

Voy a agregar tres botones dentro de un *LinearLayout*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:gravity="center_vertical" >

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/boton_uno"/>
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/boton_dos"/>
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="2"
        android:text="@string/boton_tres"/>

</LinearLayout>
```

Ilustración 82 - Asignación de pesos de resolución a cada Widget

Este código nos dará el siguiente resultado:

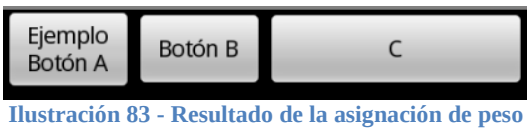



Ilustración 83 - Resultado de la asignación de peso

Lo anterior, corresponde claramente a una distribución del 25%, 25% y 50% respectivamente para los elementos agregados al Layout.

Además, se observa que el botón "C" que tiene más peso, abarca más espacio a lo ancho, que los demás (ya que su valor es "dos").

Para nuestro diseño empleamos estos pesos, mostrados en la gráfica siguiente al lado de su correspondiente botón o layout y todos con un ancho igual a Cero.

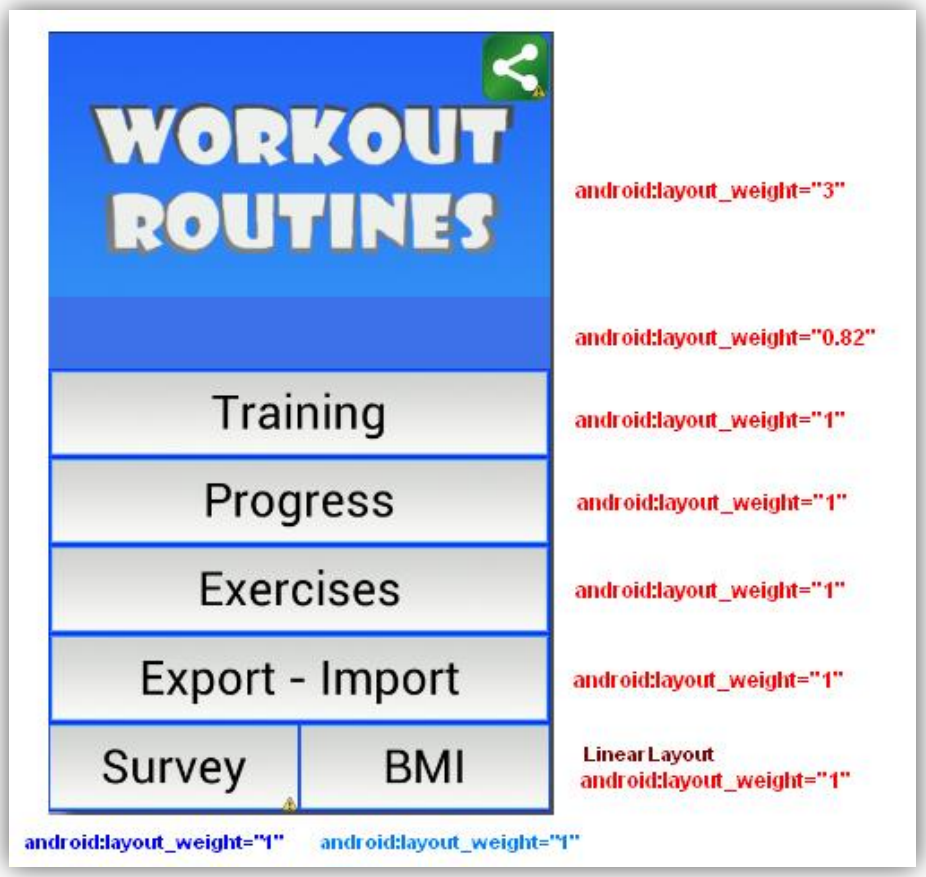



Ilustración 84 - Asignación de pesos al menú Principal de la App

La gráfica anterior muestra la distribución de espacio y redimensionamiento que toman mis objetos a la hora de programar y configurar. Además, se ve que en la parte inferior hay dos botones que no corresponden con el diseño que se ejecuta en los dispositivos. Esto se debe a que a la hora de programar, por defecto se oculta el botón *BMI*, dando lugar a que el botón *ENCUESTA* tome todo el ancho posible, una vez hecha la encuesta, este botón desaparece y se hace visible el botón *BMI* para siempre. Esto no interfiere con nuestro diseño en general, ya que está encapsulado dentro de un layout que tiene un comportamiento diferente.

#### 4.4.5.2. Programación Actividad Principal

En este capítulo voy a mostrar y explicar todas aquellas Líneas de Código, Funciones o Clases que me parecen importantes en el desarrollo de esta aplicación.

Para el desarrollo de la actividad principal, creamos la clase **MainMenuInicio.Java**, donde esta extiende de la Clase *Activity*, el cual sirve para trabajar en conjunto con el Layout asociado a este (main_menu_inicio.xml). Esta combinación permite integrar lo que es el diseño de la interfaz en XML con la programación en lenguaje *Java*.

Al crear una clase que extienda *Activity* hay que implementar al menos la función *onCreate()*, que es la que se ejecuta cuando se crea la actividad.

La siguiente imagen muestra cómo se nos crea por defecto la función *onCreate(),* junto con el nombre dado y el paquete asociado.

```
1  package com.workout.routines.activitys;
2
3  import com.google.ads.AdRequest;
19
20 public class MainMenuInicio extends Activity {
21
22     SQLiteBD bd = new SQLiteBD(this);
23
24     @Override
25     public void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         // Quitar el titulo de la aplicacion
28         requestWindowFeature(Window.FEATURE_NO_TITLE);
29         // Cargar el Layout asociado
30         setContentView(R.layout.main_menu_inicio);
31         // Poner siempre Vertical
32         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
```

Ilustración 85 - Creación de la clase Main con Java

En las líneas 29 y 31 de la imagen anterior, explico a través de un comentario, qué hace la instrucción siguiente.

#### 4.4.5.3. Botón Entrenamientos (Cambiar de Actividad)


Ilustración 86 - Botón Entrenamientos y su funcionalidad

La función de cambiar de actividad es común para los botones **Progreso, Entrenamientos *(Musculación, Resistencia y Tonificación),* Ejercicios, Exportar – Importar, Encuestas, IMC***, y*a que lo que hacemos es prácticamente lo mismo. Crear un botón con XML, darle un identificador y una imagen de fondo. Luego instanciar ese botón y cargarlo a un objeto de tipo *Button*, al cual le hacemos un *Clic Listener* que se encargue de avisar si se ha pulsado o no, y agregar su correspondiente *Intent*, el cual nos sirve para navegar entre Actividades.

La siguiente línea de código, permite manipular los eventos asociados al botón Entrenamientos.

```
// Boton que nos lleva a otra actividad, Crearemos los entrenamientos
Button bentrenar = (Button) findViewById(R.id.entrenamiento);
bentrenar.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {

        Intent i = new Intent(getApplicationContext(), SeleccionTipoEntrenamiento.class);
        i.setAction(Intent.ACTION_VIEW);
        i.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(i);
    }
});
```

Ilustración 87 - Código para ejecutar el botón Entrenamientos

Con el código anterior, lo que hacemos es cambiar de una actividad a otra. Para ello lo que hacemos es crear un botón, a éste agregarle el diseño personalizado y darle un identificador único, el cual nos sirve para instanciarlo desde las clases que queramos. Para esto lo que hacemos es:

- Reconocer el botón al cual aplicar el código. Para esto lo que hacemos es crear un objeto de tipo *Button*, al cual pasamos el identificador (Entrenamientos) del botón creado anterior mente por el XML.

```
Button bentrenar = (Button) findViewById(R.id.entrenamiento);
```

- Después, lo que hacemos es crear un *Clic Listener* de este botón, el cual reconozca que hemos pulsado el botón. Una vez reconocida la acción de pulsar, lo siguiente es establecer las funcionalidades correspondientes a esta acción, que me permitan navegar entre actividades, para ello usamos la Clase *Intent* de la librería *Android*.

- Para navegar a través de las actividades, lo que tenemos que hacer es crear un objeto llamado ***Intent***, con lo parámetros de la clase en la que estamos y la clase a la que queremos invocar.

```
Intent i = new Intent(getApplicationContext(), SeleccionTipoEntrenamiento.class);
```

- para que esto funcione, lo que hacemos es iniciar la ejecución con ***startActivity(i);*** de esta forma cambiamos de Actividad.

Como el resto de código es similar, ya que el origen de donde se parte es el mismo pero el destino no, lo que voy hacer es poner sus correspondientes destinos, con sus correspondientes clases y gráficas de dónde parten y adónde se dirigen.

#### 4.4.5.4. Botón Progreso

Este botón nos lleva a una Actividad donde nos mostrará el avance que logramos, a partir de gráficas separadas por pestañas de grupos musculares.

```
// Boton para ir a la grafica de progresos
Button bprogreso = (Button) findViewById(R.id.progress);
bprogreso.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        Intent i = new Intent(getApplicationContext(), TabsEstadisticos.class );
        startActivity(i);
    }
});
```

Ilustración 88 - Código para ejecutar el botón Progreso


Ilustración 89 - Demostración de la funcionalidad del botón Progreso

#### 4.4.5.5. Botón Ejercicios

Este botón nos lleva a una Actividad donde se nos muestran los diferentes ejercicios que existen para llevar el día a día de un entrenamiento muscular. Éstos están separados por pestañas diferenciando los grupos musculares a trabajar. Además, los nombres, con sus respectivas imágenes, están ordenados alfabéticamente para que la búsqueda sea más fácil.

```
// Boton para ver como hacer los ejercicios
Button bejercicios = (Button) findViewById(R.id.formas);
bejercicios.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        Intent i = new Intent(getApplicationContext(), TabsListasDesplegables.class );
        startActivity(i);
    }
});
```

Ilustración 90 - Código para ejecutar el botón ejercicios

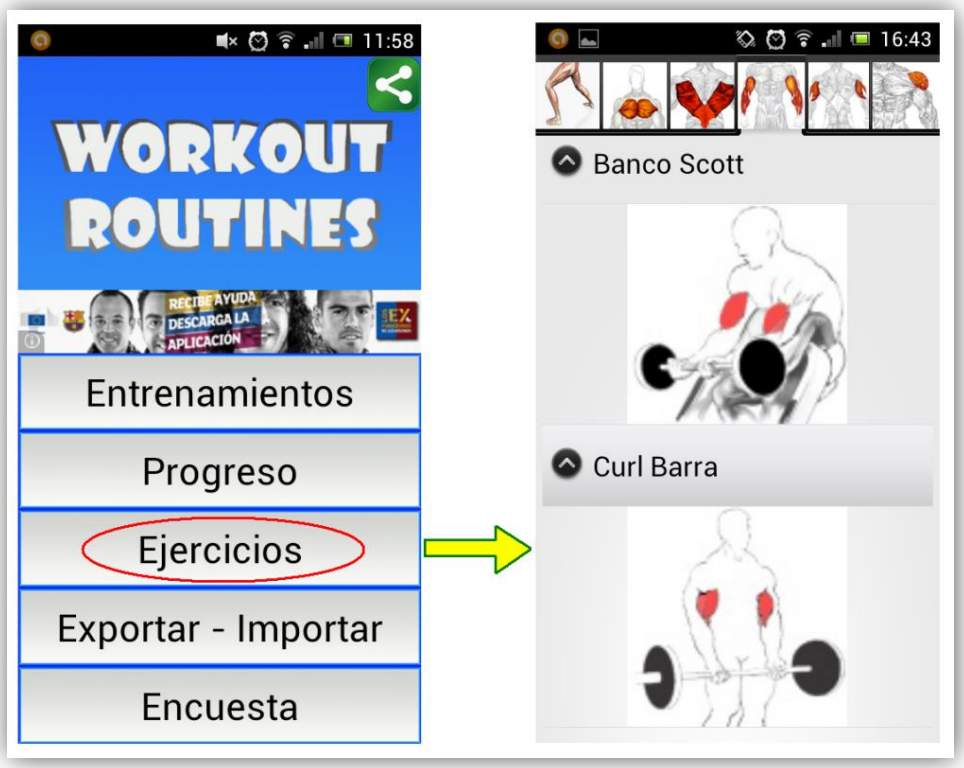



Ilustración 91 - Demostración de la funcionalidad del botón Ejercicios

#### 4.4.5.6. Botón Exportar – Importar

Esta herramienta de Exportar e Importar es la que destaca mi aplicación de otras aplicaciones en *Google Play*, ya que de esta manera los usuarios de esta aplicación podrán traspasarse los entrenamientos creados previamente. Esta herramienta es muy útil, ya que en el mundo del gimnasio y entrenamientos musculares es muy habitual intercambiarse las tablas de entrenamiento para variar un poco en la rutina diaria.

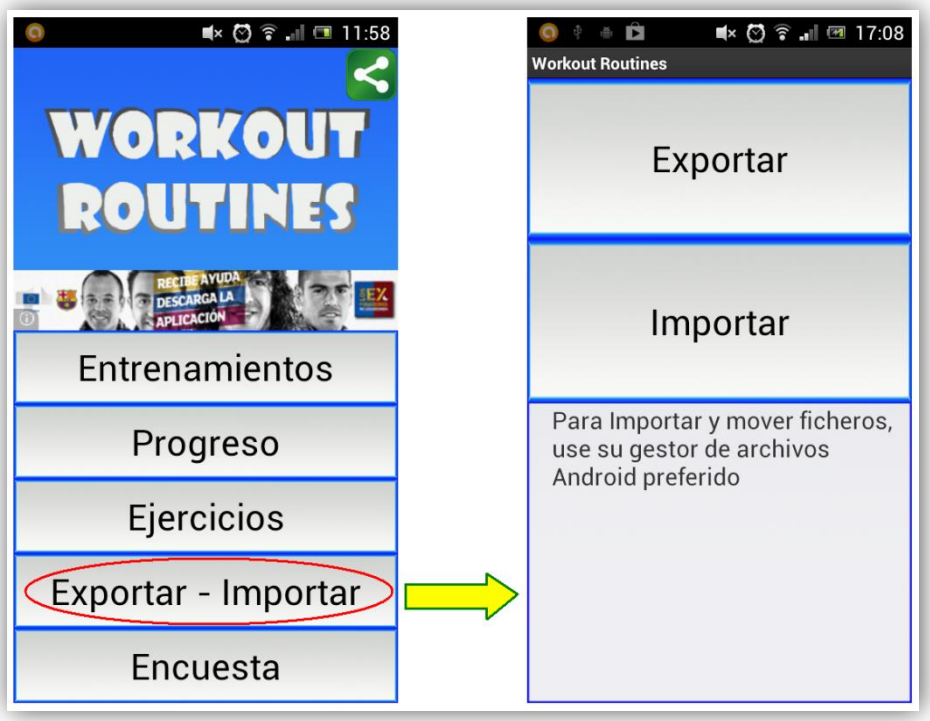



Ilustración 92 - Funcionalidad del botón Exportar – Importar

```
// Boton para exportar e importar
Button bexportar= (Button) findViewById(R.id.export);
bexportar.setOnClickListener(new View.OnClickListener() {

    public void onClick(View v) {
        Intent i = new Intent(getApplicationContext(), ExportarImportar.class );
        startActivity(i);
    }
});
```

Ilustración 93 - Código para ejecutar el botón Exportar - Importar

#### 4.4.5.7. Botón Encuesta

Este botón nos lleva a una actividad, la cual nos sirve de unas pequeñas preguntas simples y fáciles de responder, que son de tipo test y con un cuadro de diálogo para que los usuarios escriban libremente y den una opinión crítica que ayude a mejorar este proyecto. Esta encuesta se envía a un servidor externo, básicamente a un *host* gratuito llamado http://www.rapps.hol.es/encuesta.php/. (El Enlace anterior está protegido mediante autentificación para evitar intrusiones de personas malintencionadas y así evitar daños a la aplicación o al servidor).
Esta funcionalidad la explicaré más detenidamente en los siguientes apartados.

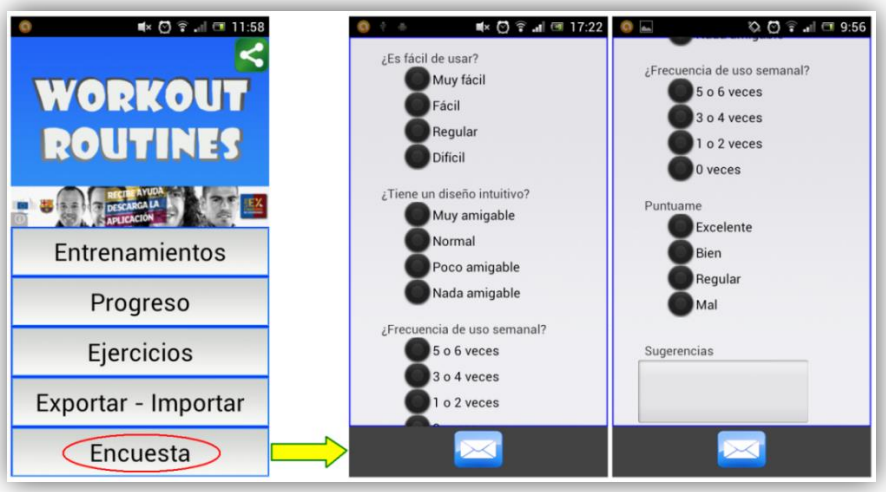



Ilustración 94 - Funcionalidad del botón encuesta

```
// Boton de encuesta, si se envia la encuesta correctamente
// ocultamos el boton, si no, NO

final Button bmi= (Button) findViewById(R.id.bmi);
final Button sondeo= (Button) findViewById(R.id.eval);
if (EncuestaEnviada.enviado){
    sondeo.setVisibility(1000);
}else{
    bmi.setVisibility(1000);  // Hago invisible el boton de Calcuradora
    sondeo.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
             Intent i = new Intent(getApplicationContext(), SondeoSatisfaccion.class );
             startActivity(i);
         }
    });
}
```

Ilustración 95 - Código para ejecutar el botón Encuesta

El código anterior lo que hace es ocultar el Botón de Índice de Masa Corporal hasta que la encuesta haya sido enviada con éxito. Una vez enviado se oculta el botón de ***Encuesta*** y se hace visible el botón ***IMC***. Si no se ha enviado nada, lo que hace es enviarnos a la Actividad *Encuesta*.

Este código pregunta si se ha enviado o no, esto lo hace a través de la variable "***enviado***", que es una variable global, cuyo valor cambiamos a true en el momento de enviar la encuesta.

#### 4.4.5.8. Botón IMC (Índice de Masa Corporal)

Esta herramienta es un obsequio que doy a los usuarios de "***Workout Routines***" para los que hacen la encuesta, ya que esta funcionalidad solo se hace visible si el Usuario rellena y envía el formulario. Este botón lo que hace es calcularnos para un peso y una altura determinada, cuánta grasa aproximada tenemos en el cuerpo.

El resultado lo comparamos con la gráfica que indica las cuatro categorías existentes y allí sabremos nuestro estado.

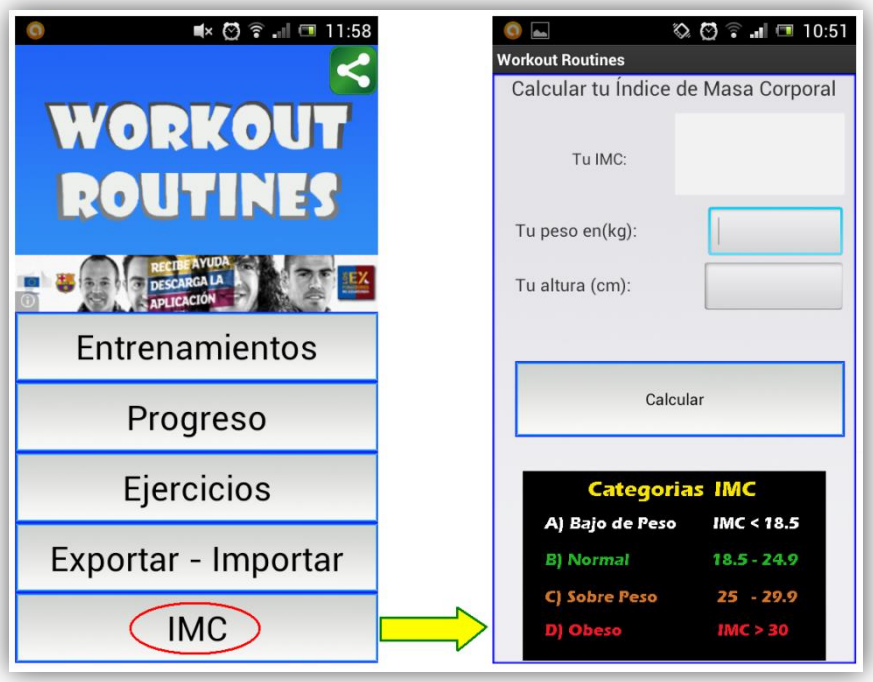



Ilustración 96 - Funcionalidad del botón IMC

```
// Si la encuesta se envia con exito, la Variable Golobal "enviado"
// se pondra a TRUE, haciendo posible usar el Boton de IMC, ya que se hace visible
if (EncuestaEnviada.enviado){
    // Hago visible la Calculadora BMI despues de enviar el formulario
    bmi.setVisibility(0);
    bmi.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(getApplicationContext(), CalculadoraBMI.class );
            startActivity(i);
        }
    });
}
```

Ilustración 97 - Código para ejecutar el botón IMC

El código anterior, lo que hace es preguntar si la encuesta se ha enviado con éxito, esto lo sabe por la variable "*encuesta*", que estará a true. Una vez comprobado esto, lo que hace es hacer visible el botón en cuestión y hacer que se redirija a la clase CalculadoraBMI.

#### 4.4.5.9. Publicidad de AdMob by Google

Estos widget son muy usuales encontrárselos en aplicaciones gratuitas, ya que los desarrolladores esperan ganar dinero o financiarse a través de publicidad, aunque las ganancias suelen ser muy pocas. Estos son anuncios publicitarios que son gestionados por *Google*, donde los anunciantes pagan una pequeña cantidad por cada anuncio visitado.

La idea de introducir publicidad en mi aplicación fue por saber cómo funciona esto y conocer los requisitos que hay que tener para poder implementarlo en cualquier aplicación *Android*. Pues bien, para esto AdMob te ofrece un API y unos tutoriales completamente gratuitos, el cual podemos utilizarlo para implementarlo en nuestra aplicación.

Pasos:

1. Crearse una cuenta en AdMob http://www.google.es/ads/admob/?_adc=ww-es-ha-bk&gclid=CKC9k8eW5rUCFXDKtAodqxsAYA
2. Rellenar nuestros datos de solicitud y datos bancarios, que en mi caso ha sido añadir mi cuenta de *PayPal* para recibir los pagos.
3. Instalación de la API, para esto nos la descargamos y la añadimos a nuestro proyecto. Hacemos clic en el botón derecho sobre el proyecto, y seleccionamos ***Propiedades/Java Build Path/*** **Pestaña *Libraries* / Botón *Add External JARS***

Y seleccionamos la API descargada en la ruta que le especifiquemos. Luego confirmamos en el botón OK o Guardar.

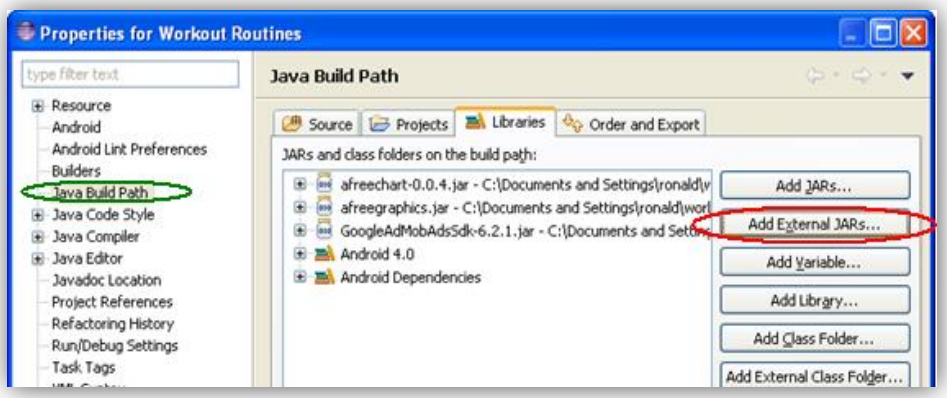



Ilustración 98 - Asignación API de AdMob

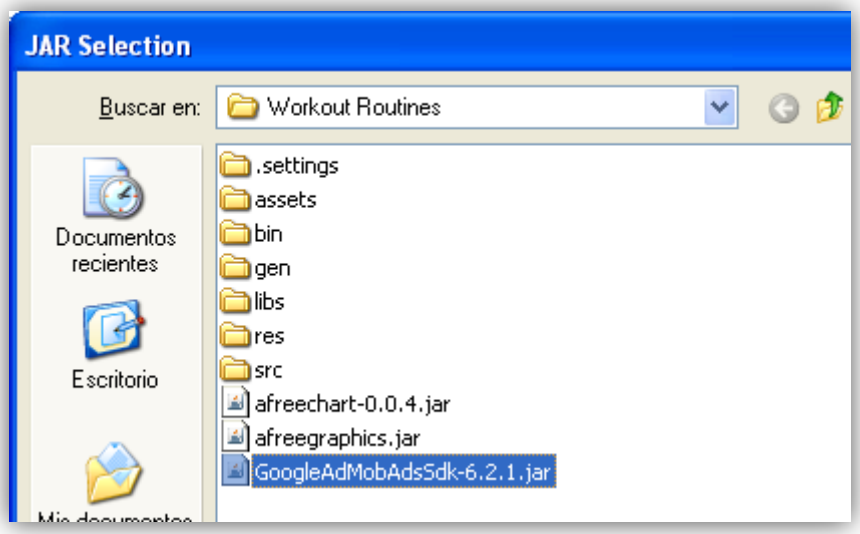



Ilustración 99 - Selección API Google AdMob

4. Luego lo que hacemos es cambiar de Pestaña a ***Order and Export*** y marcamos el cuadro con un tic al API de ***GoogleAdMobAdsSdk***, de esta manera hacemos que durante la compilación, instalación y ejecución, estas librerías se carguen en nuestro proyecto con todas sus clases para evitar problemas, ya que si este paso no se hace, daría error a la hora de ejecutar la publicidad.

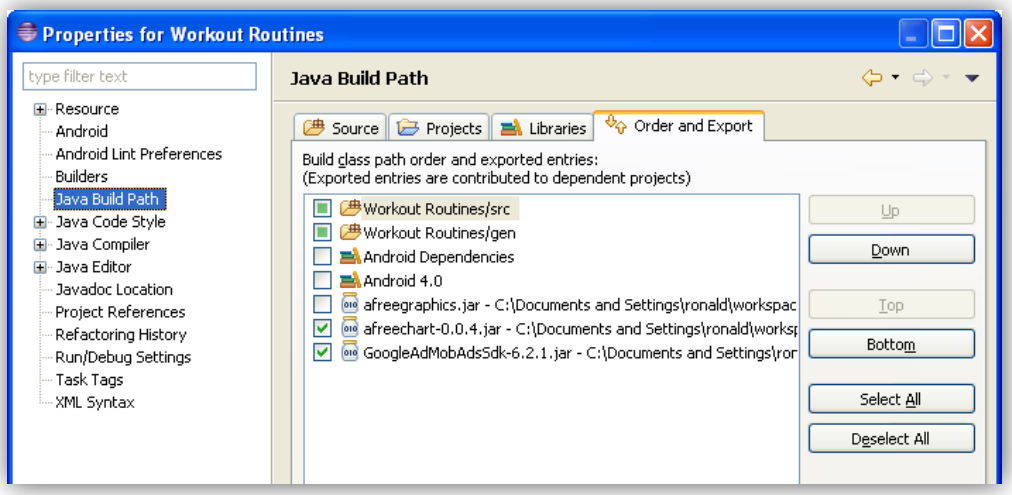



Ilustración 100 - Agregar el Orden en que se exportan las API

5. Crear una publicidad asociada al proyecto, una vez hecho esto, AdMob nos da una clave asociada, la cual introduciremos en nuestro código Java o XML.

Una vez hechos los pasos anteriores, lo que hacemos es configurar y ajustar nuestra publicidad. Para ello la solución que utilicé, fue crear un layout vacío, pero con un fondo personalizado, que dé idea de qué va la aplicación.

Esto se hizo, ya que la publicidad de *Google* solo funciona cuando hay conexión a Internet, de lo contrario, lo que nos saldría sería un fondo simple. En la siguiente imagen muestro dos imágenes, una sin acceso a internet y la otra con acceso. La diferencia consiste en que la de internet muestra una publicidad que se va alternado cada 60 seg, mientras que en la otra solo nos muestra el fondo personalizado que le hemos puesto.

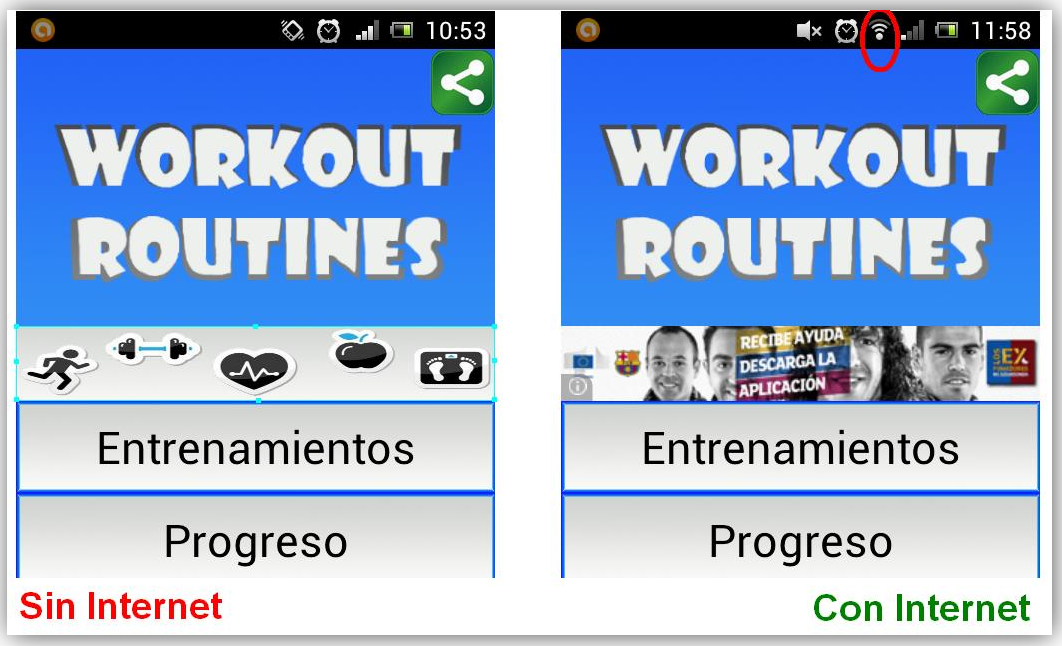



Ilustración 101 - Demostración de la publicidad AdMob

```
<LinearLayout
    android:id="@+id/publicidad"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="0.82"
    android:background="@drawable/publi" >
</LinearLayout>
```

Ilustración 102 - Diseño XML para la Publicidad

El código anterior nos muestra cómo se creó el layout vacío con un fondo personalizado.

Una vez hecho el layout, ahora podemos hacer referencia a este mediante código java. Con el nombre del identificador del layout, las clases de AdMob y el código dado por AdMob, podemos cargar toda la publicidad.

```
//admob widget
AdView adView = new AdView(this, AdSize.BANNER, "a1511028de67183");
// Hacemos referencia al linear layout vacio
LinearLayout layout = (LinearLayout)findViewById(R.id.publicidad);
layout.addView(adView);// Add the adView to it
adView.loadAd(new AdRequest());
```

Ilustración 103 - Código Java para ejecutar la publicidad automáticamente

- En la primera línea de código lo que hacemos es crear un objeto de la Clase AdView, con los siguientes parámetros (Contexto, tipo de publicidad (Banner) y Código del Desarrollador).
- En la segunda línea usamos un objeto de tipo Layout, haciendo referencia a un identificador de tipo layout creado en XML para que este contenga la información del banner publicitario.
- En la tercera y última línea de código, lo que hacemos es cargar el banner publicitario y agregarlo al layout contenedor.

### *4.4.5.10. Inicialización de la BD a valores por defecto*

```
// Esta linea se hace una vez, ya que insertamos solo una vez
EjerciciosEstaticos.insertarEnFormas(this);
String idioma = getResources().getConfiguration().locale.getDisplayName().trim();

if(idioma.equals("español (España)")){
    EjerciciosEstaticos.inicioBD(this);
}else{
    EjerciciosEstaticos.inicioBDIngles(this);
}
```

Ilustración 104 - Código para inicializar la BD del dispositivo móvil

El código anterior inicializa la base de datos de nuestro dispositivo móvil, a un estado de valores por defecto, esto ayuda a los usuarios a tener una idea intuitiva de lo que tienen que hacer con nuestra aplicación. Los valores estarán en inglés o español, dependiendo de la configuración de idioma del móvil. Para saber esto lo que hacemos es preguntar en los recursos del móvil la configuración que tiene, si es en español lo que hacemos es llamar a un método que insertara valores en castellano y si no, llamará al otro método que los insertará en inglés.

### 4.4.6. Construcción y Diseño de la BD SQLite


Ilustración 105 - Logo SQLite con Android

Antes de continuar con los otros apartados donde detallaré cómo ha sido el desarrollo de las otras actividades que componen esta *App*, me gustaría explicar el diseño de la base de datos, más que nada, porque es fundamental tenerla construida para luego hacer referencia a ella, ya sea para almacenar los entrenamientos de los usuarios o las consultas o modificaciones que ellos desean hacer.

#### 4.4.6.1. Que es SQLite

*SQLite* es una base de datos de código abierto que está integrada en *Android*. *SQLite* es compatible con las características estándar de bases de datos relacionales, como la sintaxis *SQL*, transacciones y declaraciones ya preparadas. Además, solo requiere un poco de memoria en tiempo de ejecución (aprox. 250 Kb).

Además, al estar integrada en el *Api* de *Android*, no es necesario descargarse ningún complemento adicional. Por otro lado, *SQLite* proporciona compatibilidad con los tipos de datos que se trabaja en *Java*, como lo son los de tipo ***String***, ***Integer***, ***Double*** o ***Boolean***.

*SQLite*, al ser una versión muy pequeña, no posee un entorno gráfico tan potente como otros gestores de base de datos, pero esto no representa una desventaja a la hora del trabajar con ella, ya que lo que nos interesa es almacenar pequeños datos de manera segura y consistente.

#### 4.4.6.2. Configuración SQLite Android

Antes que nada, hay que configurar en nuestro entorno de programación (*Eclipse*) un explorador de archivos o ficheros, éste nos ayudará en todo momento a saber si la base de datos ha sido creada correctamente en el dispositivo móvil.

Para ello, vamos a la barra de herramientas de *Eclipse*: ***Window/ Show View / Other*** y en la ventana que aparece nos ubicamos en la carpeta que pone *Android* y seleccionamos el nombre ***File Explorer***, que está junto con el logo de *Android*. El resultado se muestra en la Ilustración 106.

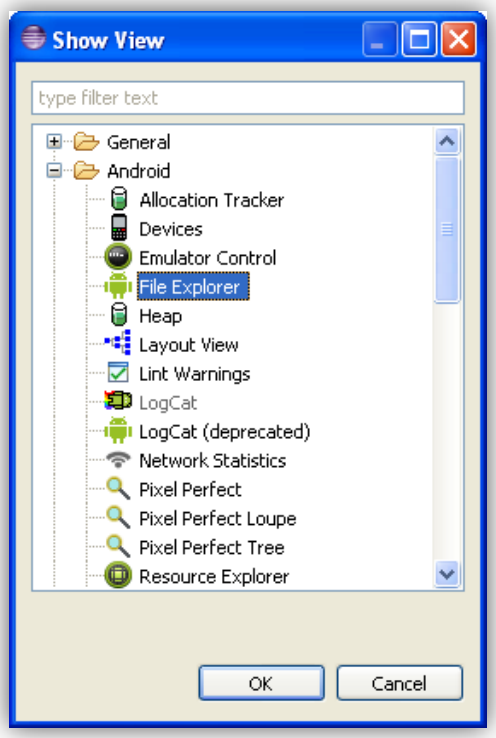



Ilustración 106 - Selección de una nueva pestaña, File Explorer

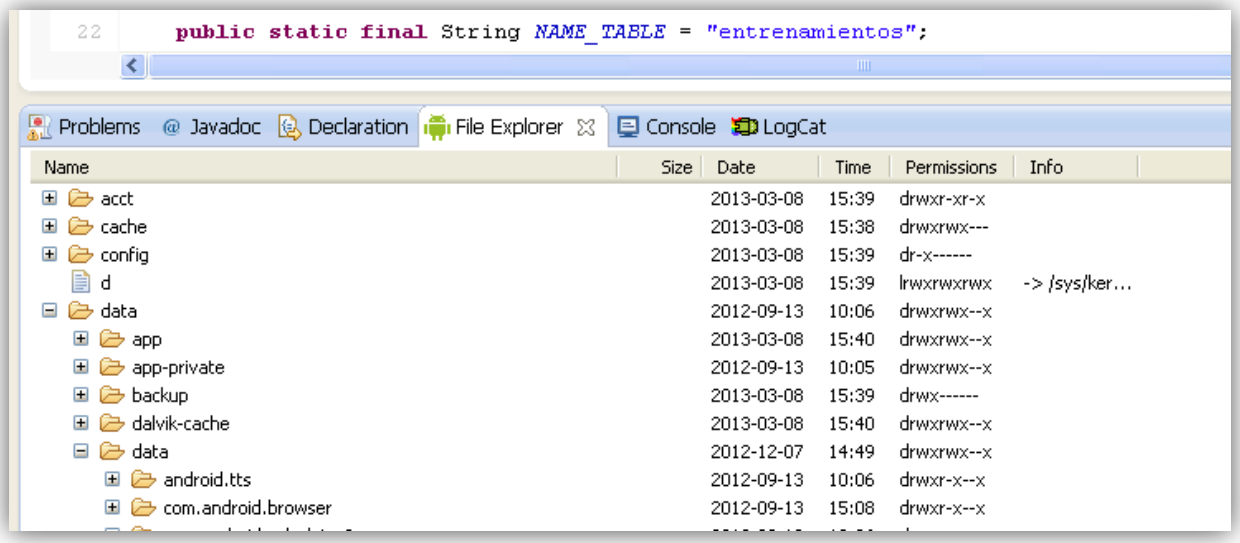



Ilustración 107 - Forma estructural de la pestaña de trabajo, File Explorer

En este explorador de ficheros y directorios, podremos ver las bases de datos creadas en nuestro dispositivo móvil, Se puede apreciar que hay diferentes BD y que estas corresponden a otras aplicaciones.

El siguiente paso es visualizar los elementos (Tablas) que contiene nuestra base de datos. Para ello tendremos que tirar de la ayuda de una herramienta que dispone Android (***adb.exe***), pero bajo la ayuda de la línea de comandos de Windows CMD.

Lo primero que hay que hacer es ejecutar el terminal de Windows y ubicarnos en la siguiente ruta mediante el comando "***CD***". Cabe decir que la ruta cambia para cada equipo, ya que depende de donde hayan instalado el API de Android.

Lo importante de esto es saber ubicar el archivo adb.exe.

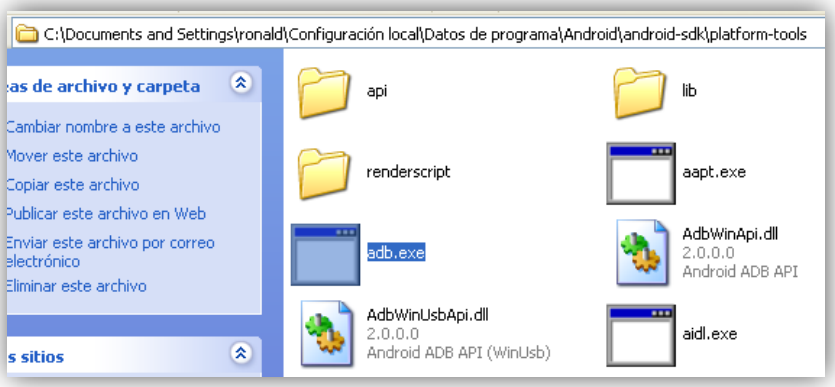



Ilustración 108 - Fichero a ejecutar para trabajar directamente con la BD de SQLite

Pasos:

1) Ubicarnos en la ubicación de donde se encuentre el ejecutable adb.exe

> **cd C:\Documents and Settings\ronald\Configuración local\Datos de programa\Android\android-sdk\platform-tools**

2) Ejecutamos el programa mencionado mediante este comando. Este comando nos lleva a ubicarnos en la Shell de SQLite, allí podremos usar sus propios métodos para encontrar y ejecutar lo que queremos.

> **adb shell**

3) Por último ubicamos la ruta donde se almacena nuestra base de datos. La podemos ubicar por el nombre del paquete (**com.workout.routines.activitys**). Usando un comando propio de SQLite, podemos ubicarnos en la siguiente ruta donde se guarda nuestra base de vados, en nuestro caso se llama "***workouts***".

**# sqlite3 /data/data/com.workout.routines.activitys/databases/workouts**

Para saber que estamos donde queremos estar, al completar estos tres sencillos pasos, nos llevará a un Shell diferente al de Windows `C:\>` , ya que nos tiene que mostrar algo igual a esto: `sqlite>`

La siguiente imagen muestra los pasos comentados anteriormente, dando como resultado final un Shell con comandos propios de *SQLite* para *Android*.

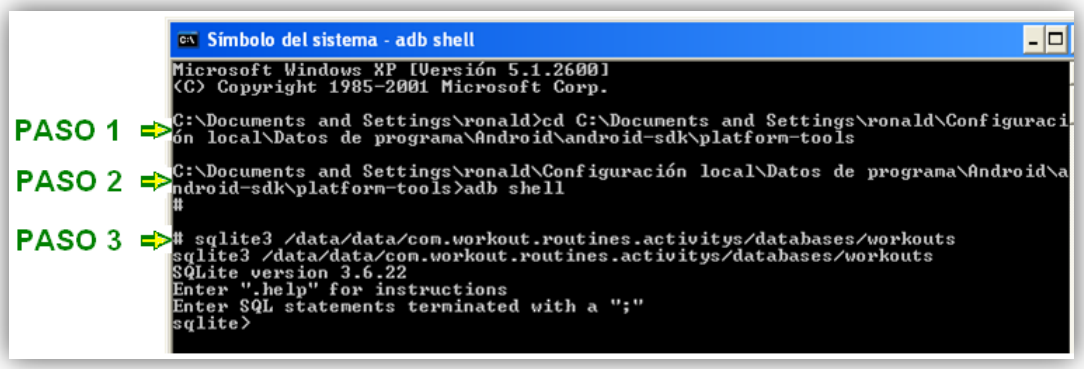



Ilustración 109 - Visualización de los pasos ejecutados

La ruta del paso tres la podemos extraer mirando el explorador de fichero de *Eclipse*. Tal cual como muestro en la siguiente imagen. Cabe recordar que el explorador de ficheros solo es visible cuando el terminal virtual de Android se está ejecutando con nuestra aplicación, de lo contrario mostraría una ventana en blanco.

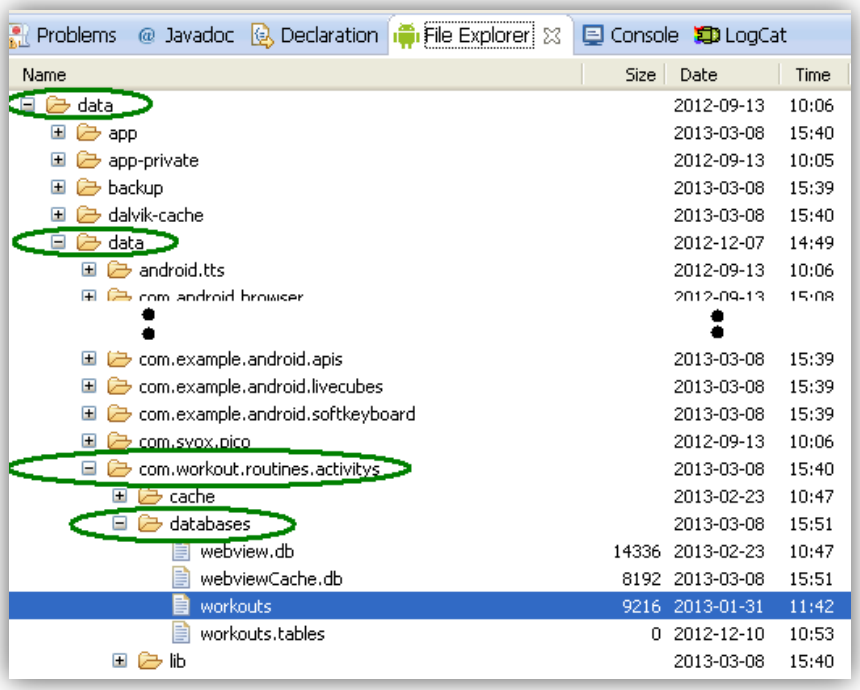



Ilustración 110 - Estructura de directorios de nuestra BD interna

Una vez dentro de lo que sería *SQLite*, podremos ver nuestras tablas y hacer operaciones sobre ellas, como la de consultar, modificar o eliminar. Para ello escribimos en la línea de comandos ***(.tables)*** tal cual se muestra en la siguiente figura.

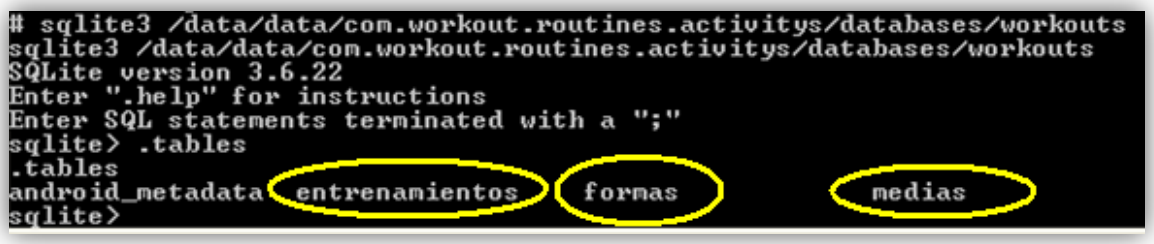



Ilustración 111 - Visualización de la correcta creación de las tablas en nuestro móvil

La imagen anterior nos muestra las tres tablas creadas, las cuales son: **entrenamientos, formas** y **medias*; é***stas las iré explicando más detenidamente en los siguientes ítems.

Ahora voy a mostrar una pequeña imagen donde muestro un ejemplo de cómo es la estructura de *SQLite*.

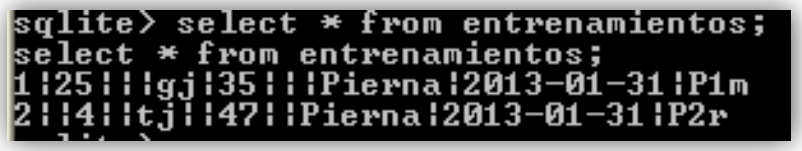



Ilustración 112 - Ejecución de un Select, dentro del terminal SQLite

Como se puede apreciar, la estructura es muy simple, basta con hacer un simple ***select*** para ver las columnas que componen la tabla consultada, estas están separadas por una barra espaciadora y sin nombre alguno que detalle cuál es cada columna.

En esta línea de comandos podemos ejecutar las órdenes propias de *SQL*. Como los ***INSERT, UPDATE, DELETE, DROP***, etc.

#### 4.2.6.3. Programación SQLite

Para empezar a usar el gestor de bases de datos *SQLite* primero hay que crear una clase que extienda de esta utilidad (***SQLiteOpenHelper***) y allí dar un nombre a la base de datos, y crear las tablas correspondientes a esa base de datos con sus elementos correspondientes.

El nombre de esta Clase la he llamado **SQLiteBD.Java**, ya que de esta manera me sería muy fácil reconocerla y está incluida en un paquete independiente llamado (**com.workout.routines.bd**).

La siguiente imagen muestra cómo es la estructura inicial de la tabla **entrenamientos**, junto con sus atributos y el nombre de la base de datos que le hemos dado. Extendemos de la Clase ***SQLiteOpenHelper*** para acceder a todas las funciones o métodos que nos ofrece *SQLite*.

```
package com.workout.routines.bd;

import java.text.SimpleDateFormat;

public class SQLiteBD extends SQLiteOpenHelper {

    // No usar extencion ".db" DA PROBLEMAS !!
    // Nombre de Nuestra Base de Datos
    private static final String DATABASE_NAME = "workouts";

    // Nombre de la Tabla Principal con sus Columnas correspondientes
    public static final String NAME_TABLE = "entrenamientos";
    public final static String KEY_ID = "_id";
    private static final String SERIEM = "seriesm";
    private static final String SERIER = "seriesr";
    private static final String SERIET = "seriest";
    private final static String EJERCICIO = "ejercicio";
    private final static String PESOM = "pesom"; // Peso Musculacion
    private final static String PESOR = "pesor"; // Peso Resistencia
    private final static String PESOT = "pesot"; // Peso Tonificacion
    private final static String GRUPO = "grupo"; // Grupo Muscular seleccionado
    private final static String FECHA = "fecha";
    private final static String PESTAÑA = "pestaña";
    private final static int VERSION_BD = 2;
```

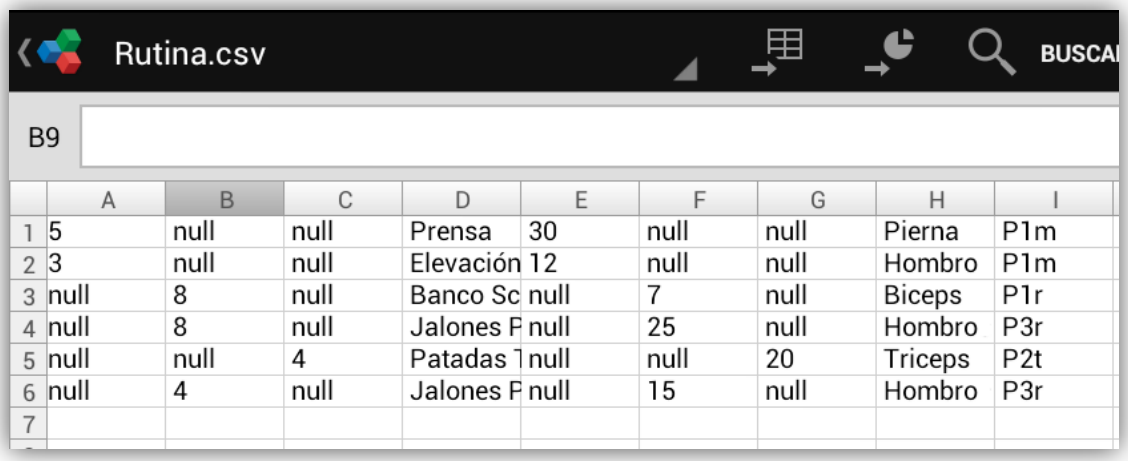
Ilustración 113 - Variables para la construcción de la tabla "entrenamientos"

Recordar que las tablas de entrenamientos suelen tener tres atributos importantes que hay que rellenar, como son las Series, Nombre de Ejercicio y el Peso asociado a este. Es por eso que en la tabla entrenamientos he querido reflejar esto.

A continuación voy a explicar todos los atributos que componen la tabla de entrenamientos.

- **_id:** es el identificador y clave primaria de la tabla entrenamientos, es de tipo *INTEGER* y está configurado como un *AUTOINCREMENT*, ya que de esta manera controlo de manera automática cada vez que inserto una nueva fila en esta tabla. También controlo cuándo elimino una o varias. Esto lo hago para mantener un orden secuencial de identificadores.
- **seriem, serier, seriet:** estos tres atributos corresponden con las series de cada ejercicio de entrenamiento muscular, de resistencia o tonificación respectivamente. Es de tipo *TEXT* ya que la función de ésta es mostrar información de cuántas series o repeticiones necesita hacer un usuario por cada entrenamiento elegido. Al inicio se empezó con un solo atributo que diferenciara los tres, pero como el proyecto fue evolucionando, al final se decidió por tener tres diferentes. Este puede ser un dato nulo, ya que los usuarios no están obligados a introducir un peso si no lo desean.
- **ejercicio:** Éste es muy importante, ya que refleja el nombre del entrenamiento a seguir, aunque solo no dice mucho, ya que tiene que estar acompañado de una serie y su peso correspondiente. Es de tipo *TEXT*.
- **pesom, pesor, pesot**: Estos tres atributos reflejan el peso que hay que usar en cada entrenamiento, ya sea para musculación, resistencia o tonificación respectivamente. Es de tipo TEXT, pero para hacer operaciones sobre éstas hago la conversión a INTEGER. Este puede ser un dato nulo, ya que los usuarios no están obligados a introducir un peso si no lo desean.
- **grupo**: este atributo es el encargado de almacenar a qué grupo muscular pertenece cada ejercicio, este atributo es útil a la hora de hacer consultas pertinentes con los grupos musculares más usados. Éste almacenará los grupos: *pierna*, *hombro*, *tríceps*, *bíceps*, *espalda* y *pecho*.
- **fecha**: este atributo es importante, ya que sirve para reflejar el progreso de cada entrenamiento, separado por fechas, ya que cada vez que se inserta, modifica o actualiza un entrenamiento en cuestión, este atributo cambia (Actualiza por la fecha actual) ayudando a saber en cada momento si ha habido mejora o no. Es de tipo *DATE* y lo tengo de manera automática con *DEFAULT CURRENT_DATE*.
- **pestaña**: Este atributo es también muy importante, ya que me ayuda a saber en cada momento que fila corresponde a qué tipo de entrenamiento y a que pestaña, ya que a la hora de hacer la consulta por mostrar los datos, estos los tengo que separar en entrenamientos de musculación, resistencia o tonificación, además de identificar a que pestaña corresponde, para que no hayan colisiones de datos y produzcan errores.

Ahora mostraré un ejemplo de cómo sería una tabla de entrenamientos rellenada.




| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | null | null | Prensa | 30 | null | null | Pierna | P1m |
| 2 | 3 | null | null | Elevación | 12 | null | null | Hombro | P1m |
| 3 | null | 8 | null | Banco Sc | null | 7 | null | Biceps | P1r |
| 4 | null | 8 | null | Jalones F | null | 25 | null | Hombro | P3r |
| 5 | null | null | 4 | Patadas T | null | null | 20 | Triceps | P2t |
| 6 | null | 4 | null | Jalones F | null | 15 | null | Hombro | P3r |
| 7 | | | | | | | | | |

Ilustración 114 - Ejemplo de la tabla entrenamientos

La imagen anterior no muestra cómo se llama cada atributo, ya que *SQLite Android* no posee un entorno gráfico potente como otros gestores, ya que es una versión liviana y lo que pretende es dar un servicio para dispositivos de poca memoria.

Como había comentado antes, el orden de las columnas es el mismo orden que se dio a la hora de crear las tablas, las columnas de la "A" - "I" corresponden a los siguientes atributos: **seriem**, **serier**, **seriet**, **ejercicio**, **pesom**, **pesor**, **pesot**, **grupo** y **pestaña** respectivamente.

Para este ejemplo no muestro los atributos **_id** y **fecha**, ya que éstos son automáticos, ya que de las inserciones se encarga el propio gestor de base de datos de *Android*.

#### *4.2.6.3.1. Creación del Constructor*

```
private final static int VERSION_BD    = 2;
private static final String TAG = "SQLiteBD";

public SQLiteDatabase db;

// Constructor
public SQLiteBD(Context context) {
    super(context, DATABASE_NAME , null, VERSION_BD);
}
```

Ilustración 115 - Constructor de la clase SQLiteBD

La imagen anterior muestra cómo creamos un constructor para nuestra clase, haciendo **super** hacemos referencia a todos los métodos de la clase heredada.

Para ello le pasamos el nombre de la base de datos, un contexto de trabajo y una versión de evolución a cambios en la propia *BD*. Este último es un valor de tipo entero, que lo que quiere decir es que cada vez que manipulemos la base de datos, ya sea para eliminar, renombrar una tabla o columna, este valor debe ser incrementado en uno, para que se hagan efectivos los cambios en las actualizaciones de los usuarios. Si este valor no se incrementara, produciría un error y la aplicación no funcionaria.

Para que *Android* sepa que hay una nueva modificación importante sobre su base de datos, basta con sobrescribir el método ***onUpgrade*** con una línea de eliminación de tablas con comandos SQL, esto lo hace si comprueba que existe una nueva versión, al comprobar las variables *oldVersion* y *newVersion*. Después de ejecutar este método procederá a ejecutar el otro método ***OnCreate*.** La siguiente imagen muestra cómo se hace este proceso.

```
// Se llama a esta funcion cuando la version de la BD es actualizada.
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Eliminamos la tabla si ya existia por otra mas Actualizada

    Log.w(TAG, "Actualizando version de la Base de Datos " + oldVersion + " to "
            + newVersion + ", La cual sera destruida.");

    db.execSQL("DROP TABLE IF EXISTS entrenamientos;");

    db.execSQL("DROP TABLE IF EXISTS formas;");

    db.execSQL("DROP TABLE IF EXISTS medias;");


    onCreate(db);
}
```

Ilustración 116 - Método onUpgrade

#### *4.2.6.3.2. Creación de las tablas que componen nuestra app.*

Para crear las tablas hay que sobrescribir el método **OnCreate** de SQLite, este método nos permite crear nuestras tablas, y para ello solamente nos basta con introducir en una variable *String* la consulta *SQL* de creación de tablas.

La siguiente imagen muestra cómo hemos hecho el comando **CREATE** de *SQL* para la creación de las tablas **entrenamientos**, **formas** y **medias.** Luego basta con ejecutar el comando **execSQL** que es el que nos ejecuta las estas líneas código *Java* (*String*) con estructura *SQL* en la propia *BD* de *Android*.

```
@Override
public void onCreate(SQLiteDatabase db) {
    try {

        String creaEntrenamientos = "create table entrenamientos " +
                "(_id integer PRIMARY KEY AUTOINCREMENT, " +
                "seriesm TEXT, " +
                "seriesr TEXT, " +
                "seriest TEXT, " +
                "ejercicio TEXT, " +
                "pesom TEXT, " +
                "pesor TEXT, " +
                "pesot TEXT, " +
                "grupo TEXT, " +
                "fecha DATE default CURRENT_DATE, " +
                "pestaña TEXT); " ;

        String creaFormas = "create table formas " +
                "(_id integer PRIMARY KEY AUTOINCREMENT, " +
                "seriesm TEXT, " +
                "seriesr TEXT, " +
                "seriest TEXT, " +
                "ejercicio TEXT, " +
                "PesoNulo TEXT, " +
                "grupo TEXT);" ;

        String creaMedias = "create table medias " +
                "(fecha DATE , " +
                "m1 TEXT, " +
                "m2 TEXT, " +
                "m3 TEXT, " +
                "m4 TEXT, " +
                "m5 TEXT, " +
                "m6 TEXT);" ;

        // ejecutamos los comandos de crear tablas
        // Con la estructura de SQL
        db.execSQL(creaEntrenamientos);
        db.execSQL(creaFormas);
        db.execSQL(creaMedias);
    }
    catch (SQLException e) {
         e.printStackTrace();
    }
}
```

Ilustración 117 - Método onCreate

#### *4.2.6.3.3. Funciones Sobre la Base de Datos SQLite Tabla Entrenamientos*

En este apartado voy a mostrar los métodos más importantes que he construido para el desarrollo de esta *App.* Estas funciones se encargan de insertar, eliminar o actualizar la base de datos de mi aplicación.

**Función insertar_bd.**

```
public long insertar_bd(String serie, String ejercicio,
        String peso, String grupo, String entrenamiento,
        String pestaña) {

    //db = this.getWritableDatabase();
    abrir(); // Para poder insertar en la BD
    ContentValues cv = new ContentValues();

    if(entrenamiento == "musculacion"){

        // Con trim(), quitamos los posibles espacios
        // en blanco al Inicio o Final
        cv.put(SERIEM, serie.trim());
        cv.put(EJERCICIO, ejercicio.trim());
        cv.put(PESOM, peso.trim());
        cv.put(GRUPO, grupo.trim());
        cv.put(PESTAÑA, pestaña.trim());

    }else if(entrenamiento == "resistencia"){

        cv.put(SERIER, serie.trim());
        cv.put(EJERCICIO, ejercicio.trim());
        cv.put(PESOR, peso.trim());
        cv.put(GRUPO, grupo.trim());
        cv.put(PESTAÑA, pestaña.trim());

    }else{

        cv.put(SERIET, serie.trim());
        cv.put(EJERCICIO, ejercicio.trim());
        cv.put(PESOT, peso.trim());
        cv.put(GRUPO, grupo.trim());
        cv.put(PESTAÑA, pestaña.trim());

    }
    // Nombre de las columnas a introducir los valores

    return db.insert("entrenamientos", null, cv);
}
```

Ilustración 118 - Método insertar_bd

La función anterior es la encargada de insertar elementos en la *BD*, y es muy importante ya que inserta los datos de tal manera que reconoce en cada momento para qué ejercicio es apropiado, ya sea Musculación, Resistencia o Tonificación, dejando los otros elementos a nulos en caso de que no precedan.

La razón, es que estos valores en un futuro me gustaría utilizarlos para dar más información acerca del progreso de los usuarios, además, así puedo controlar en cada momento qué es que cada cosa (fila).

En un principio se tenía solo los atributos ***SERIE*** y ***PESO***, no existían los repetidos, *SERIEM, SERIER, RESIET, PESOM, PESOR* y *PESOT* respectivamente, pero dado el caso de que esta aplicación evolucione, es preferiblemente tener estos bien separados y definidos. Por lo que cualquier cambio en la base de datos, significa que la base de datos será eliminada para volver a crear la nueva.

Se puede observar que no uso los atributos _id y fecha, ya que estos son automáticos por así decirlo y se insertan ellos mismos por cada registro introducido.

Para insertar en la base de datos me valgo de este método que he creado. Con esta función solo necesito pasar la serie, el ejercicio, el peso, el tipo de entrenamiento y la pestaña en que se encuentra. Con la clase ContenValues creo un objeto al que le voy agregando los parámetros pasados y con el comando ***insert()*** propio de la clase SQLite, inserto estos datos en la tabla seleccionada.

Una curiosidad al trabajar con la BD SQLite, es que cada vez que necesito operar sobre ella necesito abrirla o cerrarla, para ello uso los comandos ***getWritableDatabase***() y ***close***().

Esta función es un poco especial, ya que no puedo cerrar la BD dentro de ella, sino que la necesito cerrarla fuera (una clase externa que use este método) por lo que al tener que insertar muchas filas a la vez y tener que abrir y cerrar por cada inserción, esto resultaría ser muy costoso y lento, así que para ello lo que hago es abrir la BD y una vez finalice las inserciones pertinentes, la cierro de la clase que la ha invocado. En realidad sí que puedo cerrarla dentro de ella, pero por hacer más ágil la aplicación, prefiero cerrarla desde fuera de esta. Caso contrario que ocurre con las demás, donde sí que las cierro dentro de ellas, olvidándome de tener que cerrarlas una vez que use estas funciones.

**Función getNumRows.**

```
// Devuelve el Numero de Filas que hay en una tabla
public int getNumRows()
{
    db = this.getWritableDatabase();

    Cursor q =  db.query(NAME_TABLE, new String[] {KEY_ID},
            null, null, null, null, null);


    int i = q.getCount();
    cerrar();
    return i;

}
```

Ilustración 119 - Método getNumRows

Este método nos devuelve cuantas filas o ejercicios tenemos creadas en la BD de la *App.*

Antes hay que saber qué es un Cursor. Un Cursor es una colección de filas y es necesario conocer y saber el tipo de cada columna, ya que los cursores solo devuelven las filas con sus correspondientes columnas seleccionadas. Yo prefiero llamarlo un puntero a la Base de Datos, que me devuelve tras hacer una consulta con el comando **query**(), porque de esta manera me ayuda a pensar de una forma más clara, además es la abstracción que le dado. Todas las consultas por norma general devuelven un elemento de tipo cursor.

Para hacer esto, basta con hacer una consulta a través del comando ***query*** propio de *SQLite* a la clave primaria de la tabla tratara y recoger su Cursor, una vez sabiendo el Cursor, podemos preguntar cuántos elementos nos ha devuelto con el comando **getCount**() de la clase Cursor, que nos devolverá un entero mayor o igual que cero.

**Función mostrarFilasMatriz**

```
public String[][] mostrarFilasMatriz() {
    // [Filas = ?][Columnas = 10]
    String[][] lista = new String[getNumRows() + 1][10];

    // Donde empiezan las filas de la tabla entrenamientos
    int i = 1;
    do {
        Cursor cr = getRow(i);

        String seriem = cr.getString(1);
        String serier = cr.getString(2);
        String seriet = cr.getString(3);
        String ejercicio = cr.getString(4);
        String pesom = cr.getString(5);
        String pesor = cr.getString(6);
        String pesot = cr.getString(7);
        String grupo = cr.getString(8);
        String pestaña = cr.getString(9);

        lista[i][1] = seriem;
        lista[i][2] = serier;
        lista[i][3] = seriet;
        lista[i][4] = ejercicio;
        lista[i][5] = pesom;
        lista[i][6] = pesor;
        lista[i][7] = pesot;
        lista[i][8] = grupo;
        lista[i][9] = pestaña;

        i++;

    } while (i <= getNumRows());

    return lista;
}
```

Ilustración 120 - Método mostrarFilasMatriz

Esta función la utilizo para extraer los datos que deseo sacar de la BD de la app, se conoce que tengo Diez atributos importantes que serán el número de columnas, pero no sé cuántas filas o elementos de estos tendré, así que me creo una matriz de la cantidad de filas que tenga más uno con la función **getNumRows()** explicada anteriormente. Esta función me va construyendo una matriz que luego será retornada para leer la información de una manera más rápida, ya que solo hago referencia a los elementos que quiero sacar, que en este caso serán todos, e ir construyendo mis datos a exportar. Esto lo profundizaré cuando haga el apartado de Exportar e Importar.

**Funciones resetAllId y setId**

```
public void resetAllId() {
    // Reseteamos el AutoIncrement del Campo "_id" a 1
    abrir();
    db.execSQL("delete from sqlite_sequence where name='entrenamientos';");
    cerrar();
}

public void setId(String id) {
    // Reseteamos el AutoIncrement, y lo ponemos de donde queremos que
    // empiece a contar
    abrir();
    db.execSQL("UPDATE SQLITE_SEQUENCE SET seq = " + id
            + " where name='entrenamientos';");
    cerrar();
}
```

Ilustración 121 - Métodos resetAllid y setId

Estas funciones para nuestra *App* de base de datos son muy importantes, ya que al tener un **_id** que es **AUTO_INCREMENT**, éste siempre intentará contar desde donde esté su valor actual. Por tanto, cuando una o varias filas son eliminadas, este contador de alguna manera tendrá que descontarse, es por tanto que me he creado estas dos funciones, donde una me resetea todo el id a Cero o desde su posición original para que empiece a contar desde Uno, y la otra, tengo la opción de elegir el punto desde donde quiero que empiece a contar.

Para resetear por completo el contador del Auto_Increment, lo que hago es ejecutar un comando propio de SQL, que es:
**delete from sqlite_secuence where name = 'NombreTabla';**
Este comando lo ejecuto dentro una instrucción que se llama ***execSQL***() la cual me permite hacer la instrucción DELETE dentro de *Android*.

Ahora, para elegir el punto de donde quiero que empiece a contar, hago la siguiente instrucción con el comando UPDATE propio de SQL:
**update sqlite_secuence set seq = NumSecuencia where name = 'NombreTabla';**

La imagen anterior muestra cómo hemos hecho estos dos métodos, donde ambos son usados para hacer nuestro trabajo.

## Función eliminarUnaFilaCorrector

```
public boolean eliminarUnaFilaCorrector(long rowId, String entrenamiento) {

    boolean salida = false, flag = false;
    String seriem = "", serier = "", seriet = "", ejercicio = "", pesom = "",
           pesor = "", pesot = "", grupo = "", pestñ = "";

    int next = (int) rowId + 1; // Siguiente elemento a consultar
    int act = (int) rowId; // fila a actualizar, (Actual - Anterior)

    while (getNumRows() >= next) {

        Cursor c = getRow(next);

        if(c.getString(1) == null || c.getString(1) == "") seriem    = "";
        else seriem    = c.getString(1).trim();
        if(c.getString(2) == null || c.getString(2) == "") serier    = "";
        else serier    = c.getString(2).trim();
        if(c.getString(3) == null || c.getString(3) == "") seriet    = "";
        else seriet    = c.getString(3).trim();
        if(c.getString(4) == null || c.getString(4) == "") ejercicio = "";
        else ejercicio = c.getString(4).trim();
        if(c.getString(5) == null || c.getString(5) == "") pesom     = "";
        else pesom     = c.getString(5).trim();
        if(c.getString(6) == null || c.getString(6) == "") pesor     = "";
        else pesor     = c.getString(6).trim();
        if(c.getString(7) == null || c.getString(7) == "") pesot     = "";
        else pesot     = c.getString(7).trim();
        if(c.getString(8) == null || c.getString(8) == "") grupo     = "";
        else grupo     = c.getString(8).trim();
        if(c.getString(9) == null || c.getString(9) == "") pestñ     = "";
        else pestñ     = c.getString(9).trim();
        if (pestñ.trim().equals("P1m") || pestñ.trim().equals("P2m")|| pestñ.trim().equals("P3m")) {
            entrenamiento = "musculacion";
        }
        if (pestñ.trim().equals("P1r") || pestñ.trim().equals("P2r")|| pestñ.trim().equals("P3r")) {
            entrenamiento = "resistencia";
        }
        if (pestñ.trim().equals("P1t") || pestñ.trim().equals("P2t")|| pestñ.trim().equals("P3t")) {
            entrenamiento = "tonificacion";
        }

        // Actualizamos la tabla, subiendo escalones,
        if (entrenamiento == "musculacion") {
            flag = actualizarFila(act, seriem, ejercicio, pesom, grupo, pestñ, "musculacion");
        } else if (entrenamiento == "resistencia") {
            flag = actualizarFila(act, serier, ejercicio, pesor, grupo, pestñ, "resistencia");
        } else {
            flag = actualizarFila(act, seriet, ejercicio, pesot, grupo, pestñ, "tonificacion");
        }

        seriem = "";
        serier = "";
        seriet = "";
        ejercicio = "";
        pesom = "";
        pesor = "";
        pesot = "";
        grupo = "";
        pestñ = "";

        next++; // Siguiente elemento
        act++;  // Anterior elemento

    }

    // eliminamos la ultima fila y seteamos el aunto-increment
    if (flag) {
        salida = eliminarUnaFila(act);    // Sera el ultimo elemento de la fila
        setId(Integer.toString(act - 1)); // Setamos el aunto-increment al
                                          // anterior elemento eliminado
    }

    return salida;
}
```

Ilustración 122 - Método eliminarUnaFilaCorrector

La imagen con el código anterior, lo que pretende hacer es que cada vez que eliminamos una fila que está en la mitad de otras filas, ésta intenta corregir el *AUTO_INCREMENT*, además de dejar las filas existentes consecutivas, evitando que queden huecos entre estas. Esto es para evitar que tengan todos sus atributos a nulos y que no nos aporten nada. Para ayudarnos a eliminar una fila usamos el siguiente método:

```
public boolean eliminarUnaFila(long rowId) {
    // Eliminar una fila en concreto
    db = this.getWritableDatabase();
    boolean salida = db.delete(NAME_TABLE, KEY_ID + "=" + rowId, null) > 0;
    cerrar();
    return salida;
}
```

Ilustración 123 - Método eliminarUnaFila

Este método de eliminar es sencillo, ya que solo elimina la fila con el ID que le pasemos, y devuelve "**True**" si la fila ha sido eliminada correctamente, de lo contrario devolverá "**False**".

Pues bien, el proceso que hace es el siguiente. Primero comprobamos que la fila a eliminar sea la última que queda. Si este es el caso, lo que hacemos es llamar a la función **eliminarUnaFila**() y seteamos su id a Cero. Si este no es el caso, lo que hacemos es preguntar por el **id** en cuestión y por el entrenamiento seleccionado y pasárselo a esta función. Esta lo que hará es trabajar con dos identificadores que apunten al elemento tratado y al siguiente.


| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | null | null | Prensa | 30 | null | null | Pierna | P1m |
| 2 | 3 | null | null | Elevación | 12 | null | null | Hombro | P1m |
| 3 | null | 8 | null | Banco Sc | null | 7 | null | Biceps | P1r |
| 4 | null | 8 | null | Jalones P | null | 25 | null | Hombro | P3r |
| 5 | null | null | 4 | Patadas T | null | null | 20 | Triceps | P2t |
| 6 | null | 4 | null | Jalones P | null | 15 | null | Hombro | P3r |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |

Ilustración 124 - Inicio de la tabla BD Android

La imagen anterior muestra un ejemplo, donde mostramos que la fila a eliminar es la que tiene el (id = 3) y su elemento siguiente el sería el (id = 4).

Lo siguiente que se hace es actualizar la fila 3 con todo el contenido de la fila 4 del entrenamiento seleccionado. De esta manera destruimos toda la información de la fila seleccionada, y este proceso lo repetimos con un bucle que recorre todas las filas hasta llegar al final de la tabla, que para este ejemplo son 6 filas, esto lo hacemos incrementando los contadores de actual y siguiente a uno por cada iteración. Ejemplo:


| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | null | null | Prensa | 30 | null | null | Pierna | P1m |
| 2 | 3 | null | null | Elevación | 12 | null | null | Hombro | P1m |
| 3 | null | 8 | null | Jalones F | null | 25 | null | Hombro | P3r |
| 4 | null | 8 | null | Jalones F | null | 25 | null | Hombro | P3r |
| 5 | null | null | 4 | Patadas T | null | null | 20 | Triceps | P2t |
| 6 | null | 4 | null | Jalones F | null | 15 | null | Hombro | P3r |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |

Ilustración 125 - Primera ejecución del algoritmo


| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | null | null | Prensa | 30 | null | null | Pierna | P1m |
| 2 | 3 | null | null | Elevación | 12 | null | null | Hombro | P1m |
| 3 | null | 8 | null | Jalones F | null | 25 | null | Hombro | P3r |
| 4 | null | null | 4 | Patadas T | null | null | 20 | Triceps | P2t |
| 5 | null | 4 | null | Jalones F | null | 15 | null | Hombro | P3r |
| 6 | null | 4 | null | Jalones F | null | 15 | null | Hombro | P3r |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |

Ilustración 126 - Penúltima ejecución del algoritmo

En la imagen anterior se puede apreciar que en la primera iteración la fila 3 ha sido actualizada por el contenido de la fila 4, quedando así dos filas duplicadas, pero el algoritmo no termina allí, ya que tiene que recorrer todas las filas de la tabla, es por eso que la imagen siguiente muestra cómo ha llegado a su final, pero con la última fila duplicada.

Una vez terminado el bucle, lo que hacemos es eliminar la última fila con la función **eliminarUnaFila**() y restaurar el contador hasta el elemento siguiente menos uno. De esta manera aseguramos que los punteros y los elementos de la BD estén en orden secuencial.

Es por tanto que la siguiente imagen muestra una tabla actualizada y consistente, que ha pasado de tener seis elementos a tener cinco. Y con el contador *AUTO_INCREMENTE* restaurado para que empiece a contar desde el seis.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | null | null | Prensa | 30 | null | null | Pierna | P1m |
| 2 | 3 | null | null | Elevación | 12 | null | null | Hombro | P1m |
| 3 | null | 8 | null | Jalones F | null | 25 | null | Hombro | P3r |
| 4 | null | null | 4 | Patadas T | null | null | 20 | Triceps | P2t |
| 5 | null | 4 | null | Jalones F | null | 15 | null | Hombro | P3r |

Ilustración 127 - Finalización del algoritmo

**Función getPesosByMusc**

```
public int[] getPesosByMusc(String grupo, String entrenamiento) {

    db = this.getWritableDatabase();
    Cursor mCursor = db.query("entrenamientos", new String[] {
            entrenamiento, GRUPO }, GRUPO + " = " + "'" + grupo + "'",
            null, null, null, KEY_ID, null);

    // Ponemos el cursor al inicio del todo (Inicializar o Resetar)
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    cerrar();

    int[] lista = null;
    if (mCursor.getCount() <= 0) {
        lista = new int[1];
        lista[0] = 0;
    } else {
        // Vamos a guardar en un array los grupos consultados
        lista = new int[mCursor.getCount()]; // Num filas afectadas por la
                                             // consulta
        int i = 1;

        do {

            lista[i - 1] = mCursor.getInt(0); // Pesos por grupo, array
                                              // empieza desde cero
            i++;

            mCursor.moveToNext();

        } while (i <= mCursor.getCount());
    }

    return lista;
}
```

Ilustración 128 - Método getPesosByMusc

Esta función lo que pretende es hacer una consulta y devolver en un Array de enteros todos los pesos existentes por cada grupo muscular consultado. Si el grupo examinado no tiene elementos, éste devolverá Cero en un Array de una posición, de lo contrario creara el Array con las dimensiones que haga falta y con sus elementos correspondientes, ordenado por la clave primaria.

#### *4.2.6.3.4. Funciones Sobre la Base de Datos SQLite Tabla Formas*

Los métodos para esta función son muy parecidos a los métodos que se hacen en la tabla de "entrenamientos", ya que comparten la mayoría de los atributos, como los de las series, el ejercicio, el peso y el identificador de clave primaria. Es por eso que no voy a entrar mucho en detalle con estas operaciones.

Lo único que no comparten son las pestañas de "fechas" y "pestaña", que en esta tabla no las necesitamos, ya que la funcionalidad de está es guardar información de alguna manera estática, los usuarios no interactúan con esta tabla de manera directa, de lo contrario que sucede con la tabla entrenamientos.

La tabla formas solo almacena la información que yo quiero, como la de cantidad de ejercicios que quiero mostrar, con sus series correspondientes y un peso nulo.

Además los métodos que he creado para esta función son muy similares a los de la tabla entrenamientos, solo que varían un poco en su nombre y en las columnas que se ven afectadas por las consultas o inserciones.

Ejemplo: el código siguiente muestras dos funciones sobre la tabla de formas, donde una inserta y la otra elimina filas. Nótese que sus nombres siempre hacen referencia a la tabla de formas.

```
public long insertarEnFormas(String seriem, String serier, String seriet,
        String ejercicio, String PesoNulo, String grupo) {
    // db = this.getWritableDatabase();
    abrir(); // Para poder insertar en la BD

    // Reseamos las filas para hacer pruebas desde cero en (tabla formas)
    db.execSQL("delete from sqlite_sequence where name='formas';");

    ContentValues cv = new ContentValues();

    // Nombre de las columnas a introducir los valores
    cv.put(SERIEM, seriem.trim()); // Con trim(), quitamos los posibles
                                   // espacios en blanco al Inicio o Final
    cv.put(SERIER, serier.trim());
    cv.put(SERIET, seriet.trim());
    cv.put(EJERCICIO, ejercicio.trim());
    cv.put("PesoNulo", PesoNulo.trim());
    cv.put(GRUPO, grupo.trim());

    return db.insert("formas", null, cv);
}

public boolean eliminarUnaFilaEnFormas(long rowId) {
    // Eliminar una fila en concreto
    db = this.getWritableDatabase();
    boolean salida = db.delete("formas", KEY_ID + "=" + rowId, null) > 0;
    cerrar();
    return salida;
}
```

Ilustración 129 - Métodos sobre la tabla formas en la BD Android

#### *4.2.6.3.5. Funciones Sobre la Base de Datos SQLite Tabla Medias*

Los métodos de esta función son muy importantes, ya que de estos datos es en donde almaceno las medias producidas por cada ejercicio con su correspondiente peso introducido. Estos datos se almacenan de manera automática cada vez que un usuario crea, modifica o elimina un entrenamiento. Además, esta tabla me sirve para mostrar información de manera gráfica de la evolución del progreso del entrenamiento en el gym.

**Función insertarFvsP**

Este método se encarga de ir almacenando las medias de los pesos por cada grupo muscular (Seis en total), con la fecha correspondiente con la creación o modificación.

```
public long insertarFvsP(String fecha, String m1, String m2, String m3,
        String m4, String m5, String m6) {

    abrir(); // Para poder insertar en la BD

    // Reseamos las filas para hacer pruebas desde cero en (tabla formas)
    // db.execSQL("delete from sqlite_sequence where name='formas';");

    ContentValues cv = new ContentValues();

    // Nombre de las columnas a introducir los valores
    cv.put("fecha", fecha.trim()); // Con trim(), quitamos los posibles
                                   // espacios en blanco al Inicio o Final
    cv.put("m1", m1.trim());
    cv.put("m2", m2.trim());
    cv.put("m3", m3.trim());
    cv.put("m4", m4.trim());
    cv.put("m5", m5.trim());
    cv.put("m6", m6.trim());

    return db.insert("medias", null, cv);
}
```

Ilustración 130 - Método insertarFvsP

### Función MediasPesos

```
public int[] MediasPesos(String media) {

    db = this.getWritableDatabase();
    Cursor mCursor = db.query(true, "medias", new String[] { "fecha", "m1",
            "m2", "m3", "m4", "m5", "m6" }, null, null, null, null, null,
            null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }

    cerrar();

    int indice = 0;
    int[] lista = null;
    if (mCursor.getCount() <= 0) { // Si no hay consultas

        lista = new int[1];
        lista[0] = 0;

    } else {

        lista = new int[mCursor.getCount()];

        if (media == "m1")
            indice = 1;
        if (media == "m2")
            indice = 2;
        if (media == "m3")
            indice = 3;
        if (media == "m4")
            indice = 4;
        if (media == "m5")
            indice = 5;
        if (media == "m6")
            indice = 6;

        for (int i = 0; i < mCursor.getCount(); i++) {
            lista[i] = mCursor.getInt(indice);
            mCursor.moveToNext();
        }
    }

    return lista;
}
```

Ilustración 131 - Método MediasPesos

Este método se encarga de hacer la consulta sobre la tabla "medias" y devolverme en un Array de enteros todas las filas existentes por cada grupo muscular, esta función es muy útil, ya que solo me falta pasarle el índice (Nombre Columna) y este me devolverá sus correspondientes valores.

### *4.2.7. Actividad Selección Tipo de Entrenamiento*

La actividad de entrenamientos fue construida por dos actividades distintas (Personalizado y Predeterminado), cada una con una programación y funcionalidad diferente, unidas a través de una tercera, que lo que hace es compactarlas en una, a través de dos pestañas. Esto ayuda a que el dinamismo y la usabilidad sean más intuitivos a la hora de usar en aplicaciones móviles.

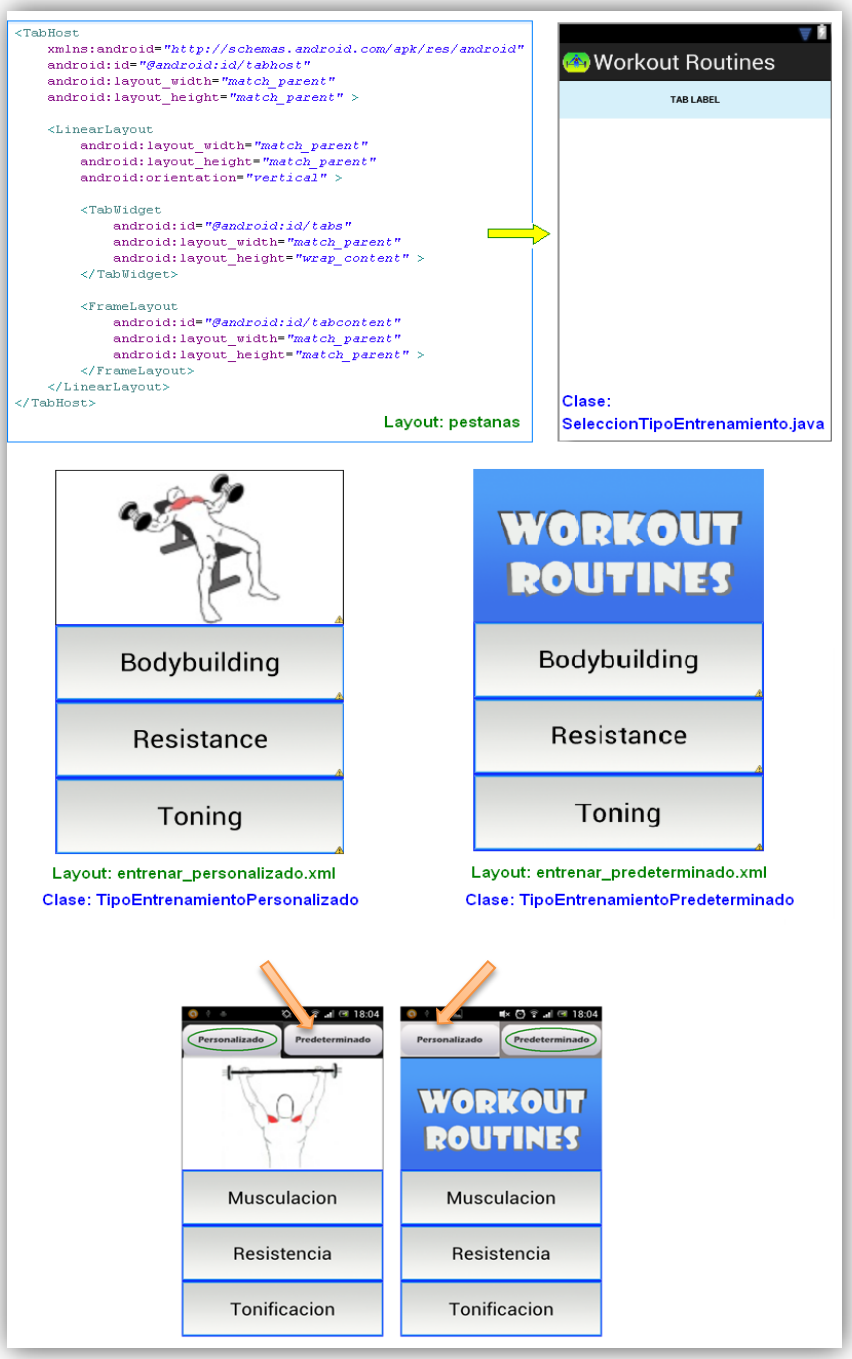



Ilustración 132 - Funcionalidad de las pestañas con TabHost

La imagen anterior muestra la unión dinámica que hace la clase **SelecciónTipoEntrenamiento.java**, a las otras dos actividades, dando la sensación de que solo pertenece a una única Actividad.

Esto se hace posible a través de la clase ***TabActivity***, aunque ahora está obsoleta para las nuevas versiones de *Android*, sirve muy bien para nuestro desarrollo de esta aplicación. Para poder trabajar con ella lo único que hay que hacer es extender de esta clase y llamar y construir los métodos correspondientes para cargar las pestañas que uno quiera con las clases que a uno le interese mostrar o añadir.

La siguiente imagen muestra el código de cómo extendemos de la clase ***TabActivity*** y como creamos un objeto de esta clase a través de ***getTabHost***(). Se puede apreciar en la imagen que nos marca como obsoleto, ya que para las nuevas versiones esta clase ya no se usa con *Java*, sino con XML.

```
import android.app.TabActivity;

@SuppressWarnings("deprecation")
public class SeleccionTipoEntrenamiento extends TabActivity{

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Quitar el titulo de la aplicacion
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Poner siempre Vertical
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        // Layaut que contiene las pestañas
        setContentView(R.layout.pestanas);

        final TabHost tabHost = getTabHost();
        getResources(); //Obtenemos los recursos
        Intent intent; // Intent que se utilizara para abrir cada pestana
```

Ilustración 133 - Código para la creación de pestañas

Luego de esto mostraremos como se crearon las pestañas, ya que estas dependen del número elementos que vallamos agregando a los ***tabHost***. Para esto hemos creado un objeto de la clase ***Intent*** el cual nos sirve para hacer el cambio entre actividades, ya que una será para la clase personalizada y la otra para la predeterminada.

También, a los ***tabHost*** se agregó unos botones gráficos, el cual estarán en inglés o en español, dependiendo de cómo esté configurado el móvil, con un simple ***if***, sabremos cual opción poner.

```
// Creamos las pestañas para cada Clase
// Dependiendo del orden, primero sera la Clase *** Personalizada ***
intent = new Intent().setClass(this, TipoEntrenamientoPersonalizado.class);
if(idioma.equals("español (España)")){
    tabHost.addTab(
            tabHost.newTabSpec("Personalizado").setIndicator(
                    "",getResources().getDrawable(R.drawable.personalizado)
            ).setContent(intent)
    );
}
else{
    tabHost.addTab(
            tabHost.newTabSpec("Personalizado").setIndicator(
                    "",getResources().getDrawable(R.drawable.customize)
            ).setContent(intent)
    );
}

// Segunda pestaña sera la CLase *** Predeterminada ***
intent = new Intent().setClass(this, TipoEntrenamientoPredeterminado.class);
if(idioma.equals("español (España)")){
    tabHost.addTab(
            tabHost.newTabSpec("Predeterminado").setIndicator(
                    "",getResources().getDrawable(R.drawable.predeterminado)
            ).setContent(intent)
    );
}
else{
    tabHost.addTab(
            tabHost.newTabSpec("Predeterminado").setIndicator(
                    "",getResources().getDrawable(R.drawable.defaul)
            ).setContent(intent)
    );
}
```

Ilustración 134 - Creación de pestañas mediante código Java

Por último, a las pestañas se les agregó un color de fondo, la cual ayuda a saber si están activas o no, esto ayuda a diferenciar e identificar en cada momento donde el usuario se encuentra. El siguiente código muestra cómo se hace.

```
//Cambiamos el fondo a los tabs, ademas de cambiar el color de fondo cada vez que se pulsa
tabHost.setOnTabChangedListener(new OnTabChangeListener(){
    public void onTabChanged(String tabId) {
        // TODO Auto-generated method stub
         for(int i=0;i<tabHost.getTabWidget().getChildCount();i++)
            {
               //unselected
               tabHost.getTabWidget().getChildAt(i).setBackgroundColor(Color.parseColor("#d9d5d5"));
            }
            // selected
            tabHost.getTabWidget().getChildAt(tabHost.getCurrentTab()).setBackgroundColor(Color.parseColor("#86807f"));
    }
});
```

Ilustración 135 - Agregar fondo a las pestañas

#### *4.2.7.1. Actividad Personalizado*

Para continuar con el desarrollo de la Actividad **Entrenamientos**, voy a explicar sus partes más representativas. Como ya se ha hablado en el apartado anterior, esta Actividad está incluida dentro de otra que hace de función de acoplamiento, pero no deja de ser una actividad como tal.

Su función principal es ayudar a los usuarios a separar sus rutinas en tres tipos, **Musculación**, **Resistencia** y **Tonificación**, incluyendo un *banner* con imágenes donde muestro como se realizan algunos ejercicios. Esto se hizo con la idea de separar y especificar cada entrenamiento, ya que es muy común encontrarse personas que usen más de dos rutinas distintas.

El diseño de este es muy similar al del menú Main (principal), explicado anteriormente, ya que el diseño de los botones es igual, incluyendo las imágenes (*Banner*). Se hizo un reparto equitativo de los pesos de los Layouts para que se ajusten a los distintos tamaños de móviles, pero dando un grado de peso más al banner de las imágenes.

Con los botones hacemos referencia a ellos a través de su identificador, después los instanciamos con código java haciendo un *Click Listener*, para capturar su valor cuando son pulsados y luego poder enviarlos a otra Actividad.

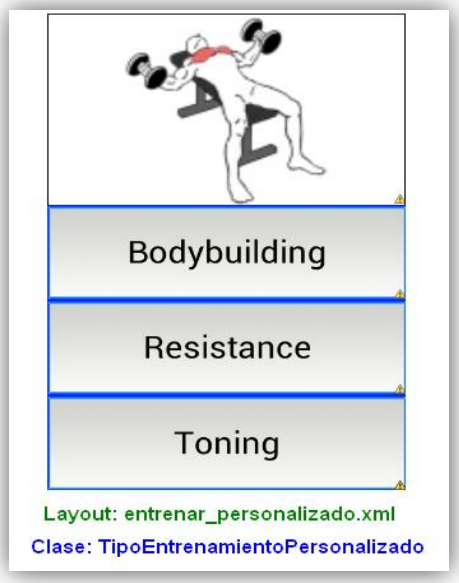



Ilustración 136 - Diseño de la ventana Entrenamientos

#### *4.2.7.1.1. Construcción y Diseño de Banners*

Para la creación de este *banner*, solo necesitamos un par de imágenes que queremos que se muestren, después definimos la velocidad de aparición en un fichero XML y ejecutarlo a través de código *Java*.

Pasos para crear un Banner.

1. Crear una carpeta "***drawable***" dentro de "***res***" ya que por defecto *Android* no la crea. En esta carpeta introduciremos las imágenes que necesitamos que se muestren en el *banner*.

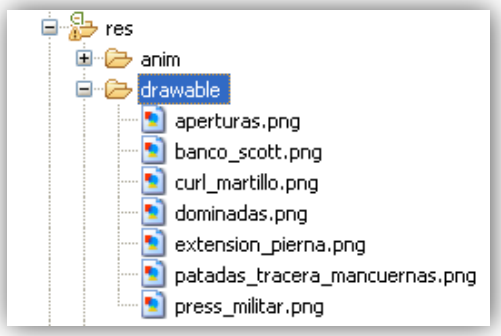



Ilustración 137 - Directorio donde almacenamos las imágenes para su animación

2. Crear un XML con la siguiente configuración, este fichero tiene que estar dentro de la carpeta "***anim***" creada anteriormente, ya que por defecto *Android* no la crea. En la imagen siguiente, se ve que hay más de un archivo xml, estos sirven para configurar una animación a la hora de pulsar un botón o cambiar una imagen por otra.

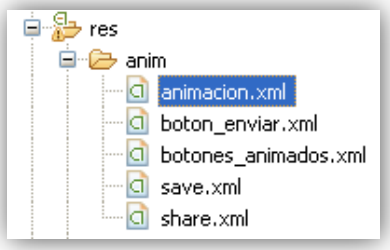



Ilustración 138 - Fichero responsable del Banner de imágenes

El siguiente paso es crear una configuración para este fichero XML. A cada imagen le damos un tiempo de aparición, el cual se estableció a 450 ms, para que el cambio de una por otra sea de forma pausada. Nuestro fichero XML se llama "**animación.xml**" y tiene el siguiente contenido.

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false"
    >
    <item
        android:drawable="@drawable/aperturas"
        android:duration="450"/>
    <item
        android:drawable="@drawable/banco_scott"
        android:duration="450"/>
    <item
        android:drawable="@drawable/curl_martillo"
        android:duration="450"/>
    <item
        android:drawable="@drawable/dominadas"
        android:duration="450"/>
    <item
        android:drawable="@drawable/extension_pierna"
        android:duration="450"/>
    <item
        android:drawable="@drawable/patadas_tracera_mancuernas"
        android:duration="450"/>
    <item
        android:drawable="@drawable/press_militar"
        android:duration="450"/>
</animation-list>
```

Ilustración 139 - Código XML para la creación del Banner

3. Ahora lo que hacemos es crear y configurar un objeto de tipo imagen, el cual le damos un identificador (nombre) "**id=banner**", el cual nos sirve para ubicarlo a través de código *Java*. La última línea de código, donde pone (**SRC = animación.xml**), sirve para hacer referencia a nuestra animación de imágenes.

```
<ImageView
    android:id="@+id/banner"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="2"
    android:src="@anim/animacion" />
```

Ilustración 140 - Creación del Widget responsable de añadir imágenes al Banner

4. En este último paso, con código Java, lo que hacemos crear un Objeto de Tipo ***ImageView***, el cual hace referencia a nuestro Banner de imágenes. Lo siguiente es hacer que las imágenes se alternen, esto se hace con la Clase

***AnimatioDrawable***, que nos permite hacer la animación en tiempo de ejecución, dando la sensación de *Banner* aleatorio o secuencial. Esto lo hace a través de su método ***start***(), que crea un hilo para poder ejecutarse independientemente de los demás procesos que se estén ejecutando dentro de la *App*.

```
/*Animacion o Banner de Imagenes.PNG */
ImageView myAnimation = (ImageView)findViewById(R.id.banner);
final AnimationDrawable myAnimationDrawable = (AnimationDrawable)myAnimation.getDrawable();
myAnimationDrawable.start();
```

Ilustración 141 - Ejecución Banner mediante código Java

#### *4.2.7.1.2. Botones Musculación, Resistencia y Tonificación*

Estos botones están programados de manera similar al de los botones de la Actividad Principal. La única diferencia, está en que se configura una línea de código que permite diferenciar el botón del cual ha sido pulsado, una vez que cambiamos de actividad, esto nos viene muy bien, ya los tres botones usan una misma actividad, pero dando la sensación de que una es diferente de la otra. En un principio se pensaba hacer todo esto de manera independiente (una actividad independiente por cada botón), pero esto nos costaría mucho trabajo, ocuparía mucho espacio de memoria, ya que tendríamos muchas clases Java, con sus correspondientes ficheros XML por cada una de ellas.

Para mejorar lo anterior, se decidió crear una plantilla llamada PlantillaEntrenarGrupoMuscular.Java, que usa como interfaz contenedor un fichero XML, llamado pestañas.xml, comentado anteriormente para crear pestañas de manera dinámica. Esta Clase es única para configurar estos tres botones (Musculación, resistencia y Tonificación), ya sea que vengan de la pestaña de Predeterminada o Personalizada. Por lo que al final, lo que hacemos es una consulta a la BD para obtener la información necesaria dependiendo de la consulta a hacer por cada actividad seleccionada.

Para pasar datos de una actividad a otra, lo que hacemos es por cada ***Intent*** (Clase que sirve para navegar entre actividades), le pasamos un parámetro, que puede ser ***String***, ***Integer***, ***Boolean*** o ***Double***. Esto lo hacemos a través del método ***putExtra***(***String Name***, ***Tipo Valor***) (Propia de la Clase *Intent*) que sirve para llevar un dato de una actividad a otra y que luego podremos recoger en la otra actividad a través de la clase ***Bundle***.

Esta plantilla la explicare más detalladamente en los apartados siguientes, después de explicar las Actividades de Personalizado y Predeterminado.

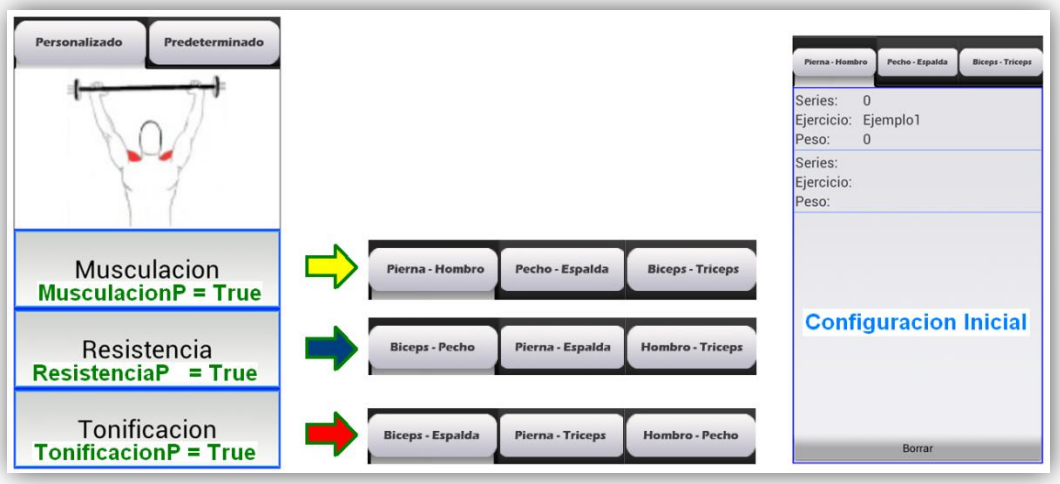



Ilustración 142 - Funcionalidad de los botones de Entrenamientos

La imagen anterior muestra cómo enviamos un dato de tipo Booleano a la clase **PlatillaEntrenarGrupoMuscular.Java**, donde luego recogeremos ese dato a través del nombre (MusculacionP o ResistenciaP o TonificacionP) que le hemos dado, usando la clase **Bundle**.

Se aprecia que las pestañas cambian de nombre entre los tres tipos de entrenamientos seleccionados, es así, porque necesitamos tener variedad de ejercicios por cada entrenamiento. Más que nada por variar a la hora de entrenar los grupos musculares, ya que es muy diferente el esfuerzo físico que se hace con uno u otro.

```
// Vamos al entrenamiento Musculacion, creamos un click listener para una imagen
Button b1 = (Button) findViewById(R.id.muscup);
b1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i1 = new Intent(getApplicationContext(), PlantillaEntrenarGrupoMuscular.class);
        i1.putExtra("MusculacionP", true); // Enviamos flags a la siguiente actividad
        i1.setAction(Intent.ACTION_VIEW);
        i1.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(i1);
    }
});
```

Ilustración 143 - ejecutar el botón Musculación

El código anterior refleja la configuración para el botón de Musculación, pero esta es igual para los botones de Resistencia y Tonificación, solo que con un nombre distinto y haciendo referencia al botón que queremos usar, ResistenciaP y TonificacionP respectivamente.

Cuando estos valores se recuperan en la siguiente Clase, solo se captura uno, el que se haya pulsado que estará a true, porque es el valor que quiero enviar, por contra parte los demás estarán a false.

Otra peculiaridad de esta programación al navegar con Intent, es que limpiamos la Pila en la cual se almacena esta actividad, esto es para que cuando damos al botón de retroceso no tengamos que volver por las mismas ventanas (rutas) por las cuales ya hemos pasado. Esto se hace limpiando a través del siguiente código.

```
i2.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
```

Esto se debe a que en Android el sistema gestiona las diferentes actividades o aplicaciones usando el concepto de Pila, de forma que, en un momento determinado, sólo la Actividad que se encuentra en la cima de esta pila, se está ejecutando (Se muestra), mientras que el resto están en Stand-by y la memoria que ocupan puede ser asignada a otros procesos.

Así, cuando una actividad se encuentra en primer plano y otra ocupa la cima de dicha pila, aquella pasa a un segundo plano y es susceptible de ser eliminada del sistema, en lo que se denomina "ciclo de vida de una actividad" y que se expresa gráficamente en la siguiente imagen.

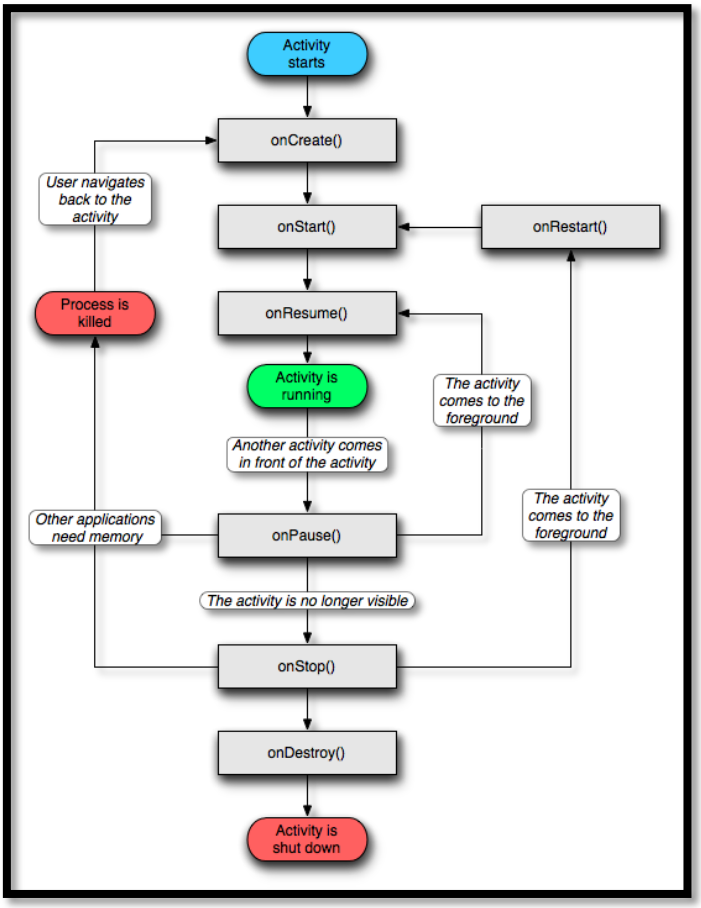



Ilustración 144 - Ciclo de vida de una Actividad

Para el siguiente ejemplo (usando ***Flag_Activity_Clear_Top***) de cómo trabajar con la Pila de *Android*, me viene muy bien representar este patrón que se propone, ya

que es el que usa mi aplicación, solo que las actividades las designaremos con **A**, **B**, **C y D**, tal como se muestra en la siguiente imagen.

Uso esta representación, ya que a la hora de trabajar con la pila y las actividades, este el proceso (camino) más largo que uso, por tanto es muy conveniente para esta explicación.


Ilustración 145 - Ejecución más extensa de las Actividades

A continuación, el diagrama siguiente muestra el camino lógico de las actividades, donde "A lanza B", "B lanza C", y "C lanza D", quedando la pila resultante **D, C, B, A**, siendo la Actividad "**D**" la Cima de la Pila. Ahora bien, al usar ***Clear_Top***, lo que hacemos es reconstruir el camino natural que seguimos a la hora de navegar entre actividades, de tal forma que cuando regresemos al inicio del todo, sea el mismo camino por el cual hemos ido. Ya que irá desapilando.

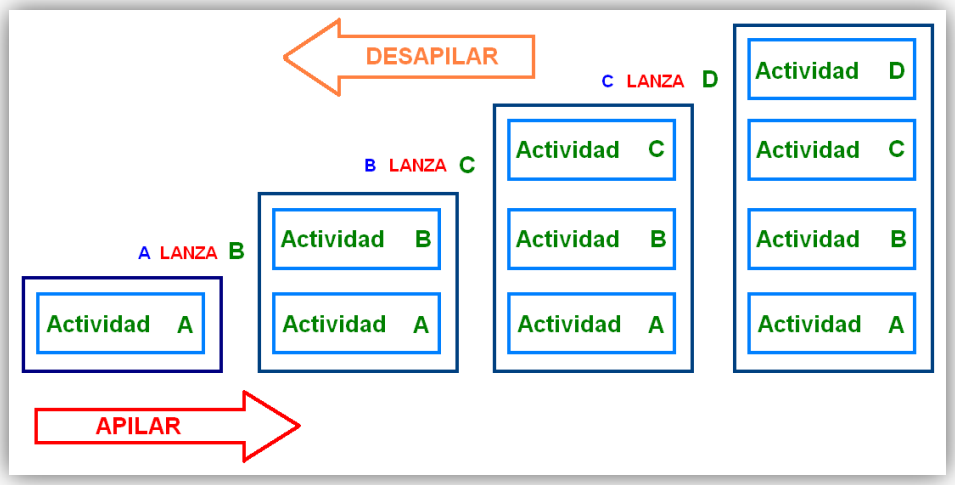



Ilustración 146 - Estado de la Pila con Actividades de la App

Un problema que surgió a la hora de usar actividades y de ir intentando desapilarlas, es que se usó el método ***finish***() de la Clase ***Intent***, pues bien, este método no permite apilar. Lo que hace, es que cada vez que se lanza una nueva

Actividad, la anterior que la ha lanzado se desapila, de tal forma que si quiero volver por el camino que he recorrido, me encontraré que saldré de la aplicación bruscamente, ya que no tendré camino por el cual volver. La siguiente imagen muestra el proceso.

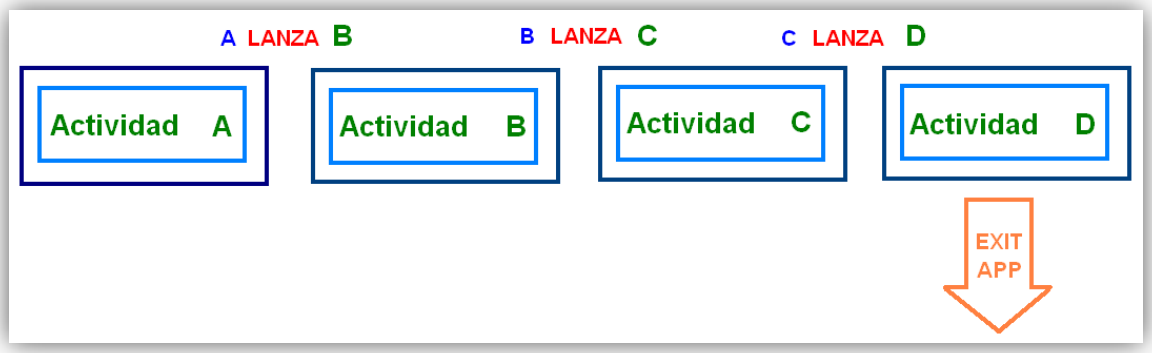



Ilustración 147 - Estado de la Pila usando el método finish

### *4.2.7.2. Actividad Predeterminado*

Esta actividad es muy similar a la Actividad Personalizado, solo que es un poco más sencilla, ya que solo se usa para mostrar información detallada de los posibles entrenamientos. La idea consiste en dejar una serie de rutinas predeterminadas, donde el usuario pueda seguir como ejemplo para poder crear las suyas propias.


Ilustración 148 - Diseño de la Actividad Predeterminado

El diseño es igual que la Actividad Personalizada, solo que en vez de usar un *banner* de imágenes, esta usa una imagen estática, donde ponemos de nuevo el nombre de la aplicación, esto se hace para que los usuarios recuerden en cada momento como se llama la app que están usando y así puedan recordarla fácilmente.

Los botones Musculación, Resistencia y Tonificación tienen el mismo efecto que los botones de la Clase Personalizada, ya que van a la misma plantilla (PlantillaEntrenarGrupoMuscular.Java) para visualizar los datos. Solo que esta vez se hace una consulta diferente, en esta mostramos una combinación de ejercicios ya diseñados y estructurados para que los usuarios puedan planificar o diseñar una serie de ejercicios musculare, estas tienen el peso con valor nulo, pero las series están puestas de tal manera que no produzca fática con otros ejercicios.

Recordar que la plantilla (PlantillaEntrenarGrupoMuscular.Java) que utilizo la explicaré en el apartado siguiente, ya que esta es genérica para muchas actividades que dependen de ella para mostrar información.

### *4.2.7.3. Actividad PlantillaEntrenarGrupoMuscular*

Esta clase es la encargada de mostrar la información agrupada por pares de grupos musculares, ya sea para los ejercicios que son consultados en la tabla de "**entrenamientos**" (Dinámica) o en la tabla de "formas" (Estática).

La elección se hace, seleccionando de donde se viene, para ello, utilizo la Clase ***Bundle***, recojo los valores que son de tipo booleano, que previamente había enviado, de esta manera puedo saber que camino he elegido. La siguiente imagen muestra como hago para recoger estos datos.

```
// Cuando hacemos la Consulta para mostrar los datos,
// pasamos los datos ya introducidos cuando hacemos click
// Datos (flags) recogidos de la clase TipoEntrenaientojava
Bundle bundle = getIntent().getExtras();
//Entrenamientos Predeterminados
Boolean m = bundle.getBoolean("bMusculacion");
Boolean r = bundle.getBoolean("bResistencia");
Boolean t = bundle.getBoolean("bTonificacion");
// Entrenamientos Personalizados
Boolean pm = bundle.getBoolean("MusculacionP");
Boolean pr = bundle.getBoolean("ResistenciaP");
Boolean pt = bundle.getBoolean("TonificacionP");
```

Ilustración 149 - Captura de mensajes con la clase Bundle

Una vez capturado estos datos (***flags***), podemos hacer una comparación para saber qué clase le corresponde mostrar información.

La Clase **ListaEntrenamientoNormal.java** tiene los ejercicios estáticos que muestro como ejemplo a seguir, esta Clase se encarga de hacer las consultas pertinentes en la tabla "**formas**".

Mientras, la clase **VerEnFilas.java** se encarga de mostrar todos los entrenamientos, ya que esta, se encarga de interactuar con la tabla de "**entrenamientos**" de nuestra *BD*,

En la siguiente imagen hago la selección, de cual Clase elegir para el caso que le corresponda. Por ejemplo, las variables Booleanas (**m, r, t**) representan los botones estáticos (**Musculación, Resistencia, Tonificación**) respectivamente, si alguno de estos valores está a ***True***, es porque venimos de la **Pestaña Predeterminado**, y es por eso, que luego asignamos a la variable "clase" de tipo ***Class***, su clase Java correspondiente a **ListaEntrenamientoNormal.class**.

Por el contrario, si alguna de las variables (**pm, pr, pt**) está a ***True***, es porque venimos de la **Pestaña Personalizado**.

```
//Elegimos la Clase a Mostrar, depende de si sera dinamica o estatica
Class<?> clase = null;
if(m | r | t){
    // Ejercicios Predeterminados o Estaticos
    clase = ListaEntrenamientoNormal.class;
}
if(pm | pr | pt){
    // Ejercicios Dinamicos o Personalizados
    clase = VerEnFilas.class;
}
```

Ilustración 150 - Determinar la Clase a usar, Actividad Predeterminada o Personalizada

Ahora voy a mostrar cómo se crea una pestaña, recordar que utilizo una plantilla.

La siguiente imagen muestra el código de como creo un grupo de tres pestañas para los ejercicios de musculación. Asignándoles una foto referente de lo que pretendo informar.

```
// Creamos las pestañas a partir de cada entrenamiento elegido
if (m | pm)
{
    im1 = new Intent().setClass(this, clase);
    im1.putExtra("Im1", true); //Llevar al Bundle
    tabHost.addTab(tabHost.newTabSpec("Pestaña1").
            setIndicator("", getResources().getDrawable(pm1)).setContent(im1));
    tabHost.getTabWidget().getChildAt(0); // que sea la pestaña 1

    im2 = new Intent().setClass(this, clase);
    im2.putExtra("Im2", true); //Llevar al Bundle
    tabHost.addTab(tabHost.newTabSpec("Pestaña2").
            setIndicator("", getResources().getDrawable(pm2)).setContent(im2));
    tabHost.getTabWidget().getChildAt(1); // que sea la pestaña 2

    im3 = new Intent().setClass(this, clase);
    im3.putExtra("Im3", true); //Llevar al Bundle
    tabHost.addTab(tabHost.newTabSpec("Pestaña3").
            setIndicator("", getResources().getDrawable(pm3)).setContent(im3));
    tabHost.getTabWidget().getChildAt(2); // que sea la pestaña 3
}
if(r | pr)
{
```

Ilustración 151 - Creación de pestañas mediante Java

Esta otra imagen muestra un ejemplo a seguir, donde tenemos las dos alternativas posibles que puede tomar este código, ya sea para ejercicios dinámicos o estáticos.


Ilustración 152 - Resultado final

En la imagen anterior se puede apreciar que la funcionalidad es la misma, pero con resultados diferentes. Ya que ambas clases usan el mismo xml, que es una plantilla de tipo ***ListView***. Este y en conjunto con otro xml que sirve de base para recargar la información, es capaz de mostrarme la información en lista dinámicas.

Para poder hacer esto, ambas clase deben extender de ***ListActivity***, que tiene las mismas características que una Clase ***Activity***, pero con la mejora que de que podemos crear listas dinámicas.

En el siguiente Ítem explicare mejor este aparatado. Como base uso la Actividad VerEnFilas, porque es la más representativa para nuestra aplicación.

#### *4.2.7.4. Actividad VerEnFilas*

Esta actividad es muy importante, ya que es la encargada de trabajar con las especificaciones de esta aplicación, porque aquí, es donde se visualizan los datos, que luego servirán para ser creados, modificados o eliminados,

Esta actividad trabaja con la clase **VerEnFilas.java** y con su *layout* correspondiente **lista_main.xml**, además de esto, es necesario crearse una clase adaptadora de tipo ***ArrayAdapter*** que trabaje con su *layout* correspondiente para ir agregando los datos estructurados a la Clase de tipo ***ListView***.

Para hacerse una idea general, propongo unas gráficas que representen esta funcionalidad.

La siguiente gráfica representa cual sería el efecto final que se obtiene al usar una clase de tipo ***ListView*** (***ListActivity***) junto con otra que es de tipo ***ArrayAdapter***. Una capa única, donde por cada ejercicio introducido, este me lo irá adaptando a mis requisitos en una lista, de tal manera, que da la sensación de que solo estoy usando una sola clase con un solo XML.

Cabe decir que el xml (ver_texto.xml) es el que se va adaptado a la lista (lista_main.xml), y lo hará las veces que haga falta, ya que depende del número de elementos introducidos en el ***ArrayAdapter*** y este a su vez, depende de la cantidad de filas que tengamos en nuestra tabla de entrenamientos.

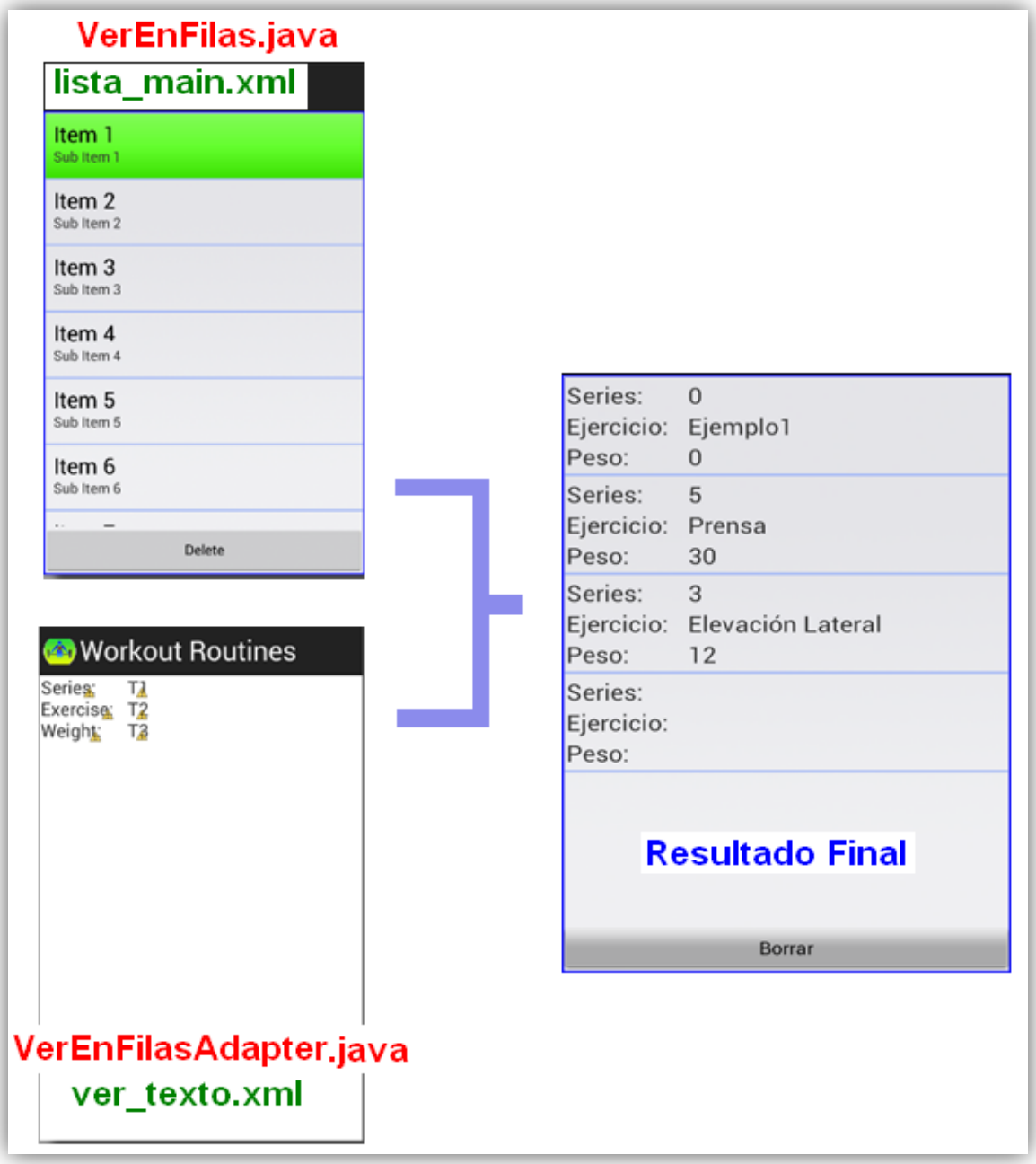



Ilustración 153 - Diseño de la Clase VerEnFilas

Aunque en la gráfica anterior no se aprecie, la clase VerEnFilas.java extiende de un ***ListActivity*** y la clase VerEnFilasAdapter.java extiende de ***ArrayAdapter***

Una característica importante de la clase **VerEnFilas**, es la opción de tener un ***onListItemClick***, este método propio de la Clase ***ListActivity***, ayuda a reconocer cuando se ha pulsado una lista y además, recoge la información de los elementos que integran la lista una vez haya detectado la pulsación.

En la siguiente imagen, muestro como se recogen los datos de la **Serie**, **Ejercicio** y **Peso**, a partir de una lista.

```
protected void onListItemClick(ListView l, View v, int position, long id) {
    String item = (String) getListAdapter().getItem(position);

    String f1 = " ", f2 = " ", f3 = " "; // filtros
    item = item.trim();

    if (item.isEmpty()) { // Si es una fila vacia, le pasaremos Datos vacios
        f1 = " ";
        f2 = " ";
        f3 = " ";
    } else {
        int n = 1;

        StringTokenizer tokens = new StringTokenizer(item);
        int max = tokens.countTokens();

        while (tokens.hasMoreTokens()) {
            if (n == 1) {
                // Sacamos las series que son el principio del String
                f1 = tokens.nextToken();
                // Toast.makeText(this, n + ") "+ tokens.countTokens()
                // +"  "+f1 , Toast.LENGTH_SHORT).show();
            } else if (n == max) {
                // Sacamos el peso, que esta al final del todo
                f3 = tokens.nextToken();
                // Toast.makeText(this, n + ") "+ tokens.countTokens()
                // +"  "+f3 , Toast.LENGTH_SHORT).show();
            } else {
                // Concateno todos las palabras que corresponden a un
                // ejercicio
                f2 = f2 + tokens.nextToken() + " ";
                // Toast.makeText(this, n + ") "+ tokens.countTokens()
                // +"  "+ f2 , Toast.LENGTH_SHORT).show();
            }
            n++;
        }
        // Filtros para hacer la consulta sobre que ID tiene estos
        // elementos
        f1 = f1.trim(); // serie
        f2 = f2.trim(); // ejercicio
        f3 = f3.trim(); // peso
    }
```

Ilustración 154 - Detectar cuando se pulsa un elemento de la lista y capturar los datos

En el código anterior, las variables "f1, f2 y f3" sirven de filtro, ya que este método me devuelve una lista y tengo que separarlos en Serie, Ejercicio y Peso respectivamente, para ello, me apoyo de la Clase ***StringTokenizer***, que me ayuda a separar el *String* en *Tokens*, al cual voy separando en las partes que corresponden con lo que yo deseo almacenar en las variables anteriormente nombradas.

Esta función no termina aquí, ya que solo es la primera parte, y quería explicar su funcionalidad. Una vez que tenga las variables con su respectivo valor, procederé hacer la consulta para ver que ID tienen en la BD. Para ello lo muestro en la siguiente imagen.

```
    /***** Consulta q me devuelva el id *****/
    SQLiteBD bd = new SQLiteBD(this);
    int ID = 0;
    if (item.isEmpty()) {
        ID = bd.getNumRows() + 1; // Sila fila esta vacia (Agrego su nuevo
                                  // ID, es el ultimo mas el siguiente id)
    } else {
        try {
            // Si hay Elementos Consulto en la BD
            if (PiernaHombro || PechoEspalda || BicepsTriceps) {
                ID = bd.verIdListener(f1, f2, f3, "musculacion");
            }
            if (BicepsPecho || PiernaEspalda || HombroTriceps) {
                ID = bd.verIdListener(f1, f2, f3, "resistencia");
            }
            if (BicepsEspalda || PiernaTriceps || HombroPecho) {
                ID = bd.verIdListener(f1, f2, f3, "tonificacion");
            }
        } catch (SQLException e) {
            Funciones.imprimir(this, "No se encontraron resultados");
        }
    }

    // Nos lleva a editar la fila seleccionada, con sus elementos
    // Los recogemos con la clase tipo Bundle
    Intent i = new Intent(VerEnFilas.this, EditarEntrenamiento.class);

    i.putExtra("id", ID); // La primera lista es 0, nuestra tabla es 1
    i.putExtra("serie", f1);
    i.putExtra("ejercicio", f2);
    i.putExtra("peso", f3);

    // Hacemos el recorrido de la pestaña elegida, y volvemos hacer el
    // inputExtra.
    // Hasta llegar a la clase EditarEntrenamiento.java, alli finaliza
    i.putExtra("Im1", PiernaHombro);
    i.putExtra("Im2", PechoEspalda);
    i.putExtra("Im3", BicepsTriceps);

    i.putExtra("Ir1", BicepsPecho);
    i.putExtra("Ir2", PiernaEspalda);
    i.putExtra("Ir3", HombroTriceps);

    i.putExtra("It1", BicepsEspalda);
    i.putExtra("It2", PiernaTriceps);
    i.putExtra("It3", HombroPecho);

    startActivity(i);
} // Fin ClickListener
```

Ilustración 155 - Algoritmo que devuelve el ID a partir de una lista seleccionada

La función principal de esta segunda parte es consultar el ID en la BD, que corresponde a cada fila pulsada, para ello disponemos de una función que he creado que se llama **verIDListener**. Esta función hace una consulta a la base de datos con

estos parámetros (Serie, Ejercicio, Peso, Tipo Entrenamiento) y me devuelve un entero, que va desde **0** hasta **N** elementos, donde Cero significa que no hay nada.

Una vez que conozca cuál es el ID, lo siguiente que procederé a hacer, es ir a otra Actividad, encargada de Crear, Eliminar o Modificar sobre la BD entrenamientos.

Pero antes de ir a la siguiente Actividad, debo enviar los siguientes parámetros, ID, Serie, Ejercicio y Peso a través de un ***putExtra***(), que luego será recogido por un objeto de la clase de tipo Bundle para su correspondiente análisis y tratamiento.

#### *4.2.7.5. Actividad EditarEntrenamiento*

Esta actividad es la encargada de **Crear**, **Modificar** o **Eliminar** los entrenamientos de los grupos musculares que son dinámicos, ya que trabajan directamente con la tabla de entrenamientos.

Su función reconocer los elementos de la lista que se pulsado anteriormente en la Actividad *ListView* y los trata de tal manera que si los elementos están vacíos y el ID es Cero (no existe), proceder a crear una nueva lista (Entrenamiento), de lo contrario, la acción es modificar los elementos recogidos por los que el usuario desea cambiar o eliminar si así lo desea.

La clase que trabaja esta funcionalidad se llama **EditarEntrenamiento.java**, que utiliza como layout el xml editar_entrenamietno.xml, además que implementa la interfaz TextWatcher, la cual me ayuda a crear una lista opcional que se autocompleta para ayudar a los usuarios a escribir el nombre de un ejercicio.

El diseño del XML, está compuesto de varios, *TextView*, *EditText*, un *CheckBox*, un *Spinner* y dos Botones para construir, crear y organizar correctamente esta actividad.

En la siguiente imagen, muestro como he diseñado este XML, se puede apreciar que los botones de modificar y eliminar, ocupan el mismo espacio de trabajo con respecto al resultado final, esto se debe a que cuando la actividad EditarEntrenamiento reconoce que no existe un ejercicio, esta me desactiva el botón de modificar y el *CheckBox*, pero si a cambio reconoce que existe el ejercicio en la BD, esta me oculta el botón de guardar y me hace visible el de modificar, tal cual como muestro en el siguiente ejemplo. A continuación, muestro el proceso de ocultar Widgets.


**Ilustración 156 - Diseño de la pantalla Editar Entrenamientos**

```
// Cuando hacemos la Consulta para modificar,
// pasamos los datos ya introducidos cuando gacemos click
String serie      = bundle.getString("serie").trim();
String ejercicio = bundle.getString("ejercicio").trim();
String peso       = bundle.getString("peso").trim();
int    id         = bundle.getInt("id");

// Hacer invisible los botones, Guardar y Modificar
if(id >=1 && id <= bd.getNumRows()){
    // Si hay datos, hacemos invisible el boton de guardar
    Button b2 = (Button) findViewById(R.id.guardar);
    b2.setVisibility(1000);

}
else{
    Button b1 = (Button) findViewById(R.id.modificar);
    b1.setVisibility(1000);
    // Hacer invisible el boton check a la hora de guardar
    check.setVisibility(1000);
}
```

Ilustración 157 - Algoritmo para hacer invisible un botón u otro

La imagen anterior, muestro en código java, lo necesario para crear las opciones de Modificar (Eliminar) o Guardar un Entrenamiento nuevo, además, de cómo se ocultan o se hacen visibles los botones de modificar o guardar.

Lo primero que hay que hacer, es capturar los elementos enviados a través de la Actividad anterior (VerEnFilas) y llamarlos con la Clase *Bundle*, tal cual como enseño en la ilustración 156.

Una vez que tenemos estos datos, lo siguiente es hacer una comparación con el ID devuelto de la consulta, y sí hay datos o la fila consultada existe en nuestra BD, entonces ocultamos el botón de Guardar, de lo contrario ocultamos el botón de modificar y el CheckBox.

#### *4.2.7.5.1. Función Autocomplete*

Para hacer el autocomplete, es necesario implementar la Interfaz ***TextWatcher***, esta es la encargada de trabajar con el *Widget* XML del *layout EditText*, la cual me permite introducir texto en un formulario y de autocompletármela con la lista que le indiquemos, esta lista se la pasamos dependiendo de donde se encuentre el usuario, es por eso que antes hago una consulta a la BD, en la tabla de "**Formas**", para obtener la lista de los grupos musculares que pertenecen al entrenamiento y grupo muscular consultado. De esta manera, cuando un usuario escribe un entrenamiento, mostramos una lista parecida a lo que está escribiendo, algo parecido es cuando se desea consultar en *Google*, que te aconseja autocompletar para ayudarte a reducir la escritura y ganar tiempo.

Las dos siguientes imágenes muestra como se hizo el código correspondiente al *autocomplete* y el resultado obtenido tras su ejecución y comprobación.

```
// Ejercicios de la tabla formas
String [] items = null;
if(PiernaHombro){
    items = bd.listaEjerciciosPorGrupos("Pierna","Hombro");
}
if(PechoEspalda){
    items = bd.listaEjerciciosPorGrupos("Pecho","Espalda");
}
if(BicepsTriceps){
    items = bd.listaEjerciciosPorGrupos("Biceps","Triceps");
}
if(BicepsPecho){
    items = bd.listaEjerciciosPorGrupos("Biceps","Pecho");
}
if(PiernaEspalda){
    items = bd.listaEjerciciosPorGrupos("Pierna","Espalda");
}
if(HombroTriceps){
    items = bd.listaEjerciciosPorGrupos("Hombro","Triceps");
}
if(BicepsEspalda){
    items = bd.listaEjerciciosPorGrupos("Biceps","Espalda");
}
if(PiernaTriceps){
    items = bd.listaEjerciciosPorGrupos("Pierna","Triceps");
}
if(HombroPecho){
    items = bd.listaEjerciciosPorGrupos("Hombro","Pecho");
}

column1 = (EditText)findViewById(R.id.serie);
// AutoComplete
autoComplete = (AutoCompleteTextView)findViewById(R.id.ejercicio);
column3 = (EditText)findViewById(R.id.peso);
check   = (CheckBox)findViewById(R.id.check);

// Agregamos un listener y el responsable de adaptar
// la lista para el Auto-Completado
autoComplete.addTextChangedListener(this);
autoComplete.setAdapter(new ArrayAdapter<String>(
        this, android.R.layout.simple_dropdown_item_1line, items));
```

Ilustración 158 – Algoritmo de opciones para la clase AutoComplete

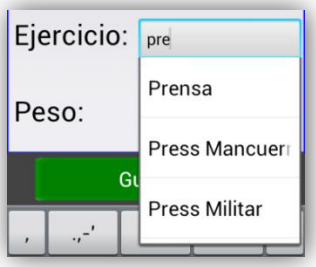



Ilustración 159 - Resultado de la Ejecución de la Clase Autocomplete

#### *4.2.7.5.2. Función Spinner*

```
/********* Spinner ********/
// Cargamos el Spinner, con las partes del cuerpo a trabajar
// Diferenciado por cada par de grupo muscular
String[] gSpinner = gruposSpiner(PiernaHombro, PechoEspalda, BicepsTriceps,
                                 BicepsPecho, PiernaEspalda, HombroTriceps,
                                 BicepsEspalda, PiernaTriceps, HombroPecho);

final Spinner sp = (Spinner) findViewById(R.id.gmuscular);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this,android.R.layout.simple_spinner_item, gSpinner);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
sp.setAdapter(adapter);

// Escuchamos al Spinner, para saber que Grupo Muscular a Seleccionado
sp.setOnItemSelectedListener(new OnItemSelectedListener() {
    public void onItemSelected(AdapterView<?> parent,
            View selectedItemView, int position, long id) {
        //Agregamos el grupo selecionado al String para luego hacer la insercion
        GrupoM = parent.getItemAtPosition(position).toString();
    }
    public void onNothingSelected(AdapterView<?> parentView) { }
});
```

Ilustración 160 - Algoritmo para la ejecución de elementos Spinner

En el código anterior, lo que pretendo hacer, es crear una lista de opciones donde los usuarios deben seleccionar únicamente un elemento (de dos posibles) del grupo muscular que desea trabajar.

Estos valores, los selecciono por pareja de donde se encuentre ubicado el usuario, así que antes, hago una consulta y obtengo que pareja muscular quiere crear o modificar, así que estos dos valores los guardo en un *Array* temporal tipo *String* y se los paso al adaptador de la Clase *Spinner*, que es la encarga de gestionar todo esto.

Una vez creado el *Spinner*, lo siguiente es conseguir el *ClickListener*, ya que es necesario saber que opción ha pulsado el cliente, y de esta manera capturar este dato y guardarlo en la BD.

La siguiente gráfica, muestra el proceso que se sigue a la hora de trabajar con Spinners. Primero se rellenan los datos a elección de cada usuario y por último selecciona el grupo muscular que va a trabajar, este dato es importante rellenarlo a conciencia, ya que de ello depende que las gráficas de progreso muestren valores correctos.

Por defecto, está seleccionado el primer elemento de la lista consultada, pero este valor se puede cambiar a opción del usuario.

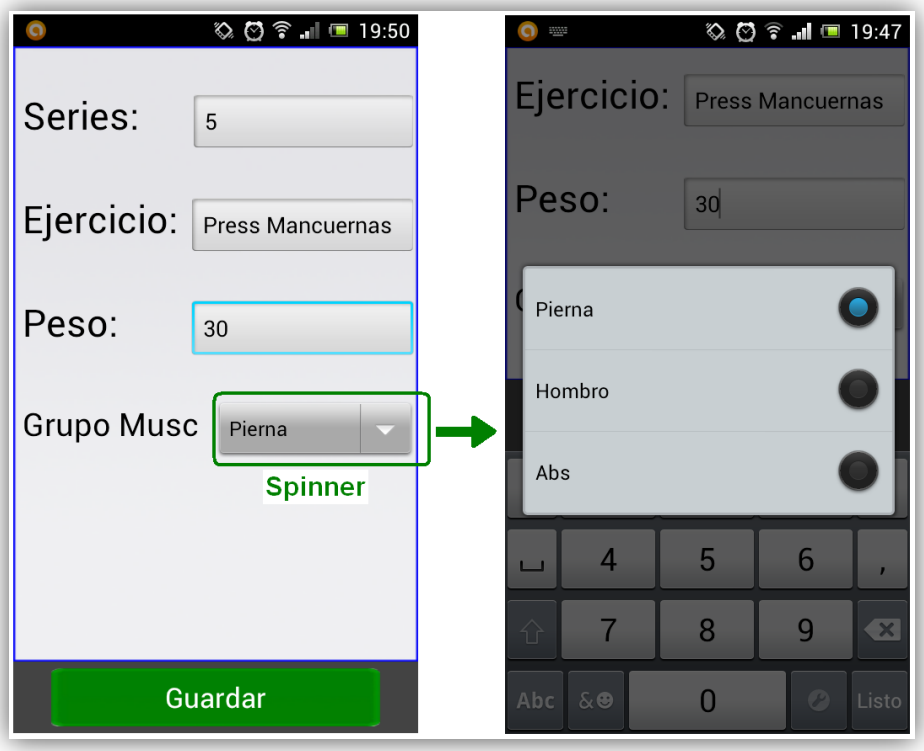



Ilustración 161 - Funcionalidad de la Clase Spinner

#### *4.2.7.5.3. Botón Guardar*

Este botón tiene la funcionalidad de insertar valores en la BD, los datos correspondientes a los ejercicios introducidos por cada usuario. Para ello, antes se debe comprobar que estos sean correctos, es por eso, que los datos que se insertan en las series y en los pesos, son de datos de tipo Numérico (Entero) y para los ejercicios solo de tipo Texto, esta opción se la doy en el Layout, definiendo el tipo de cada entra del *TextView*. Por defecto, estos campos están en blanco.

Al pulsar el botón de guardar, esta llama a un método que se llama insertar, lo siguiente que se hace, es coger los datos de los formularios de los campos *EditText*, los cuales son: la Serie, el Ejercicio y el Peso, además, de insertar información de su ubicación (P1m, P2m, P1r.., P3t), grupo muscular a entrenar y el tipo de entrenamiento seleccionado (Musculación, Resistencia, Tonificación).

El siguiente código muestra el proceso anterior descrito. Insertando sobre la BD de datos entrenamientos, usando la función insertar_bd().

```
public void insertar(View v) {

    ProgresoActivity.NumFilas = true;

    String serie = column1.getText().toString();
    String ejercicio = autoComplete.getText().toString();
    String peso = column3.getText().toString();

    // Musculacion
    if(PiernaHombro){
        //Incertamos en la BD entrenamientos
        bd.insertar_bd(serie, ejercicio, peso, GrupoM, "musculacion", "P1m");
        bd.cerrar(); // Cerramos la BD
    }
    if(PechoEspalda){
        //Incertamos en la BD entrenamientos
        bd.insertar_bd(serie, ejercicio, peso, GrupoM, "musculacion", "P2m");
        bd.cerrar(); // Cerramos la BD
    }
    if(BicepsTriceps){
        //Incertamos en la BD entrenamientos
        bd.insertar_bd(serie, ejercicio, peso, GrupoM, "musculacion", "P3m");
        bd.cerrar(); // Cerramos la BD
    }
```

Ilustración 162 - Método insertar en la BD SQLite

Una vez hecho la inserción, lo siguiente es volver a la actividad en donde se encontraba en un principio el usuario, esto lo hacemos usando la Clase *Intent* para navegar entre Actividades. Tal cual como se muestra el siguiente código.

```
// Reseteamos los Datos para la proxima vez
column1.setText("");
autoComplete.setText("");
column3.setText("");

//Toast.makeText(this, serie +" "+ ejercicio + " "+ peso , Toast.LENGTH_SHORT).show();
String idioma = getResources().getConfiguration().locale.getDisplayName().trim();
if(idioma.equals("español (España)")){
    Toast.makeText(this, "Datos introducidos correctamente... \n",
            Toast.LENGTH_SHORT).show();
}else{
    Toast.makeText(this, "Data entered correctly... \n",
            Toast.LENGTH_SHORT).show();
}

Intent i = new Intent(this, SeleccionTipoEntrenamiento.class );
i.setAction(Intent.ACTION_VIEW);
i.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(i);
finish();
```

Ilustración 163 - Volver al inicio del toso, para que se actualicen los datos

#### *4.2.7.5.4. Botón Modificar*

Como ya comente, este botón solo es visible si se quiere modificar un entrenamiento que se encuentra obviamente en nuestra tabla principal.

Así que para que este botón deba aparecer, primero teníamos que hacer una serie de comprobaciones para sacar el ***id*** de la fila consultada y así, luego poder recoger los datos correspondientes a esta, para luego ocultar el botón de Guardar y hacer visible el de Modificar.

La diferencia de este botón con respecto al de guardar, es que este actualiza la BD con el comando **UPDATE** propio de SQLite, pero usando una función que he creado, llamada ***actualizarFila***.

```
// Musculacion
if (PiernaHombro || PechoEspalda || BicepsTriceps) {

    if (PiernaHombro)
        pestaña = "P1m";
    if (PechoEspalda)
        pestaña = "P2m";
    if (BicepsTriceps)
        pestaña = "P3m";

    if (bd.actualizarFila(idFila, serie, ejercicio, peso, GrupoM,
            pestaña, "musculacion")) {
        /***** Seleccion del Idioma del telefono ******/
        String idioma = getResources().getConfiguration().locale
                .getDisplayName().trim();
        if (idioma.equals("español (España)")) {
            Toast.makeText(this, "Actualizacion Correcta \n",
                    Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Successful update \n",
                    Toast.LENGTH_SHORT).show();
        }
    } else {
        String idioma = getResources().getConfiguration().locale
                .getDisplayName().trim();
        if (idioma.equals("español (España)")) {
            Toast.makeText(this, "Error al actualizar \n",
                    Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Failed to update \n",
                    Toast.LENGTH_SHORT).show();
        }
    }
}
```

Ilustración 164 - Algoritmo para Actualizar la BD

En la imagen anterior, intento mostrar cómo sería una actualización con los parámetros pasados a esta función **actualizarFila**().

Este procedimiento es común para las opciones Resistencia y Tonificación, aunque sólo muestre el de Musculación. Además, la idea es muy simple, capturo los datos necesarios que han sido modificados, hago una actualización de esos elementos en la BD y si la actualización es correcta, muestro un mensaje indicando esto, y si no es correcto, también informo a los usuarios de que no se ha producido la actualización. Aunque se espera que la opción de información de error nunca suceda.

#### *4.2.7.5.5. CheckBox Eliminar Ejercicio*

Esta opción es importante, ya que si un usuario de la *App* quiere dar de baja un entrenamiento por los motivos que sea, solamente tenga que marcar esta opción y lo siguiente es proceder a eliminar el ejercicio elegido mediante el botón Modificar.

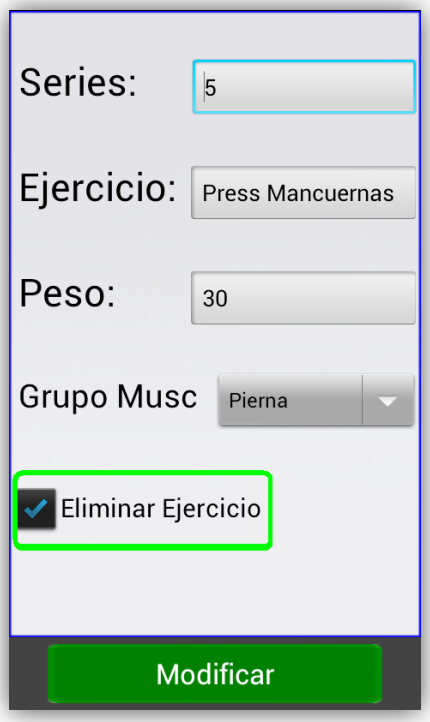



Ilustración 165 - Selección del botón eliminar ejercicio

La forma de eliminar es un poco compleja, ya que no elimino la fila de manera inmediata, si no que antes voy haciendo unas cuantas actualizaciones hacia arriba del elemento que quiero quitar. Aunque este apartado ya lo explique anteriormente.

### *4.2.8. Actividad Progreso*

Esta actividad es muy esencial en este proyecto, por lo que me sirve para mostrar información del progreso que tiene cada usuario, de manera ordenada en una gráfica estadística. Los ejes son, "Media Pesos vs Tiempo", cual ha sido el progreso evolutivo que han tenido los deportistas durante la creación de sus ejercicios.

Esta herramienta es muy importante, ya que ayuda a incentivar a las personas, al ver que su entrenamiento tiene un buen progreso. Además, esta información la muestro de manera organizada y ordenada por seis pestañas, que informan el progreso que han sufrido durante, antes y después de realizar los ejercicios. Un ejemplo seria la imagen siguiente.

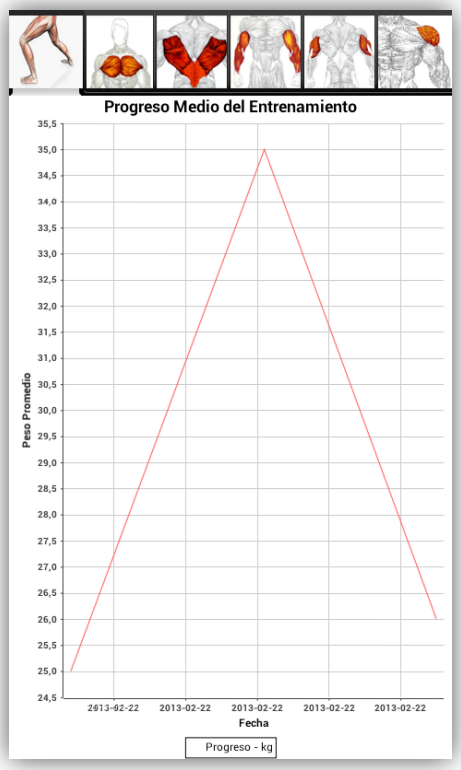



Ilustración 166 - Funcionalidad de la Clase Progreso

Cabe decir que esta herramienta en un principio pensaba diseñarla desde cero, pero investigando en la red, encontré un API (***AFreeChart***) http://code.google.com/p/afreechart/ que se encarga de hacer esto de manera rápida y sencilla. Previamente ya conocía ***JFreeChart*** http://www.jfree.org/jfreechart/ , que es propia para lenguaje java y que hace exactamente lo mismo, así que decidí buscar si tenían algo para *Android* y efectivamente lo tenían.

Además, la licencia es de tipo LGPL, lo cual significa que puede ser compartida y distribuida siempre y cuando haga referencia a ellos en mis comentarios.

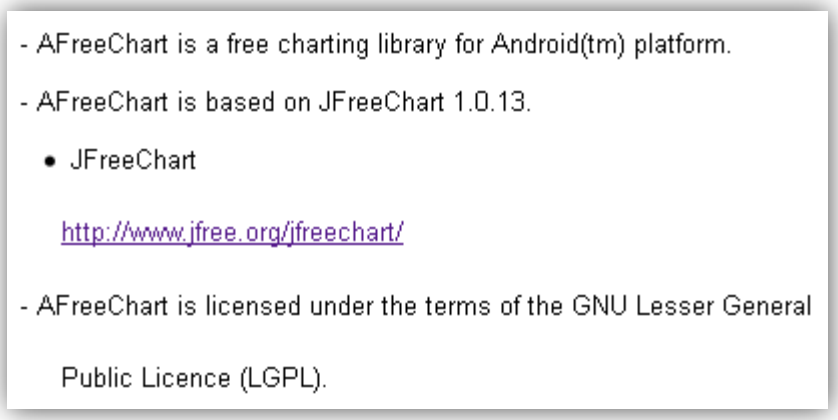



Ilustración 167 - Licencia AFreeChart

### *4.2.8.1 Configuración API AFreeChart*

Pasos para configurar adecuadamente la *API* de *AFreeChart*:

1. Descargarse el *API* (**afreechart-0.0.4.jar**) de la página principal, para este trabajo tengo instalado la última versión. Que se puede descargar de esta página: http://code.google.com/p/afreechart/downloads/list

2. Configurar Eclipse para que trabaje con esta API y que pueda reconocer sus clases en tiempo de Compilación y Ejecución. En este apartado es igual que cuando se instala la publicidad de *AdMob*, solamente que esta vez es necesario instalar otra librería (***afreegraphics.jar***) que es la que ayuda a generar una interfaz tal cual como se muestra en mi proyecto.

   La siguiente imagen muestra como se ha de cargar estas nuevas librerías.

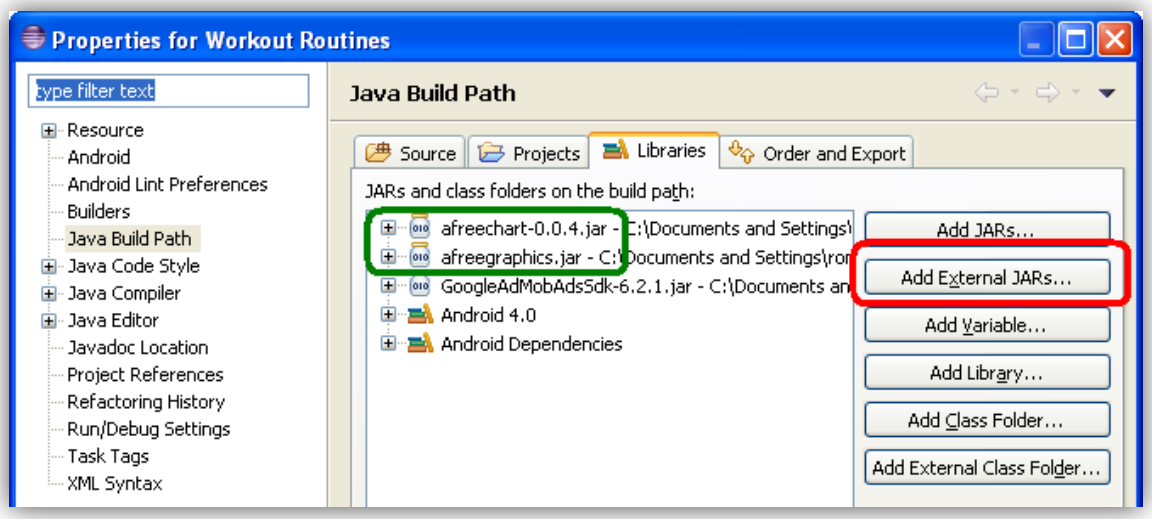



Ilustración 168 - Agregar las APIs correspondientes a AFreeChart

3. Lo siguiente es marcar la casilla correspondiente al *API AFreeChart*, para que esta cargue todas las clases necesarias a la hora de ejecutar el programa. Si no se hace esto, daría un error en la ejecución del programa en el terminal.

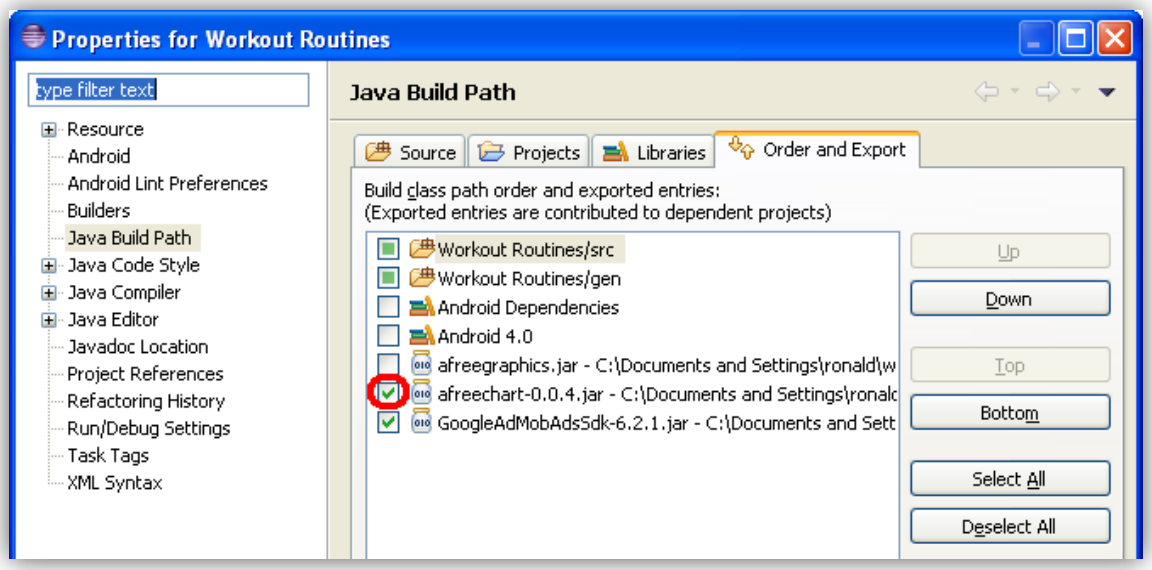



Ilustración 169 - Seleccionar el orden de ejecución de AFreeChart

Una vez configurado, lo siguiente es llamar a las clases que me construyen un interfaz de análisis, y para ello, antes hay que identificar los datos a mostrar y organizarlos para que se muestren de manera ordenada.

Los valores que voy a enseñar son los pesos por cada grupo muscular (Pecho, Espalda, Bíceps, Tríceps, Hombro y Pierna) con su correspondiente fecha de creación o modificación, ya que el tiempo es el que marca la evolución.

Estos datos los extraemos de la BD de datos de cada dispositivo móvil, particularmente de la tabla "**entrenamientos**".

Estos datos se han separado en seis pestañas diferenciadas por una imagen que representa el grupo muscular trabajado. Ya que de esta forma, los usuarios podrán identificar la evolución de su entrenamientos por cada musculo entrenado.

Los pesos que se muestran, son la media de cada entrenamiento, por ejemplo de "Musculación, Resistencia o Tonificación", ya que al tener tres diferentes rutinas, lo más probable es que los usuarios tiendan a usar dos de estas.

Para este apartado, lo que he hecho es crear una Clase llamada TabsEstadisticos.java, dentro del paquete de Actividades propias de mi proyecto.

Por otro lado, en un paquete llamado *AFreeChart*, tengo separado todo lo relacionado con estas librerías, construcción de clases, etc. Ya que al no ser de mi propiedad intelectual, me parece justo y respetable tenerlo diferenciado.

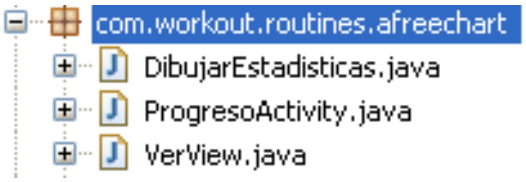



**Ilustración 170 - Paquete AFreeChart, con sus Clases**

### *4.2.8.2 Actividad TabsEstadisticos*

Esta Actividad es la encargada de generar las pestañas pertenecientes a cada grupo muscular, para ello usa la plantilla "pestañas.xml" que es la encargada de generar estas características usando la clase ***TabActivity***. La siguiente imagen muestra este proceso.

Se trata de crear Pestañas diferenciables, así que para ello creo seis variables de tipo Intent, para poder crear su correspondiente pestaña, con su imagen y posición, usando el método ***addTab***(). Además de que cada una me lleva a un *Frame* independiente de los demás, con resultados distintos y semejantes características.

```
Intent i1, i2, i3, i4, i5, i6;

i1 = new Intent().setClass(this, ProgresoActivity.class);
i1.putExtra("Pierna", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña1").
        setIndicator("",getResources().
                getDrawable(R.drawable.pierna)).setContent(i1));
tabHost.getTabWidget().getChildAt(0); // que sea la pestaña 1

i2 = new Intent().setClass(this, ProgresoActivity.class);
i2.putExtra("Pecho", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña2").
        setIndicator("", getResources().
                getDrawable(R.drawable.pecho)).setContent(i2));
tabHost.getTabWidget().getChildAt(1); // que sea la pestaña 2

i3 = new Intent().setClass(this, ProgresoActivity.class);
i3.putExtra("Espalda", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña3").
        setIndicator("", getResources().
                getDrawable(R.drawable.espalda)).setContent(i3));
tabHost.getTabWidget().getChildAt(2); // que sea la pestaña 3

i4 = new Intent().setClass(this, ProgresoActivity.class);
i4.putExtra("Biceps", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña4").
        setIndicator("", getResources().
                getDrawable(R.drawable.biceps)).setContent(i4));
tabHost.getTabWidget().getChildAt(3); // que sea la pestaña 1

i5 = new Intent().setClass(this, ProgresoActivity.class);
i5.putExtra("Triceps", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña5").
        setIndicator("", getResources().
                getDrawable(R.drawable.triceps)).setContent(i5));
tabHost.getTabWidget().getChildAt(4); // que sea la pestaña 2

i6 = new Intent().setClass(this, ProgresoActivity.class);
i6.putExtra("Hombro", true); //Llevar al Bundle
tabHost.addTab(tabHost.newTabSpec("Pestaña6").
        setIndicator("", getResources().
                getDrawable(R.drawable.hombro)).setContent(i6));
tabHost.getTabWidget().getChildAt(5); // que sea la pestaña 3
```

Ilustración 171 - Creación de Pestañas para la Clase Progreso

### *4.2.8.3. Actividad ProgresoActivity*

En esta actividad lo que pretendo hacer es una consulta a una función propia de esta clase, llamada **mediaGrupoMuscular**, la cual me calcula la media por cada peso perteneciente a cada entrenamiento y a cada grupo elegido.

```
/** Elijo la pestaña para sacar el progreso por cada grupo muscular **/
Bundle bundle   = getIntent().getExtras();
Boolean pierna  = bundle.getBoolean("Pierna");
Boolean pecho   = bundle.getBoolean("Pecho");
Boolean espalda = bundle.getBoolean("Espalda");
Boolean hombro  = bundle.getBoolean("Hombro");
Boolean biceps  = bundle.getBoolean("Biceps");
Boolean triceps = bundle.getBoolean("Triceps");

if(NumFilas){
    String media1 = Integer.toString(mediaGrupoMuscular(Pierna));
    String media2 = Integer.toString(mediaGrupoMuscular(Pecho));
    String media3 = Integer.toString(mediaGrupoMuscular(Espalda));
    String media4 = Integer.toString(mediaGrupoMuscular(Hombro));
    String media5 = Integer.toString(mediaGrupoMuscular(Biceps));
    String media6 = Integer.toString(mediaGrupoMuscular(Triceps));

    Calendar c = Calendar.getInstance();
    int mes   = c.get(Calendar.MONTH);
    int año   = c.get(Calendar.YEAR);
    int dia   = c.get(Calendar.DAY_OF_MONTH);
    int hora     = c.get(Calendar.HOUR_OF_DAY);
    int minutos  = c.get(Calendar.MINUTE);
    int segundos = c.get(Calendar.SECOND);

    mes = mes+1;
    String date = año+"-"+mes+"-"+dia+"-"+hora+"-"+minutos+"-"+segundos;

    bd.insertarFvsP(date, media1, media2, media3, media4, media5, media6);
    bd.cerrar();
}
NumFilas = false;
```

Ilustración 172 - Algoritmo para insertar dato en la tabla Medias

Una vez calculada la media, lo siguiente es insertar estos valores en la tabla "medias", la cual me guarda cualquier modificación correspondiente a los pesos, incluyendo la fecha, por si no se tiene. Luego se hará una consulta a esta tabla para que AFreeChart pueda representare gráficamente estos datos.

Ahora, mostrare la función mediaGrupoMuscular, que fue creada para calcular la media de los diferentes entrenamientos por cada grupo muscular.

```
private int mediaGrupoMuscular (String musculo){
    int i, sumaPesos = 0, N=0, N1=0, N2=0, N3=0;
    int[] PM = bd.getPesosByMusc(musculo, "pesom");
    int[] PR = bd.getPesosByMusc(musculo, "pesor");
    int[] PT = bd.getPesosByMusc(musculo, "pesot");

    // Para pesos de Musculacion
    for(i=0; i< PM.length; i++){
        if(PM[i] == 0){
            // Si el peso es Cero, quitamos el divisor y no Sumamos
            N1--;
        }else{
            sumaPesos = sumaPesos + PM[i];
            N1++;
        }
    }
    //Para pesos de Resistencia
    for(i=0; i< PR.length; i++){
        if(PR[i] == 0){
            // Si el peso es Cero, quitamos el divisor y no Sumamos
            N2--;
        }else{
            sumaPesos = sumaPesos + PR[i];
            N2++;
        }
    }

    for(i=0; i< PT.length; i++){ // Para pesos de tonificacion
        if(PT[i] == 0 ){
            // Si el peso es Cero, quitamos el divisor y no Sumamos
            N3--;
        }else{
            sumaPesos = sumaPesos + PT[i];
            N3++;
        }
    }
    N = (N1+N2+N3) * -1;

    int MEDIA;
    if(PM.length <= 0 && PR.length <= 0 && PT.length <=0){
        // Si los tres banderines son ciertos, quiere decir que
        // no hay nada en la BD Por tanto la media es Cero
        MEDIA = 0;
    }
    else{
        MEDIA = sumaPesos / N;
    }
    return MEDIA;
}
```

Ilustración 173 - Método que calcula la media entre los grupos musculares

Básicamente, esta función lo que hace es extraer los datos (Pesos) de la tabla entrenamientos y guardarlos en un Array de enteros, perteneciente a cada grupo muscular consultado, por cada entrenamiento.

Una vez obtenido los datos, recorremos cada *Array* para ir sumando los pesos, además de que tenemos un contador que nos dice cuántos elementos han sido acumulados y sean diferentes de un peso igual a Cero.

Una vez hecho esto para los tres *Arrays*, lo siguiente es hacer la media y retornar este valor.

```
int P[];
String fecha[];
P = bd.MediasPesos(indice);
fecha = bd.MediasFecha();


/** funcion que llama al AFRECHART para dibujar los datos **/
/** (Peso vs Fecha) **/
DibujarEstadisticas mView = null;
mView = new DibujarEstadisticas(this, P, fecha);
requestWindowFeature(Window.FEATURE_NO_TITLE);
setContentView(mView);

P = null;
fecha = null;
```

Ilustración 174 - Ejecutar la Clase DibujarEstadisticas

Por último, queda llamar a la case **DibujarEstadísticas.java**, Clase que extiende de *AFreeChart* para que me construya estos datos. A esta clase le pasamos la media de pesos y su correspondiente fecha de creación por cada ejercicio.

### *4.2.8.4. Actividad DibujarEstadísticas*

En esta clase, lo que se pretende crear lo que sería el entorno gráfico, donde se mostraran los datos relacionados con la evolución del entrenamiento (fuerza/peso), configurando las características más habituales: color de fondo, trazado de la líneas (Color), un título, subtítulos para los ejes X e Y, y un formato de fecha. En la siguiente imagen muestra cómo se hace.

```
public DibujarEstadisticas(Context context, int[] peso, String[] fecha){
    super(context);

    AFreeChart  chart = createChart(createDataset(peso, fecha));
    setChart(chart);
}

@SuppressLint("SimpleDateFormat")
private static AFreeChart createChart(XYDataset dataset) {

    AFreeChart chart = ChartFactory.createTimeSeriesChart(
        "Progreso Medio del Entrenamiento",  // title
        "Fecha",             // x-axis label
        "Peso Promedio",      // y-axis label
        dataset,             // data
        true,                // create legend?
        true,                // generate tooltips?
        false                // generate URLs?
    );

    chart.setBackgroundPaintType(new SolidColor(Color.WHITE));

    XYPlot plot = (XYPlot) chart.getPlot();
    plot.setBackgroundPaintType(new SolidColor(Color.WHITE)); // Fondo
    plot.setDomainGridlinePaintType(new SolidColor(Color.LTGRAY));
    plot.setRangeGridlinePaintType(new SolidColor(Color.LTGRAY));

    plot.setAxisOffset(new RectangleInsets(1.0, 1.0, 1.0, 1.0));
    plot.setDomainCrosshairVisible(true);
    plot.setRangeCrosshairVisible(true);

    XYItemRenderer r = plot.getRenderer();
    if (r instanceof XYLineAndShapeRenderer) {
        XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) r;
        renderer.setBaseShapesVisible(true);
        renderer.setBaseShapesFilled(true);
        renderer.setDrawSeriesLineAsPath(true);
    }

    DateAxis axis = (DateAxis) plot.getDomainAxis();
    axis.setDateFormatOverride(new SimpleDateFormat("yyyy-MM-d"));

    return chart;
}
```

Ilustración 175 - Creación de la representación gráfica del Progreso

Lo siguiente es crear la función a usar para representar mejor los datos, así que para eso me ayudo de la clase ***XYDataSet***, que lo que hace es unirme dos puntos a través de una línea recta, esto lo represento en un plano cartesiano con los datos pesos y su fecha asociada. La siguiente imagen muestra como se ha creado esto.

```
public XYDataset createDataset(int[] pesos, String[] fecha)  {
    TimeSeries s1 = new TimeSeries("Progreso - kg"); // Labels

    // Recorremos todos los pesos pasados por la consulta a la BD
    for(int i=0; i < pesos.length ; i++){
        String[] Date = fecha[i].split("-");
        int mes = Integer.parseInt(Date[1]);
        int año = Integer.parseInt(Date[0]);
        int dia = Integer.parseInt(Date[2]);
        int hora = Integer.parseInt(Date[3]);
        int min = Integer.parseInt(Date[4]);
        int seg = Integer.parseInt(Date[5]);

        // Solo mostramos el ultimo dato actualizado de
        // varios existentes para un dia
        s1.addOrUpdate(new Second(seg, min, hora,dia, mes, año), pesos[i]);
        Date = null;
    }

    TimeSeriesCollection dataset = new TimeSeriesCollection();
    dataset.addSeries(s1);

    return dataset;
}
```

Ilustración 176 - Método encargado de agregar los datos para representarlos en forma de líneas

En el código anterior, cabe destacar la clase ***Second***, ya que con ella voy construyendo las fechas y sus pesos a mostrar, he usado esta clase, ya que es la que me da mayor rango de diferencia entre una fecha y otra. La razón, es que si un usuario inserta o modifica en un mismo un entrenamiento, este no se va a ver reflejado en las gráficas, pero si tomo en cuenta los segundos, sí que se notara el cambio.

Por otra parte, el método ***addOrUpdate***() me viene muy bien para no tener errores de ejecución, por lo que un datos puede ser igual a otro, y si este caso se da, lo único que tiene que es hacer es actualizarlo si ya existía, de lo contrario lo agrega.

### *4.2.8.5. Clase VerView*

Esta clase no es de mi propiedad, la tome como ayuda para poder diseñar el entorno gráfico en donde se mostraran los datos por cada usuario. Esta clase pertenece a AFreeChart y es la que ellos usan para representar los datos que muestran como ejemplos. Y dado el caso también la utilizo, ya que la construcción que ellos hacen, luego la agrego a mi proyecto. Además es de licencia ***GNU***. Para poder trabajar con esta clase hay que usar la librería ***afreegraphics.jar***.

### 4.2.9. Actividad Ejercicios

Esta herramienta está diseñada para ayudar a los usuarios como ejecutar ejercicios de forma correcta, ya que sin esto, los deportistas pueden sufrir graves lesiones musculares.

En esta actividad se mostrarán alrededor de 40 ejercicios que ayudarán a los usuarios a identificar por el tipo de musculo, como se llama cada entrenamiento y como se debe ejecutar. Tal cual como muestra la siguiente ilustración. Donde muestro como ejemplo el musculo a trabajar, que son los bíceps, y luego una lista de posibles entrenamientos para este grupo muscular, con su correspondiente imagen que muestra cómo se deben ejecutar cada ejercicio.

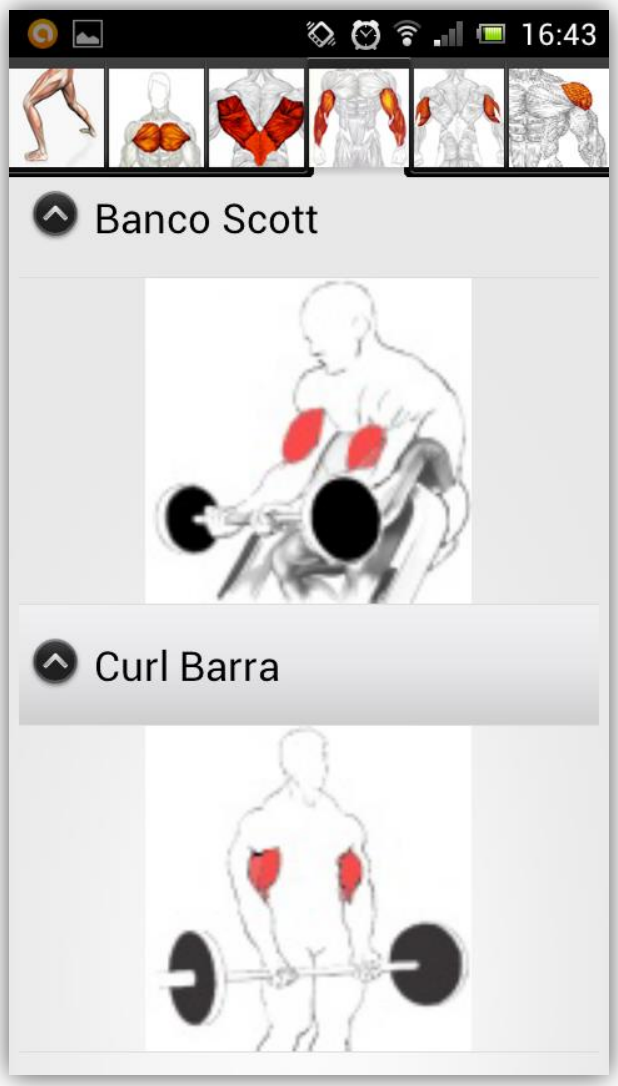



Ilustración 177 - Funcionalidad final de la Actividad Progreso

#### 4.2.9.1. Actividad TabsListasDesplegables

Esta clase lo que hace es crear seis pestañas, una para cada musculo donde le asignamos una imagen para su correspondiente localización. Esta utiliza la plantilla "pestanas.xml", que es la que trabaja con la clase TabActivity para generar pestañas.

Esta clase es igual que la clase TabsEstadisticos, la cual utilizamos para crear gráficas de progreso. Solo que en vez de usar las clase de AFreeChart después de navegar con los Intents, usamos una propia, la cual se llama ListaEjecutarEjercicios.java.

#### 4.2.9.2. Actividad ListaEjecutarEjercicios

Esta Activity utiliza y extiende de la Clase ExpandableListActivity, ayudando a crear listas dinámicas con la opción de expandir cada ítem para mostrar más sub-ítems.

Esta nos viene bien, ya que queremos mostrar por cada ejercicio existente y para cada grupo muscular, una imagen que muestre como se hace el entrenamiento a buscar por el usuario. La próxima gráfica muestra un ejemplo de la utilización de esta utilidad.

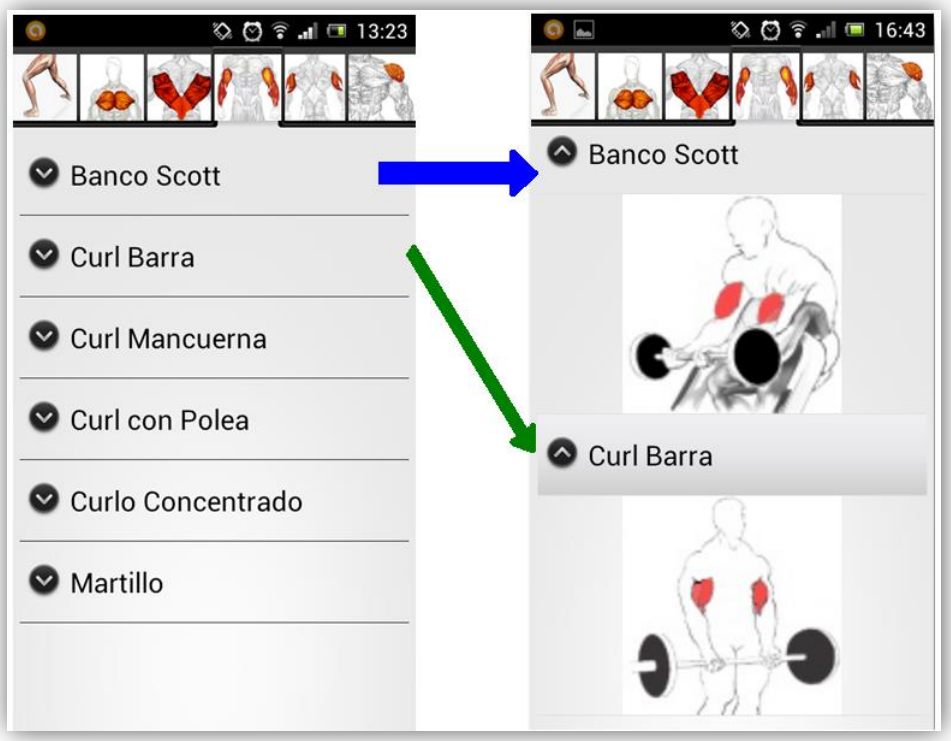



Ilustración 178 - Funcionalidad de la Actividad Ejercicios

Lo siguiente a mostrar es como se crea un XML que sea de tipo ExpandableListView, para ello me ayudo del código xml y de una gráfica del resultado que mostrara la paleta de diseño de Android.

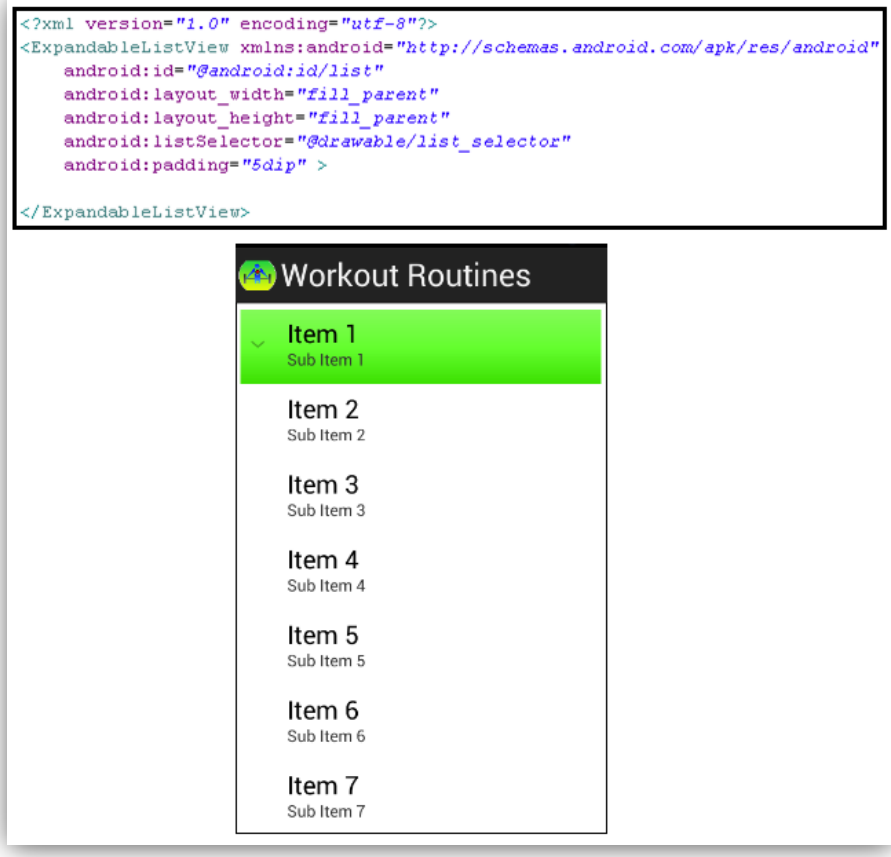



Ilustración 179 - Diseño en XML de la Actividad Ejercicios

Lo siguiente es trabajar con código java, lo primero que tenemos que hacer es sacar los padres o el ítem principal que componen las listas con los ejercicios que queremos enseñar.

Luego hacemos una lista separada por cada pestaña, así que hacemos una consulta para cada grupo muscular existente y luego lo que haremos es irla cargando en un ExpandableListView cada una y en su propia pestaña.

La lista la obtenemos de los ejercicios estáticos, que se encuentran en la tabla de "formas", para ello, hacemos una consulta por cada grupo muscular y mostramos la lista ordenada. La función se llama listaGrupoMuscularFormas(). Método creado en la clase SQLiteBD pata este propósito.

```
if("Pierna" == pierna)
{
    padres = bd.listaGrupoMuscularFormas(pierna);

    //Cargamos el array de Fotos o Imagenes de los ejercicios
    fotos = EjerciciosEstaticos.ArrayDeImagenes("Pierna");
}


// Sacamos los padres a partir de la consulta a la BD
final ArrayList<HashMap<String, String>> headerData = new ArrayList<HashMap<String, String>>();
for(int i=0; i< padres.length; i++)
{
    HashMap<String, String> groupadres = new HashMap<String, String>();

    groupadres.put(NAME, padres[i]);
    headerData.add( groupadres );

}
```

Ilustración 180 - Código para consultar los ejercicios existentes

Las consultas a la BD son iguales para los seis grupos musculares existentes, aunque en el ejemplo anterior solo mostremos uno. (Pierna).

El código anterior, muestra cómo se enlazan las listas de la BD con el XML que nos sirve de platilla para ir creando los ítems principales. En un Array de String vamos almacenando los ejercicios creados anteriormente para este apartado y los vamos recorriendo uno a uno y los voy agregando a la cabecera.

Lo siguiente es ir creando los sub-ítems para cada ítem principal, en este caso, los hijos serán las imágenes que queremos mostrar al pulsar en la lista seleccionada. El código siguiente muestra cómo se hace. Ya que recorremos los padres por cada grupo y vamos dejando su correspondiente hijo a través de una imagen que se va cargando de la carpeta drawable.

```
// Sacamos un hijo por cada padre
final ArrayList<ArrayList<HashMap<String, Object>>> childData;
childData = new ArrayList<ArrayList<HashMap<String, Object>>>();
for(int i=0; i< padres.length; i++)
{
    final ArrayList<HashMap<String, Object>> groupdata;
    groupdata = new ArrayList<HashMap<String, Object>>();
    childData.add(groupdata);
    HashMap<String, Object> map = new HashMap<String,Object>();

    //recorremos los hijos, por cada padre
    map.put(IMAGE, getResources().getDrawable(fotos[i]));

    groupdata.add(map);
}
```

Ilustración 181 - Código para añadir una imagen por cada ejercicio consultado

Las imágenes fueron creadas y llamadas tal cual como se muestran en la lista que se encuentra en la tabla de formas, ya que de esta manera es más fácil hacer una asociación.

Las imágenes fueron diseñadas manualmente y asociadas a los Array de igual forma, luego estas fueron introducidas en un método que devuelve un Array de enteros, ya que las imágenes guardan un valor de tipo entero cuando son referenciadas a estas a través del archivo R.java.

La siguiente imagen, muestra cómo fue creada esta función.

```
public static int[] ArrayDeImagenes(String grupoM) {

    int img[] = null;

    if(grupoM == "Pierna"){
        img = new int[5];
        //Pierna
        img[0]  = R.drawable.curl_femoral;
        img[1]  = R.drawable.elevacion_talones;
        img[2]  = R.drawable.extension_pierna;
        img[3]  = R.drawable.prensa;
        img[4]  = R.drawable.sentadilla_multipower;
    }
    if(grupoM == "Pecho"){
        img = new int[10];

        //Pecho
        img[0]  = R.drawable.aperturas;
        img[1]  = R.drawable.cruce_poleas;
        img[2]  = R.drawable.pectoral_contractor;
        img[3]  = R.drawable.press_banca;
        img[4]  = R.drawable.press_banca_declinado;
        img[5]  = R.drawable.press_banca_inclinado;
            //=press_mancuerna;
        img[6] = R.drawable.press_mancuerna_declinado;
        img[7] = R.drawable.press_mancuerna_inclinado;
        img[8] = R.drawable.press_maquina;
        img[9] = R.drawable.pullover;

    }
    if(grupoM == "Espalda"){
```

Ilustración 182 - Método que almacena la dirección de las imágenes a usar en la Clase Ejercicios

Esta funcionalidad se encuentra en una clase externa que se llama EjerciciosEstaticos.java, la cual tiene como finalidad, dar apoyo a la creación de métodos donde la construcción de estas funciones implique que se hagan de forma manual.

### *4.2.10. Actividad Exportar – Importar*


**Ilustración 183 - Funcionalidad y diseño de la Actividad Exportar e Importar**

Esta actividad es la que hace diferente de otras aplicaciones, ya que como muchas otras *App* existentes, solo esta espera poder ayudar al intercambio de rutinas entre deportistas, gracias a las funcionalidades de Exportar e Importar.

El diseño de esta característica es muy simple, solo dos botones, uno que se encargue de exportar y el otro de importar, y que lo haga de manera autónoma y autosuficiente para que las personas no tenga que intervenir demasiado con este proceso, que deberá ser simple y sencillo a la vista de los usuario.

Para este diseño, solo se necesita llamar a su layout correspondiente, que se llama ***exportar_importar.xml***, la cual se encarga de gestionar las acciones anteriormente mencionadas. Todo el código se gestiona en la clase **ExportarImportar.java**.

#### *4.2.10.1. Botón Exportar*

Este botón utiliza como comunicación entre dispositivos el Bluetooth, ya que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia.

Los móviles *Android*, cuentan con esta tecnología, así que lo único que tengo que hacer es crear un enlace con otro dispositivo y adjuntar el **fichero.db** que quiero enviar al otro móvil. El nombre del fichero en cuestión que envió se llama "***Rutina.csv***", tiene esta extensión porque son un tipo de documento en formato abierto y sencillo

para representar datos en forma de tabla y a la hora de importar es más fácil leer estos datos.

La siguiente imagen muestra como se hizo la construcción de la función **Exportar**. Para ello se basa en una Clase java (***FileManager.java***) que hace las funciones de leer y escribir sobre ficheros, además, esta se encuentra en el paquete de clases, separada de las demás porque su función es de apoyo.

```
Button exportar = (Button) findViewById(R.id.bexport);
exportar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        if(bd.getNumRows() != 0){
//String Texto = "SerieM , SerieR , SerieT , Ejercicios , PesoM , PesoR , PesoT, Grupo, Pestaña";
            String Texto = "";
            String[][] Tablas = bd.mostrarFilasMatriz();

            for (int i=1; i <= Tablas.length - 1; i++)
            {
                Texto  = Texto + Tablas[i][1] + " , " + Tablas [i][2] + " , "+ Tablas[i][3]+
                         " , " + Tablas[i][4] + " , " + Tablas[i][5] + " , " + Tablas [i][6] +
                         " , " + Tablas[i][7] + " , " + Tablas[i][8] + " , " + Tablas [i][9] +
                         "\n";
            }

            FileManager fm = new FileManager();
            fm.writeFile("Rutina.csv", Texto ); // Creamos el Fichero
            Uri uri  =  Uri.fromFile(fm.ficheroCSV()); // cogemos la ruta del fichero --> URI

            final Intent intent = new Intent(Intent.ACTION_SEND);

            intent.setType("plain/text");   // Enviar solo a dos (Bluetooth o Gmail)
            intent.putExtra(Intent.EXTRA_SUBJECT, "Download WorkOut Fitnees");
            intent.putExtra(Intent.EXTRA_TEXT, "Download my app to Android at:  " +
                    "httt://roniapps.net23.net");
            intent.putExtra(Intent.EXTRA_STREAM, uri); // Pasamos la ruta del file a enviar

            startActivity(Intent.createChooser(intent, getString(R.string.app_name)));
        }
        else{

             /*****Seleccion del Idioma del telefono ******/
            String idioma = getResources().getConfiguration().locale.getDisplayName().trim();
            if(idioma.equals("español (España)")){
                Funciones.imprimir(v.getContext(), "No hay nada que exportar. \n" +
                        "Por favor, rellene su entrenamiento personalizado");
            }else{
                Funciones.imprimir(v.getContext(), "Nothing to export. \n" +
                        "Please fill in your personal training");
            }
        }
    }
});
```

**Ilustración 184 - Código para exportar Datos**

El código anterior, lo que intenta hacer es capturar la opción deseada, para este caso la de exportar, luego de encargarse de averiguar si hay algo para para exportar, ya que la idea no es exportar algo  en blanco, así que primero pregunto a la BD si hay al menos un elemento, de lo contrario muestro su mensaje de error correspondiente.

Una vez que sabemos que hay algo que exportar (Se ha creado una rutina personalizada), lo siguiente es crear el fichero Rutina.csv, para ello usamos el método

**writeFile**(), el cual nos crea el fichero con el contenido que le pasemos, el contenido serán las rutinas a exportar, las cuales la extraemos leyendo de la base de datos y guardándolas en un String llamado Texto.

Una vez que nos crea el fichero con el contenido a enviar, lo siguiente es elegir como enviarlo, para ello nos valemos de la clase **Intent**, el cual le pasamos un parámetro el cual es ***ACTION_SEND***, este método es igual a la hora de crear le botón de compartir la aplicación, solo que esta vez vamos a limitar las opciones y a enviar datos.

```
final Intent intent = new Intent(Intent.ACTION_SEND);

intent.setType("plain/text");    // Enviar solo a dos (Bluetooth o Gmail)
intent.putExtra(Intent.EXTRA_SUBJECT, "Download WorkOut Fitnees");
intent.putExtra(Intent.EXTRA_TEXT, "Download my app to Android at:  " +
        "http://workoutroutines.hol.es/");
// Pasamos la ruta del file a enviar
intent.putExtra(Intent.EXTRA_STREAM, uri);

startActivity(Intent.createChooser(intent, getString(R.string.app_name)));
```

Ilustración 185 - Usar la Clase Intent para cargar la tabla y luego enviar los datos a otro móvil

Para restringir las opciones solo basta con poner

```
intent.setType("plain/text");    // Enviar solo a dos (Bluetooth o Gmail)
```

, de esta manera solo vamos a tener la posibilidad de enviar datos a través de Bluetooth o vía email, dependiente de las aplicaciones que tengan los usuarios instalados en su dispositivo móvil. Tal como se muestra en la imagen siguiente.

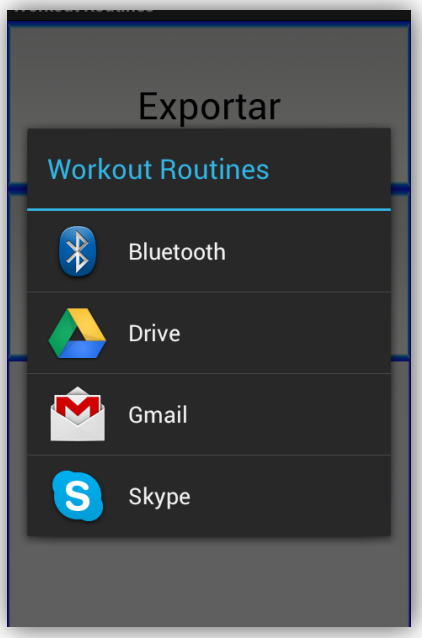



Ilustración 186 - Ejecución de la herramienta Exportar

Y para enviar el fichero, lo único que tenemos que hacer es pasarle al ***Intent*** la ruta del fichero de donde se almacena, esto lo hacemos con la clase ***URI***, tal cual como se muestra en la siguiente gráfica.

```
// cogemos la ruta del fichero --> URI
Uri uri  =  Uri.fromFile(fm.ficheroCSV());
intent.putExtra(Intent.EXTRA_STREAM, uri);
```

**Ilustración 187 - Código para cargar la tabla BD al Bluetooth**

### 4.2.10.2. Botón Importar

```
FileManager fm = new FileManager();
String[] filas = fm.readFile(v.getContext(), "Rutina.csv"); // Leemos el fichero

if(filas != null){

    StringBuilder unir =  new StringBuilder();
    String[] c; // Las palabras que iran a cada columna para ser insertadas en la BD
    for(int i=0; i< filas.length; i++){
        // Separamos por el espacio separador y optenemos las palabras por cada fila
        c = filas[i].toString().split(",");

        // Seriem, Serier, Seriet, Ejericio, Pesom, Pesor, Pesot, Grupo, Pestaña
        bd.importarRutina(c[0].trim(), c[1].trim(), c[2].trim(), c[3].trim(), c[4].trim(),
                          c[5].trim(), c[6].trim(), c[7].trim(), c[8].trim());

        c=null; // Reseteo el Array

        //Quitamos Nulos, Comas y Espacios en Blancos
        unir.append(i+1+") " + filas[i].replace("null", "").replace(",","").trim());
        unir.append('\n');
    }
    bd.cerrar();
```

**Ilustración 188 - Código para importar ejercicios de otro dispositivo**

Esta funcionalidad es similar al de exportar, en cuanto a lo de capturar e identificar la pulsación del botón Importar, así que solo voy a explicar cómo es que se lee de un fichero.

Para ello, le pasamos el nombre del fichero a leer al método ***readFile()*** de la Clase de apoyo FileManager.java, el cual me crea en un Array de String las filas creadas, después de esto lo único que queda es leer cada fila con su correspondiente columna, para luego pasar estos datos a la función **importarRutina()** de la clase SQLiteBD, siendo esta la que me va ir insertando fila a fila los elementos leídos. La siguiente imagen muestra un funcionamiento correcto de esta utilidad.


Ilustración 189 - Ejecución exitosa de la opción Importar

Además, si el fichero no existe o no esta en la ruta deseada, le mostramos el error correspondiente al usuario, el cual debe mover este fichero a la ruta que le específico para que la importación sea lo más correcta.
Para mover fichero de un lugar a otro es necesario que el usuario utilice una herramienta de gestor de ficheros, *Android Market* tiene muchas aplicaciones que hacen esto posible.

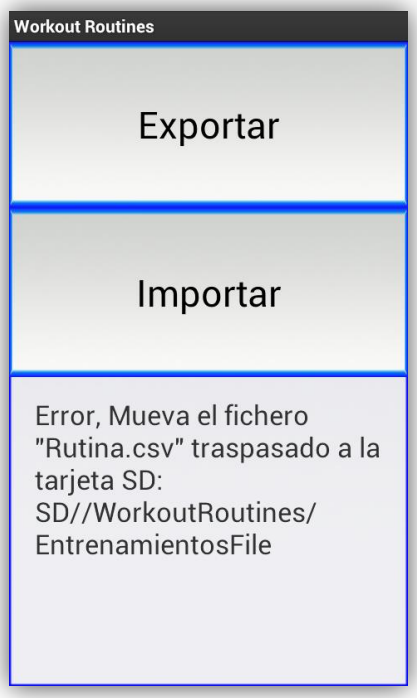



Ilustración 190 - Ejecución con error de la opción Importar

### *4.2.10.3. Funciones de apoyo exportar e importar Clase FileManager.java*

Estas funciones tratan de gestionar todo aquello a lo que se refiere a leer y escribir sobre la tarjeta *SD*, así que en el móvil se deben habilitar algunos permisos necesarios. Estos permisos son para poder trabajar con la tarjeta *SD* y poder trabajar con el Bluetooth.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Ilustración 191 - Habilitar permisos en los dispositivos

La imagen anterior muestra cómo se otorgan estos permisos al dispositivo móvil, esto lo hacemos en el fichero ***AndroidManifest.xml***.
Una vez que tenemos estos permisos otorgados, lo siguiente es crear las funciones que me crean, escriben y leen sobre la tarjeta SD.

Lo primero que hay que especificar y dejar bien claro, es donde se va a guardar y crear nuestro fichero, para ello buscamos la ruta raíz (*Path*) de nuestra SD, luego a esta ruta le agregamos una nueva, la que serán las capetas contenedoras de nuestro fichero. La siguiente imagen muestra cómo se hace este proceso.

```
// para poder obtener la ruta de nuestra tarjeta SD (getExternalStorageDirectory())
String ruta = Environment.
        getExternalStorageDirectory().
        getAbsolutePath()+"/WorkoutRoutines/EntrenamientosFile";
File file = null;
```

Ilustración 192 - Obtener la ruta de nuestro directorio en la SD

Ahora creamos dos métodos estáticos, el cual nos ayuda a saber si tenemos la tarjeta *SD* en nuestros dispositivos y sí podemos guardar sobre ella. Además, retornan un booleano indicando si se puede o no trabajar con la *SD*.

```
public static boolean isExternalStorageReadOnly() {
    String extStorageState = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState)) {
        return true;
    }
    return false;
}

public static boolean isExternalStorageAvailable() {
    String extStorageState = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(extStorageState)) {
        return true;
    }
    return false;
}
```

Ilustración 193 - Métodos que identifican la disponibilidad de la SD en cada móvil

Lo siguiente es crear el método de escribir sobre un fichero, para ello mostramos el siguiente código que muestra como se hizo el método ***writeFile***()

```
// Escribimos sobre el fichero
public void writeFile(String filename, String textfile) {

    try {
        if (isExternalStorageAvailable() && !isExternalStorageReadOnly()) {

            file = new File(ruta, filename);

            file.mkdirs(); // Creamos los directorios si no existen

            if (file.exists()) // Si existe el fichero lo elimino
            {
                file.delete();
            }
            OutputStreamWriter outw = new OutputStreamWriter(
                    new FileOutputStream(file));
            outw.write(textfile); // Le pasamos el texto a crear
            outw.close();
        }
    } catch (Exception e) {
    }
}
```

Ilustración 194 - Método para crear y leer en un fichero

Este método recibe como parámetros el nombre del fichero a crear y el contenido que este llevara. Luego usando la clase File y la ruta antes mencionada, procedemos a crear el archivo, mediante su método ***mkdirs***(), que crea los directorios necesarios si no existen y por consiguiente el documento, el cual será la BD de datos de los usuarios que quieren exportar. Por último, queda escribir sobre el fichero creado, esto lo hacemos con la Clase ***OutputStreamWriter***, que mediante el método write() le pasamos el contendido a escribir sobre el fichero creado anteriormente.

Por último, queda el método de lectura sobre un fichero ***readFile***(), este método se encargara de leer sobre la tarjeta *SD* el fichero que le pasamos, solo lee el contenido y lo retorna en un *Array* de *String*.

La siguiente imagen muestra cómo se creó este proceso. Donde crea un *buffer* de lectura sobre el objeto de tipo ***File***, al cual le pasamos la ruta y nombre del fichero a leer.

Cada vez que se lee una fila, este lo va almacenando en un ***ArrayList***, ya que no sabemos cuántas líneas va a leer, luego queda pasar esto a un *Array* de *String* y retornarlo.

```
// Leemos sobre el fichero
public String[] readFile(Context c, String filename) {

    try {

        String[] filas = null;
        ArrayList<String> arrayList = new ArrayList<String>();
        if (isExternalStorageAvailable()) {

            file = new File(ruta, filename);
            file.mkdirs(); // Creamos los directorios si no existen
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    new FileInputStream(file)));
            String Leido;
            int i = 0;

            while ((Leido = br.readLine()) != null) {

                // Creo un arrayList, porque no se cuantas lineas voy a leer
                arrayList.add(Leido);
                i++;
            }
            br.close();

            filas = new String[i];

            i = 0; // Reseteo el contador
            // Iteracion para recorrer el ArrayList
            Iterator<String> it = arrayList.iterator();
            while (it.hasNext()) {
                filas[i] = it.next();
                i++;
            }

            return filas;

        } else {
            return filas;
        }
    } catch (Exception e) {

    }
    return null;
}
```

Ilustración 195 - Método para leer un fichero ubicado en la tarjeta SD

### *4.2.11. Actividad Encuesta*

Esta actividad ha sido creada con la intención de saber la aceptación que tiene esta aplicación en los usuarios, para ello me valgo de Cuatro preguntas sencillas y fáciles de contestar, ya que son de tipo test, además que tengo un cuadro de dialogo, diseñado para que los deportistas comenten más dudas o recomendaciones que tengan.

Las preguntas son las siguientes. La imagen esta retocada para mostrar las preguntas de forma completa. Como se ve, son muy fáciles de contestar y de esta manera garantizamos que no se cansen a la hora de rellenar este formulario.

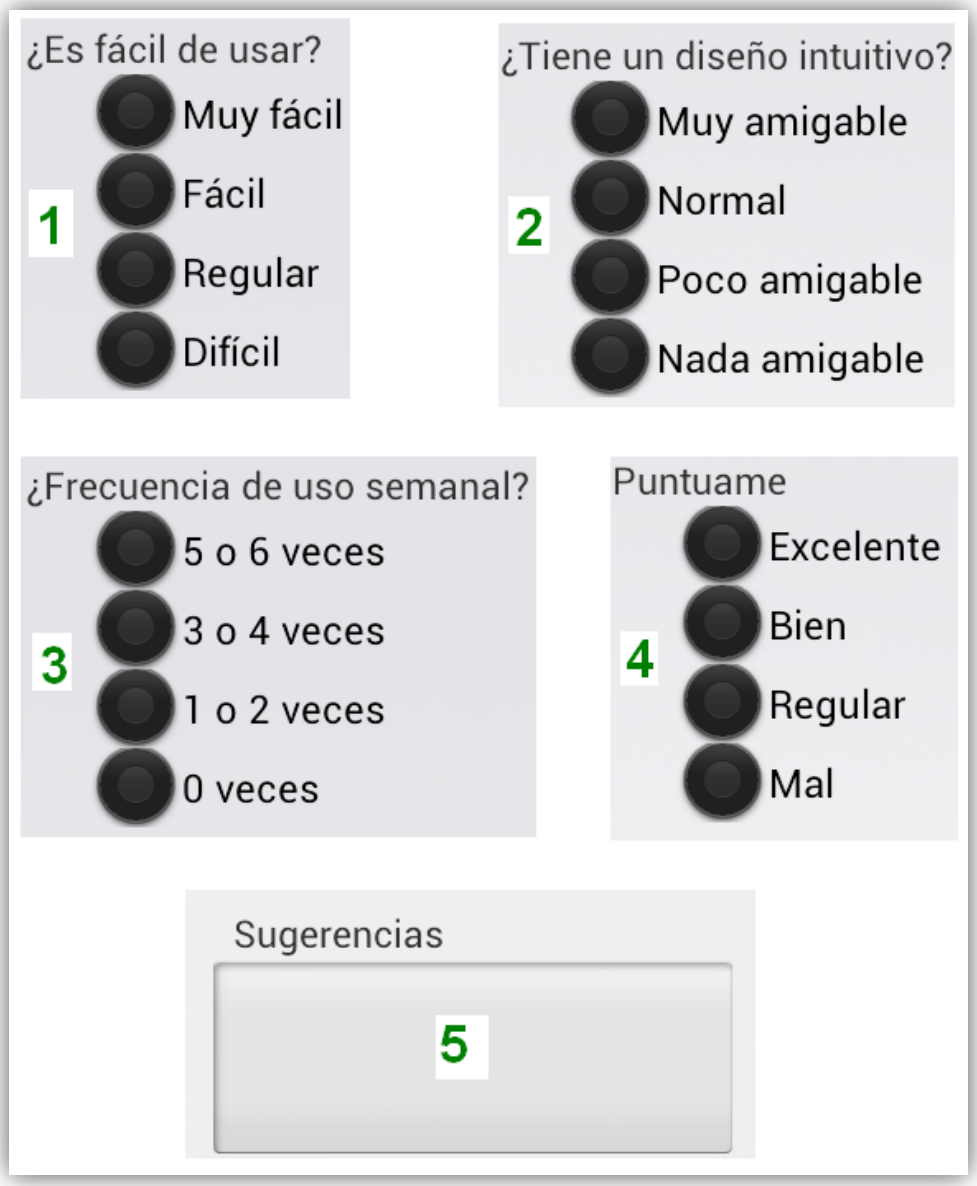



Ilustración 196 - Preguntas del cuestionario

La clase que trabaja con esta utilidad se llama **SondeoSatisfaccion.java** y su correspondiente *layout* que es **sondeo.xml**.

Para esta encuesta, comprobamos antes que nada que tenemos derecho a conectarnos y a utilizar las herramientas de navegación por internet, para ello lo que hacemos es habilitar el contenido de estas con el código que mostraré a continuación. El cual comprueba si la versión del software del dispositivo móvil es de los más recientes, ya que para las nuevas versiones hay que habilitar esta opción que esta anulada por defecto.

```
// Habilitar el trafico en RED o Internet
if (android.os.Build.VERSION.SDK_INT > 9) {
    StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
            .permitAll().build();
    StrictMode.setThreadPolicy(policy);
}
```

Ilustración 197 - Habilitar la conexión a internet para un software superior al api 9

Una vez que nos aseguramos de que podemos conectarnos a internet, también hay que comprobar que tenemos acceso a la red, ya sea por WIFI o tarifa de datos, para ello usamos un método que he creado, que se llama "**hayInternet**()", el cual me devuelve un booleano igual **True** si tenemos internet, o **False** en su caso contrario.

```
// Comprueba si el usuario tiene la Wi-fi o la red de datos habilitada
public static boolean hayInternet(Context c) {
    Context context = c.getApplicationContext();
    ConnectivityManager connectMgr = (ConnectivityManager) context
            .getSystemService(Context.CONNECTIVITY_SERVICE);
    if (connectMgr != null) {
        NetworkInfo[] netInfo = connectMgr.getAllNetworkInfo();
        if (netInfo != null) {
            for (NetworkInfo net : netInfo) {
                if (net.getState() == NetworkInfo.State.CONNECTED) {
                    return true;
                }
            }
        }
    } else {
        // Log.d("NETWORK", "No network available");
    }
    return false;
}
```

Ilustración 198 - Método que comprueba si tenemos conexión a internet

Las preguntas están definidas de tipo etiqueta, a ellas les tengo asociado un grupo de ***RadioButtons***. Estos me sirven para capturar solamente el elemento seleccionado por cada pregunta. Para ello miramos cual ha sido pulsado y lo guardamos en una variable de tipo *String*. Este proceso es igual para las siguientes tres preguntas, solo cambia la precedencia de la pregunta (*RadioButtons*).

```
RadioGroup pre1 = (RadioGroup) findViewById(R.id.pregunta1);
int seleccion1 = pre1.getCheckedRadioButtonId();
switch (seleccion1) {
case R.id.radio1P1:
    opcion1 = Preg1_resp1;
    break;
case R.id.radio1P2:
    opcion1 = Preg1_resp2;
    break;
case R.id.radio1P3:
    opcion1 = Preg1_resp3;
    break;
case R.id.radio1P4:
    opcion1 = Preg1_resp4;
    break;
default:
    marcado1 = false;
    break;

}
```

Ilustración 199 - Ejemplo de cómo capturar la opción seleccionada para cada pregunta

Si no se ha contestado a la pregunta, ponemos a false la variable de tipo Booleana (**marcado1...4**), el cual me sirve como ***flags***, para saber que todas las cuestiones han sido tratadas. Después de comprobar que el formulario ha sido rellenado por completo, procedemos a ejecutar el hilo que envía por internet este formulario.

```
// Si el formulario esta todo completo, Lo enviamos
if (marcado1 && marcado2 && marcado3 && marcado4) {

    new MiTarea()
            .execute("http://www.rapps.hol.es/encuesta.php");
} else {

    /***** Seleccion del Idioma del telefono ******/
    String idioma1 = getResources().getConfiguration().locale
            .getDisplayName().trim();
    if (idioma1.equals("español (España)")) {
        Funciones.imprimir(v.getContext(),
                "Por favor, rellene todo el formulario...");
    } else {
        Funciones.imprimir(v.getContext(),
                "Please fill out the form completely...");
    }
```

Ilustración 200 - Procedemos a ejecutar el hilo que se encarga de enviar la encuesta

El hilo que me ejecuta esto se llama **MiTarea** (que es una clase que extiende de AsinkTask), tal cual como muestro en la imagen anterior, este hilo se ha creado porque al enviar datos a través de la red, estos suelen tardar un cierto tiempo, y varía de la conexión de internet de cada usuario, así que si no hiciera esto, es posible que al enviar la encuesta, esto proceso tarde mucho y pueda dar malas sensaciones a los usuarios por lentitud.

A este hilo le pasamos una *URL* (http://www.rapps.hol.es/encuesta.php), el cual he creado en un servidor gratuito que se llama Hostinger, esta *URL* tiene solo contenido *PHP*, ya que lo único que necesito es pasar datos vía ***POST***, para que luego sean insertados en la BD del Servidor *PHP* con *MySQL*.

Más adelante, explicaré como está desarrollado la parte del Servidor externo, el cual me sirve para hacer una pequeña encuesta.

Para enviar los datos, me apoyo de una función que se llama **postData**(), Este método me permite poner los datos en la red como si se tratara de un envió tipo *POST*. A este método solo hay que pasarle la *URL*, al cual queremos hacer la conexión, luego habrá que poner los datos a enviar, con sus correspondientes variables asociadas a ellos, después se hace la conexión y se envían los datos, en donde la página PHP se encargara de recoger estos datos e insertarlos en la BD creada. El siguiente código muestra este desarrollo.

```
public void postData(String url) {
    // Create a new HttpClient and Post Header
    // long tiempoInicio = System.currentTimeMillis();

    InputStream is = null;
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost(url.trim());

    try {
        // Add your data
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);

        // Variables tipo POST
        nameValuePairs.add(new BasicNameValuePair("SEGURIDAD", "true"));
        nameValuePairs.add(new BasicNameValuePair("pregunta1", opcion1));
        nameValuePairs.add(new BasicNameValuePair("pregunta2", opcion2));
        nameValuePairs.add(new BasicNameValuePair("pregunta3", opcion3));
        nameValuePairs.add(new BasicNameValuePair("pregunta4", opcion4));
        nameValuePairs.add(new BasicNameValuePair("pregunta5", sugerencias));

        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        // Execute HTTP Post Request
        HttpResponse response = httpclient.execute(httppost);

        HttpEntity entity = response.getEntity();
        is = entity.getContent();
        getRespuestaHtml(is);

        Log.d("Http Response: -----> ", response.toString());
```

**Ilustración 201 - Método encargado de enviar los datos al servidor externo**

Para hacer el proceso de creación de hilos, me apoyo de una clase que se llama **MiTarea**, el cual me crea un hilo independiente de la ejecución normal del código. En este código llamo a la función **postData**() con su correspondiente parámetro, el cual es la *URL* y lo ejecuto.

Otra intencionalidad para lo que sirve hacer el hilo con ***AsyncTask***, es que me puedo permitir crear un menú de progreso, el cual me permitirá ver lo que le queda para completar en él envió del formulario. Esto es necesario, ya que al tardar un poco este proceso, siempre viene bien mostrar cuanto le queda por terminar a la hora de hacer esta operación. La siguiente imagen muestra como seria esta barra de progreso.

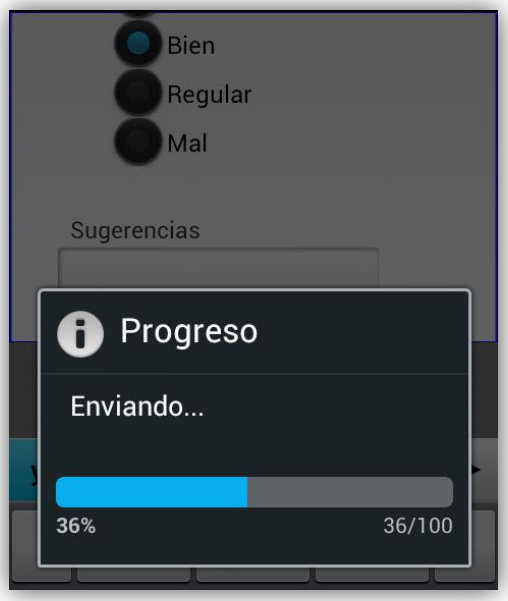



Ilustración 202 - Funcionalidad del proceso de envío de datos

Ahora bien, para poder hacer una barra de progreso, es necesario usar la Clase ProgressDialog, el cual me permite crear esta funcionalidad.

El tiempo que tarda en enviar el formulario al servidor externo es alrededor de 886 Milisegundos, así que usamos la Cuarta parte de este tiempo para simular él envió, ya que al tratarse de un hilo, este proceso se ejecutara en otro proceso independiente de donde se encuentra. Para ello se deben implementar tres clases propias del AsyncTask, como lo son: ***doInBackground, onPreExecute, onProgressUpdate*** y ***onPostExecute.***

Método ***doInBackground***, a este le pasamos la *URL* en un *Array* de *String*, es por eso que este valor lo capturamos en *urls*[0] y se lo pasamos a la función **postData**().

Luego utilizando un método (***publishProgress***) propio de la clase *AsyncTask*, le pasamos los valores que tiene que ir actualizando para luego verse reflejado en la barra de progreso.

```
@Override
protected Integer doInBackground(String... urls) {

    // 886 Milisegundos Es lo que tarda en enviar la Encuesta mas o
    // menos
    // Usamos la 4 parte de ese tiempo para simular el envio de datos
    Integer result = 250;

    for (int i = 0; i < result; i++) {

        if (i == 1) {
            /** Funcion Envio POST **/
            postData(urls[0]);
        }

        // Simulamos cierto retraso
        publishProgress(i / 250f); // Actualizamos los valores
    }
    return result;
}
```

Ilustración 203 - Método doInBackground

Los otros dos métodos, ***onPreExecute*** y ***onProgressUpdate***, sirven para mostrar el menú de dialogo y el otro para ir actualizando este menú respectivamente a medida que el contador se va incrementando.

```
@Override
protected void onPreExecute() {
    dialog.setProgress(0);
    dialog.setMax(100);
    // Mostramos el diálogo antes de comenzar
    dialog.show();
}

// Mostramos el progreso del envio de datos, la barra se Movera
@Override
protected void onProgressUpdate(Float... valores) {
    int p = Math.round(100 * valores[0]);
    dialog.setProgress(p);
}
```

Ilustración 204 - Métodos onPreExecute y onProgressUpdate

Por último, queda mostrar por finalizado el envío a través de un dialogo. Esto lo hacemos con el método ***onPostExecute***, que ayuda a determinar cuando la simulación de progreso ha terminado.

Se ve al final de este código, que hay una variable global que se llama éxito, esta variable me sirve para saber si se ha enviado la encuesta, ya que una vez ha sido enviado con éxito, debo cerrar esta opción para siempre y dar paso a otra herramienta que doy como premio por hacer esta encuesta, la cual se llama IMC (Índice de Masa Corporal).

```
// Finalizamos el dialogo de envio de Datos
@Override
protected void onPostExecute(Integer bytes) {
    dialog.dismiss();

    /* Identificamos el Idioma */
    String idioma = getResources().getConfiguration().locale
            .getDisplayName().trim();
    if (idioma.equals("español (España)")) {
        Toast.makeText(context, "Encuesta enviada con Exito !!",
                Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(context, "Survey sent successfully...",
                Toast.LENGTH_SHORT).show();
    }

    exito = true;

}
```

Ilustración 205 - Método onPostExecute

Una vez puesta esta variable a True, procedo a salir de la aplicación con la invocación del botón retroceso de nuestro móvil, para ello, lo que hago es saber cuándo se ha pulsado, y esto lo hago sobrescribiendo el método abstracto ***onKeyDown***(), el cual me reconoce esta acción. Una vez allí dentro, lo siguiente que tengo que hacer, es poner una variable global estática que se llama enviado igual a true, siendo esta la principal, ya que esta variable se ve reflejada en toda la App. Así que comprobando esta variable procedo a habilitar o deshabilitar los botones de Encueta y de IMC respectivamente.

El siguiente código muestra como se hizo este procedimiento.

```
/**
 * Metodo que reconoce cuando se preciona el boton atras y carga la variable
 * global de enviado a TRUE. Tambien sirve para actualizar o refrescar la
 * Activity Main, Ya que si no, el boton no se ocultaba a la primera
 * aparicion y habia que ir de una actividad a otra para ver reflejados los
 * cambios
 */
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK)) {
        Intent i = new Intent(this, MainMenuInicio.class);
        i.setAction(Intent.ACTION_MAIN);
        i.addCategory(Intent.CATEGORY_HOME);
        i.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

        // Si se ha enviado bn la encuesta, a la hora de ir hacia atras,
        // este ocultara el boton
        if (exito) {
            EncuestaEnviada.enviado = true;
        }

        startActivity(i);
        // android.os.Process.killProcess(android.os.Process.myPid());
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

Ilustración 206 - método onKeyDown

### 4.2.12. Botón IMC (Índice de Masa Corporal)

Este botón se hizo con la intención de otorgar un premio a los usuarios de la aplicación "Workout Routines", ya que si estos envían la encuesta de satisfacción, esta herramienta aparecerá y podrá ser utilizada para que los deportistas controlen su peso, y tengan en cuenta este dato a la hora de planificar sus rutinas.

El diseño de esta opción es muy simple, ya que la organización mundial de la salud tiene una fórmula para calcular este dato. El cuál es el peso (kg) dividió por la altura (metros). Este índice luego se compara con una tabla que ellos ofrecen para mirar el grado de salud en el que se encuentra cada persona.

$$\text{IMC} = \frac{peso(kg)}{altura^2(m)}$$

Ilustración 207 - Formula IMC

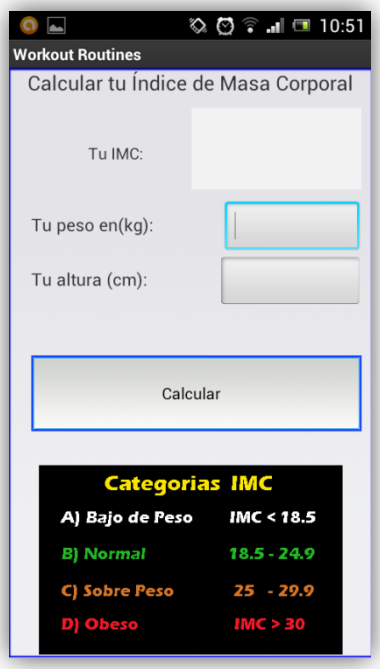



Ilustración 208 - Diseño del cálculo del Índice de Masa Corporal

En la imagen anterior se puede apreciar este diseño, el cual está compuesta por dos campos que recogen los datos y un botón llamado calcular, donde su función es producir una salida con el IMC, una vez hecho este cálculo, los usuarios pueden comparar este dato con la tabla de Categorías del IMC.

La clase que hace todo esto se llama **CalculadoraBMI.java** y su *layout* es calculadora.xml.

La siguiente imagen muestra cómo hacemos este proceso, el cual recogemos los datos, los pasamos a enteros para poder trabajar con ellos, pasamos los centímetros a metros y procedemos a hacer la fórmula del IMC y mostrarlo por la pantalla, además que le damos formato de dos decimales con la ayuda de la clase ***DecimalFormat***.

```
// Boton para computar el Peso Ideal
Button computar = (Button) findViewById(R.id.computar);
computar.setOnClickListener(new View.OnClickListener() {

    EditText pesobmi = (EditText) findViewById(R.id.peso_bmi);
    EditText alturabmi = (EditText) findViewById(R.id.altura_bmi);

    public void onClick(View v) {

        double peso, altura2;

        /***** Seleccion del Idioma del telefono ******/
        String idioma = getResources().getConfiguration().locale
                .getDisplayName().trim();
        if (idioma.equals("español (España)")) {
            peso = Integer.parseInt(pesobmi.getText().toString());
            altura2 = Integer.parseInt(alturabmi.getText().toString())
                    * Integer.parseInt(alturabmi.getText().toString());
            altura2 = altura2 / 100; // Para pasar de cm a metros
            IMC = (peso / altura2) * 100; // Formula del peso ideal
        } else {
            peso = Integer.parseInt(pesobmi.getText().toString()) * 703;
            altura2 = Integer.parseInt(alturabmi.getText().toString())
                    * Integer.parseInt(alturabmi.getText().toString());
            IMC = (peso / altura2); // Formula del peso ideal
        }

        DecimalFormat df = new DecimalFormat("##.##");

        TextView resultado = (TextView) findViewById(R.id.salida_bmi);
        resultado.setText(df.format(IMC));
    }

});
```

Ilustración 209 - Código para calcular el IMC

### *4.2.13. Servidor Externo en Hostinger*

Para hacer la encuesta, era necesario almacenar estos valores en una base de datos, el cual sea accesible para los usuarios y para mí, ya que dependía de ellos para hacer un análisis de calidad y satisfacción.

Así que, se decidió usar un servidor gratuito para este proyecto, por lo que lo único que quería mostrar, era una página web estática, donde reflejara lo más significativo de esta aplicación, así que para ello, se creó una cuenta en esta empresa.

El nombre de la página del proyecto es http://workoutroutines.hol.es/ donde he utilizado una plantilla web y la he restructurado a mis necesidades, claro está, que conociendo un poco de *HTML* y estilos *CSS* es muy fácil hacer estos cambios.

En este servidor *PHP* con *MySQL* he creado una base de datos llamada **u545106689_workout**, siendo este nombre el aconsejable por esta empresa.

Luego he creado una tabla llamada "**cuestionario**", el cual está compuesto por cinco columnas, las cuales corresponden con las cuatro preguntas enviadas por el formulario de mi aplicación y la Quinta hace referencia al cuadro de texto donde los usuarios podrán escribir propuestas o comentarios constructivos.

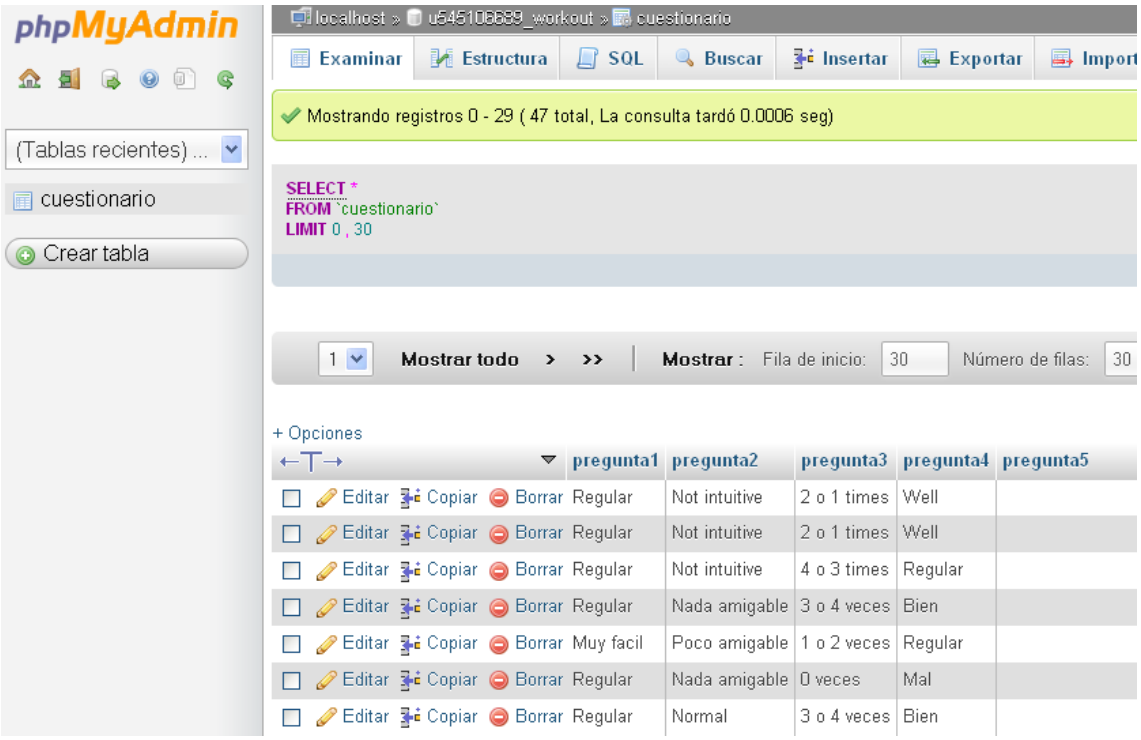



Ilustración 210 - phpMyAdmin del servidor externo

#### 4.2.13.1. Ficheros De Conexión A La BD Externa

**Fichero conectar.php**

```php
<?php
  $server = 'mysql.hostinger.es';
  $user = 'u545106689_gym';
  $pass = '715m631';
  $database = 'u545106689_workout';

  $conexion = mysql_connect($server,$user,$pass)
              or die ('Fallo al conectar con la base de datos');
  mysql_select_db($database) or die ('Error al seleccionar BDD test');
?>
```

Ilustración 211 - Código PHP para la conexión al servidor Externo

En este fichero especificamos el servidor al cual queremos conectarnos, pasando el **usuario**, el **password** y el nombre de la BD a usar, estos datos han sido proporcionados por Hostinger para su correcta utilización.

Este fichero ayuda a conectarnos a la BD sin problemas, para luego hacer todas las operaciones que queramos sobre ella.

**Fichero funciones.php**

La función de este fichero es encargarse de escapar caracteres extraños y de otras cosas, ya que hay gente muy capaz de usar contenido programable para hacer daño a las bases de datos (Inyección SQL). Para ello, utilizo la función de escapar caracteres extraños.

En este fichero, me creo una función que se llama insertar, el cual tiene cinco parámetros, que son las preguntas y el cuadro de texto. Esta función devuelve True si se ha insertado bien y False cuando no.

Las inserciones las hago sobre la tabla "**cuestionario**" anteriormente especificada.

Además, tengo otra función que se llama privilegios, el cual verifica que la información que ha sido mandada vía *POST*, ha sido enviada desde un móvil y no desde una aplicación web. Es por eso, que cuando alguien intenta acceder a esta ruta de conexión mediante una página web, se le redirige a la página principal inmediatamente.

La siguiente imagen muestra lo anteriormente descrito.

```php
<?php
    function no_injection($variable)
    {
        $variable = htmlspecialchars(trim($variable));
        return $variable;
    }

    function insertar($pregunta1, $pregunta2, $pregunta3, $pregunta4, $pregunta5)
    {
        // Esta funcion inserta en la BD los datos de la aplicacion

        $c1 = mysql_real_escape_string(no_injection($pregunta1));
        $c2 = mysql_real_escape_string(no_injection($pregunta2));
        $c3 = mysql_real_escape_string(no_injection($pregunta3));
        $c4 = mysql_real_escape_string(no_injection($pregunta4));
        $c5 = mysql_real_escape_string(no_injection($pregunta5));

        $query = "INSERT INTO cuestionario (pregunta1, pregunta2, pregunta3, pregunta4, pregunta5)
                  VALUES ('$c1','$c2','$c3','$c4','$c5')";
        $result = mysql_query($query) or die("Error insertando comentarios");

        if ($result) // Si se ha insertado bien, devolvemos TRUE
        {
            return true;
        }
        else // si NO
        {
            return false;
        }
        mysql_free_result($result);
    }

    function privilegios ($seguridad)
    {
        if ($seguridad  != "true")
        {
            header("Location: http://workoutroutines.hol.es/");
        }
    }
?>
```

Ilustración 212 - Funciones que ayudan a evitar Inyección SQL

## Fichero encuesta.php

```php
<?php

include 'conectar.php';
include 'funciones.php';

$seguridad = $_REQUEST['SEGURIDAD'];
privilegios($seguridad);

$pregunta1 = $_REQUEST['pregunta1'];
$pregunta2 = $_REQUEST['pregunta2'];
$pregunta3 = $_REQUEST['pregunta3'];
$pregunta4 = $_REQUEST['pregunta4'];
$pregunta5 = $_REQUEST['pregunta5'];

echo "$pregunta1, $pregunta2, $pregunta3, $pregunta4, $pregunta5";
insertar($pregunta1, $pregunta2, $pregunta3, $pregunta4, $pregunta5);

?>
```

Ilustración 213 - Fichero donde enviamos los datos de la encuesta a la BD externa

En el código anterior, lo que hago es llamar a los ficheros de conectar y funciones, para poder acceder a ellas, en este apartado lo que hacemos es recoger las variables con su contenido, y guardarlas en unas variables auxiliares. Eso sí, antes que nada comprobamos que los datos han sido enviados desde mi aplicación y no desde otro lugar.

Después lo que hacemos es llamar al método (insertar) que he creado para insertar en la BD.

### *4.2.14. Opción de Multilenguaje*

Esta utilidad es muy útil, ya que ayuda a que la aplicación pueda estar en dos Idiomas, los cuales son, inglés o español, por defecto la aplicación está en Ingles, pero con la creación de variables que me proporciona *Android*, en formato xml, que se encuentran en los directorios values y values-es, con sus ficheros string.xml, me ayudan a cambiar de idioma, dependiendo de la opción especificada en cada dispositivo.

El directorio values, es para las variables que usan los *Widgets* (*TextView*) con contenido en inglés o en otro idioma que no sea español.

La otra carpeta es values-es, el cual contiene todas las traducciones al español, si el dispositivo se encuentra con este idioma predeterminado.

Lo siguiente, es escribir las mismas variables en los ficheros xml y darle su correspondiente traducción al inglés o español, este proceso es automático y es por eso que es más fácil hacerlo de esta manera.


Ilustración 214 - Directorios responsables de la traducción de un idioma a otro

La siguiente imagen, se muestra como es este proceso. Se puede ver que comparten las mismas variables, pero su contenido o valor es diferente dependiendo del idioma.

Esta funcionalidad la podemos expandir a alemán, francés, etc. Siguiendo los mismos pasos que con el español.

```
<string name="app_name">Workout Routines</string>
<string name="hello_world">Hola Mundo</string>
<string name="menu_settings">Condiguración</string>
<string name="title_activity_main_menu_inicio">Workout Routines</string>

<string name="entrenamientos">Entrenamientos</string>
<string name="progreso">Progreso</string>
<string name="ejercicios">Ejercicios</string>
<string name="exportar">Exportar</string>
<string name="importar">Importar</string>
<string name="imc">IMC</string>
```

Español

```
<string name="app_name">Workout Routines</string>
<string name="hello_world">Hello world!</string>
<string name="menu_settings">Settings</string>

<string name="title_activity_main_menu_inicio">Workout Routines</string>
<string name="entrenamientos">Training</string>
<string name="progreso">Progress</string>
<string name="ejercicios">Exercises</string>
<string name="exportar">Export</string>
<string name="importar">Import</string>
<string name="imc">BMI</string>
```

Inglés

Ilustración 215 - Ejemplo de código XML para el multilenguaje

Para comprobar en qué idioma se encuentra cada teléfono, me valgo de la siguiente instrucción en código *Java*, el cual me permite identificar el idioma del teléfono.

```
/***** Seleccion del Idioma del telefono ******/
String idioma = getResources().getConfiguration().locale
        .getDisplayName().trim();
if (idioma.equals("español (España)")) {
   Español
} else {
   Inglés
}
```

Ilustración 216 - Algoritmo capaz de reconocer el idioma del teléfono

### *4.2.15. Reducción del tamaño de las imágenes.*

Este aparato es muy importante, ya que la decisión que se tomo fue de almacenar las imágenes dentro de la aplicación, es por eso, que era necesario reducir proporcionalmente el tamaño de estas, sin perder calidad. Estas imágenes son la de los entrenamientos a mostrar en la Actividad Ejercicios.

Así que, se usó un programa que se llama ***IrfanView***, siendo este capaz de reducir el tamaño de lo que pesan las imágenes, sin perder calidad.


Ilustración 217 - Logo de IrfanView

Al usar esta aplicación, se pasó de tener **2.47 Mb** del total de las imágenes a que solamente pesaran la mitad, unos **1.06 Mb**. Esta reducción es mucho y ayuda a que la aplicación no ocupe tanto y sea bien apreciada por los usuarios que suelen tener poca memoria en sus móviles.

### *4.2.16. Resultados Encuesta de Satisfacción de la app "Workout Routines"*

Estos datos son recogidos de nuestro Servido Externo, se ha visto que en la encuesta han participado personas que no son de habla hispana, aunque para la construcción de estos datos, los voy a tratar como si hubiera sido en castellano.
En total se cuentan que son más de 800 personas las que se descargaron esta *App*, pero que solo 41 personas han querido hacer la encuesta.

**Servidor:** localhost
**Base de datos:** u545106689_workout
**Tiempo de generación:** 19-03-2013 a las 11:21:31
**Generado por:** phpMyAdmin 3.5.2.2 / MySQL 5.1.66
**consulta SQL:** SELECT * FROM `cuestionario`;
**Filas:** 41

| pregunta1 | pregunta2 | pregunta3 | pregunta4 | pregunta5 |
|---|---|---|---|---|
| Regular | Not intuitive | 2 o 1 times | Well | |
| Regular | Not intuitive | 2 o 1 times | Well | |
| Regular | Not intuitive | 4 o 3 times | Regular | |
| Regular | Nada amigable | 3 o 4 veces | Bien | |
| Muy facil | Poco amigable | 1 o 2 veces | Regular | |
| Regular | Nada amigable | 0 veces | Mal | |
| Regular | Normal | 3 o 4 veces | Bien | |
| Facil | Normal | 3 o 4 veces | Bien | poder incluir tus propias rutinas, eligiendo los m... |
| Regular | Normal | 1 o 2 veces | Excelente | |
| Easy | Normal | 4 o 3 times | Well | |
| Easy | Normal | 5 o 6 times | Well | |
| Difficult | Unintuitive | 0 times | Bad | |
| Regular | Normal | 1 o 2 veces | Regular | linkear tus propias tablas con el progreso |
| Facil | Normal | 3 o 4 veces | Regular | falta poder ver el dibujo de los ejercicios al pin... |
| Facil | Normal | 3 o 4 veces | Regular | mas ejercicios |
| Facil | Muy amigable | 3 o 4 veces | Bien | |
| Muy facil | Muy amigable | 5 o 6 veces | Excelente | |
| Regular | Very intuitive | 4 o 3 times | Excellent | |
| Facil | Muy amigable | 3 o 4 veces | Bien | |
| Regular | Nada amigable | 1 o 2 veces | Mal | cambiar la fotma |
| Easy | Not intuitive | 5 o 6 times | Well | |

Ilustración 218 - Datos consultados sobre la encuesta

| pregunta1 | pregunta2 | pregunta3 | pregunta4 | pregunta5 |
|---|---|---|---|---|
| Easy | Not intuitive | 5 o 6 times | Well | |
| Muy facil | Muy amigable | 5 o 6 veces | Excelente | |
| Regular | Poco amigable | 5 o 6 veces | Regular | |
| Regular | Poco amigable | 3 o 4 veces | Bien | |
| Easy | Unintuitive | 0 times | Bad | |
| Easy | Not intuitive | 2 o 1 times | Regular | Mejorar diseno y dietas para ver |
| Muy facil | Normal | 3 o 4 veces | Bien | estar |
| Dificil | Normal | 5 o 6 veces | Mal | as |
| Muy facil | Muy amigable | 1 o 2 veces | Bien | |
| Difficult | Unintuitive | 0 times | Bad | la usaria si tuviera ejercisios de resistencia com... |
| Muy facil | Muy amigable | 5 o 6 veces | Excelente | |
| Dificil | Nada amigable | 1 o 2 veces | Mal | |
| Regular | Muy amigable | 1 o 2 veces | Bien | tendrias de poner si quieren ganar masa muscular o... |
| Muy facil | Normal | 3 o 4 veces | Regular | Dejar expandir m |
| Facil | Normal | 1 o 2 veces | Bien | |
| Difficult | Normal | 4 o 3 times | Regular | |
| Easy | Very intuitive | 4 o 3 times | Excellent | |
| Dificil | Normal | 5 o 6 veces | Regular | poned las tablas o rutinas m |
| Easy | Normal | 2 o 1 times | Well | |
| Dificil | Normal | 5 o 6 veces | Mal | as |

Ilustración 219 - Más datos sobre la encuesta

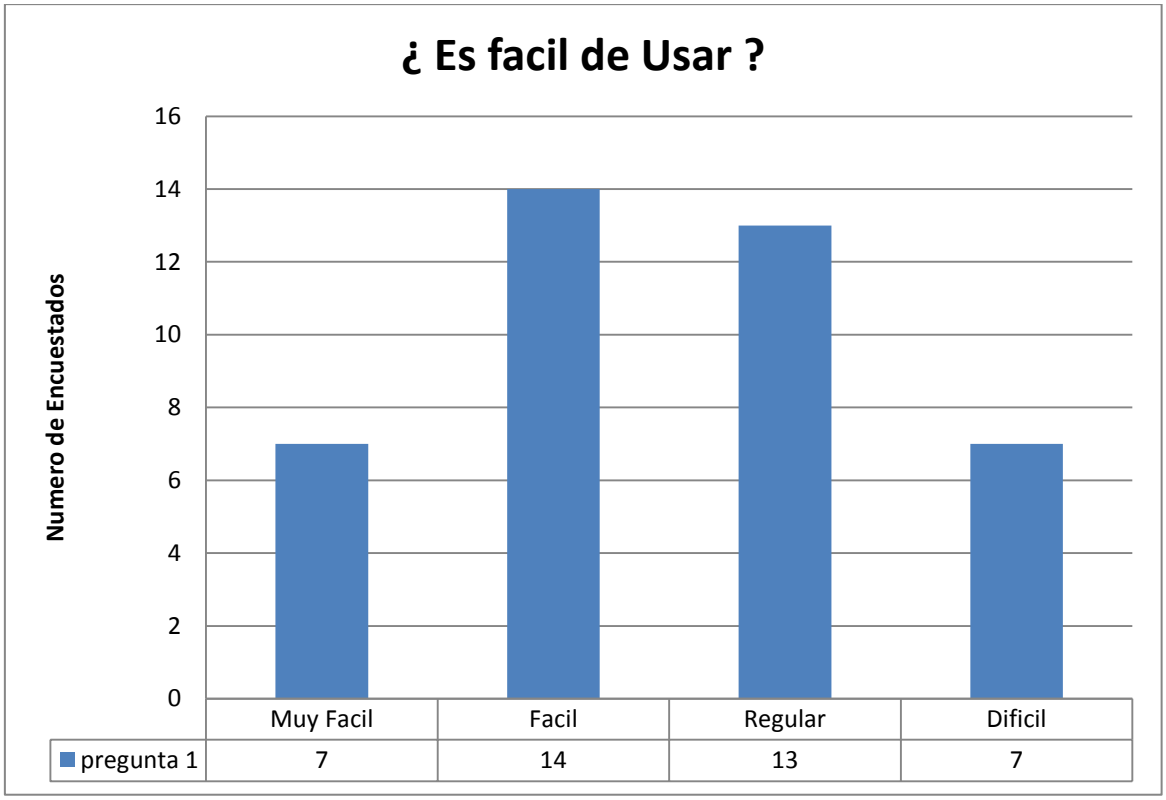
Estos han sido estudiados para dar futuras mejoras, ya que la opinión de los usuarios es muy importante, porque la finalidad de todo proyecto es complacer a la mayoría de estos.

Es por eso, que en unas futuras versiones de "**Workout Routines**" tomare en cuenta estas recomendaciones o al menos parte de ellas.

El total de encuestados hasta la fecha (19-03-2013) son de 41 personas.

Los resultados de la encuesta son las siguientes.

#### 4.2.16.1. Pregunta 1 ¿Es fácil de usar?



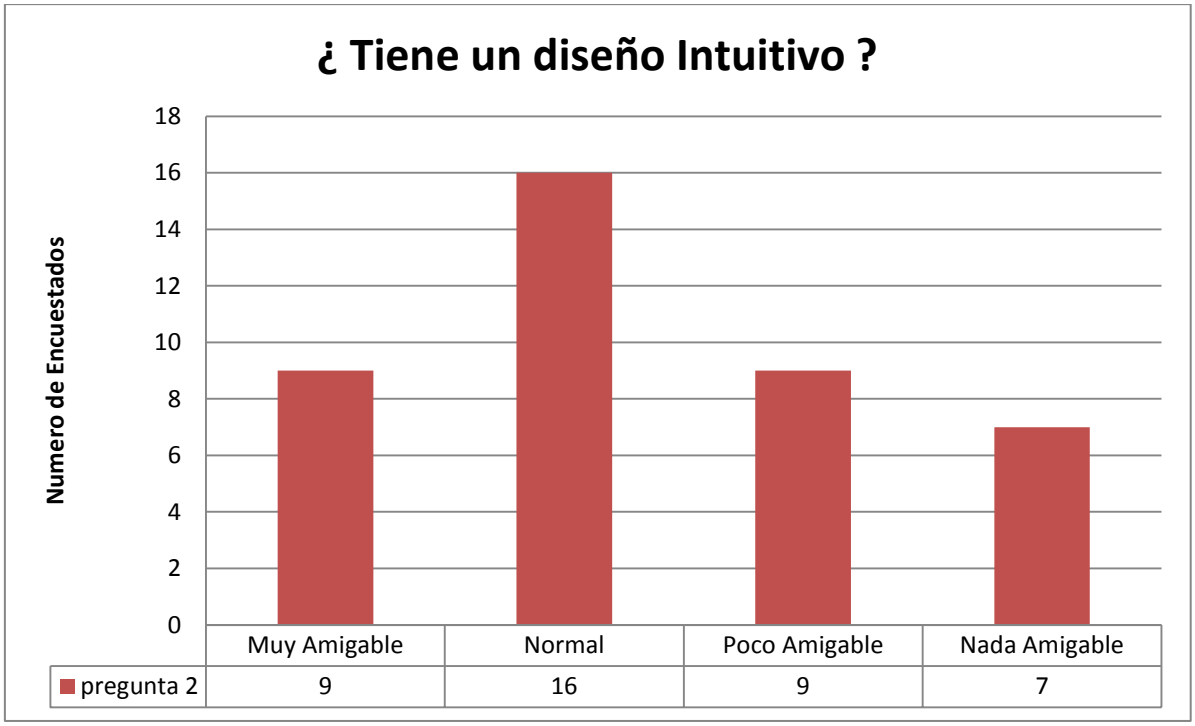

Según el informe de esta gráfica, se puede decir que a la mitad de los encuestados les parece relativamente fácil usar esta aplicación, mientras que a la otra, no tanto. Podría decir con esto, que la aplicación tiene una usabilidad no muy clara, aunque la idea del proyecto desde un principio era que fuera lo más fácil de usar. Es conveniente tener este resultado presente para futuras evoluciones.

| pregunta 1 | |
|---|---|
| Muy Facil | 7 |
| Facil | 14 |
| Regular | 13 |
| Dificil | 7 |
| | |
| Mediana: | 10 |

Ilustración 220 - Resultado de la encuesta, pregunta 1

Para ello calculamos la mediana de los datos que corrobora que la aplicación esta entre fácil y regularmente fácil de usar.

#### 4.2.16.2. Pregunta 2 ¿Tiene un diseño intuitivo?




Se puede decir que la aplicación es intuitiva a la hora de usar, y que cada botón, cada pestaña, cada imagen, etcétera, representa bien su significado y que los deportistas no tendrán pérdidas a la hora de usar esta app.

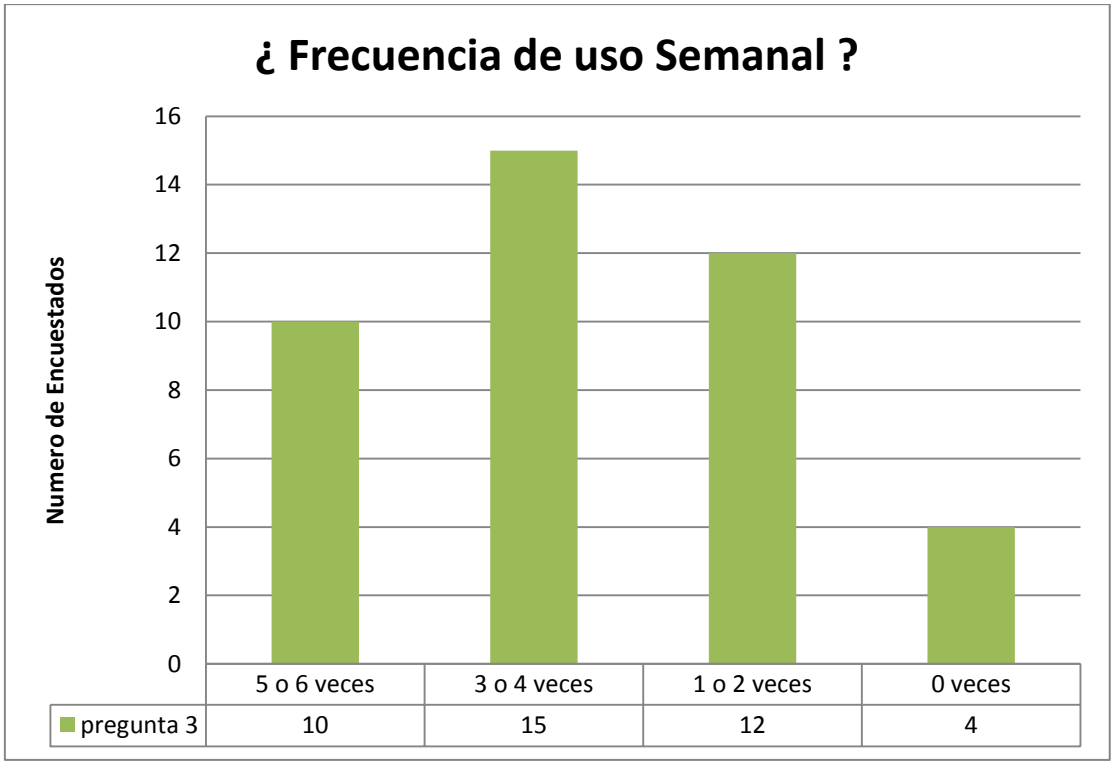
Por otro lado, si comparamos esta pregunta, con la contestada en la cuestión uno, se puede decir, que esta aplicación es fácil e intuitiva de usar por mayoría de votos. Y me veo bastante satisfecho, ya que la finalidad de esta era conseguir estos resultados, claro está, sin olvidar a los que no comparten este resultado.

| pregunta 2 | |
|---|---|
| Muy Amigable | 9 |
| Normal | 16 |
| Poco Amigable | 9 |
| Nada Amigable | 7 |
| | |
| Mediana: | 9 |

Ilustración 221 - Resultado de la encuesta, pregunta 2

La mediana de esta pregunta es 9, este dato significa que lo de ser intuitivo es normal, y que podemos estar seguros, pero habrá que mejorar estos resultados para futuras versiones.

#### 4.2.16.3. Pregunta 3 ¿Frecuencia de uso semanal?




Con la gráfica anterior, se ve que los usuarios pueden usar esta app al menos de 3 a 4 veces a la semana, que más o menos corresponde con lo ideal para las personas que suelen visitar los gimnasios para fortalecer su cuerpo.

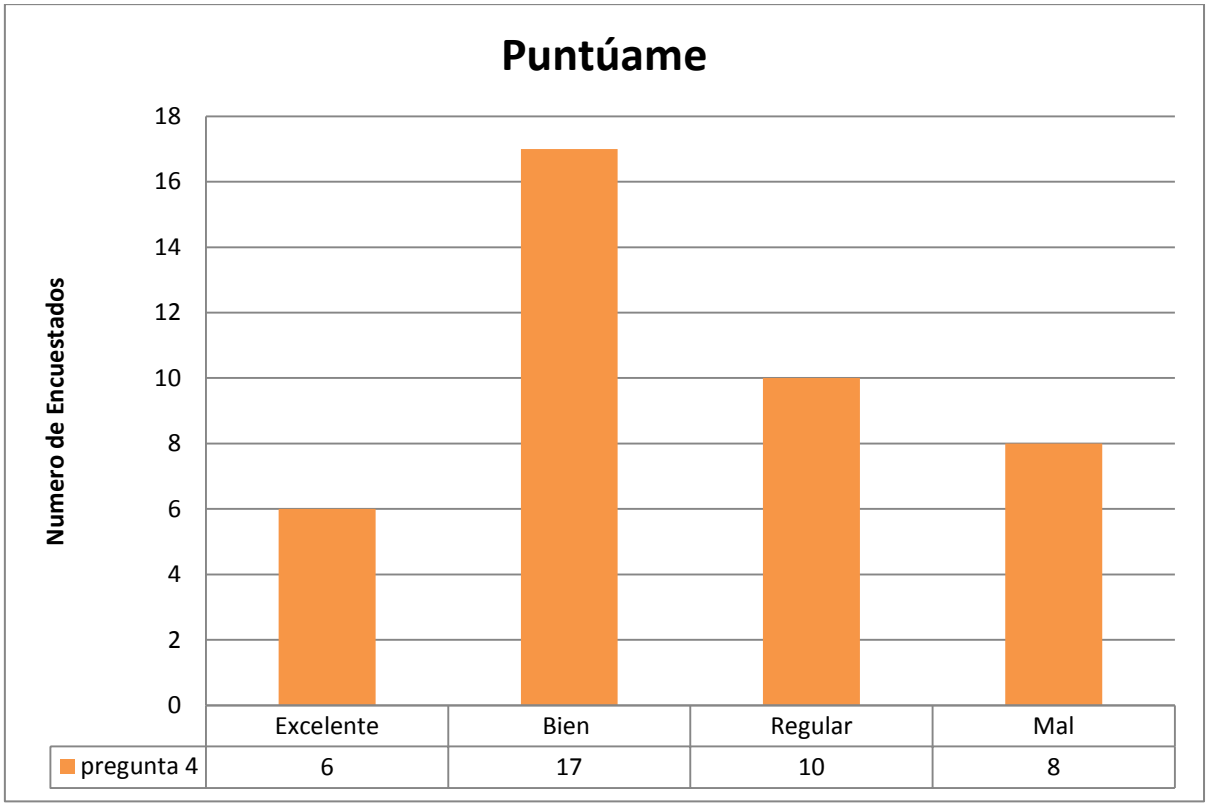
Se ve que, 4 personas contestaron Cero veces, esto puede significar que son personas que de alguna manera pueden ser otros desarrolladores o gente que se encarga de promocionar o criticar de manera constructiva aplicaciones para ponerlas en sus portales *web* o *blogs*.

| pregunta 3 | |
|---|---|
| 5 o 6 veces | 10 |
| 3 o 4 veces | 15 |
| 1 o 2 veces | 12 |
| 0 veces | 4 |
| | |
| Mediana: | 11 |

Ilustración 222 - Resultado de la encuesta, pregunta 3

Los resultados obtenidos de esta pregunta me agradan, ya que significa que usaran esta aplicación más de dos veces por semana y que la llevaran al gimnasio para ponerla a prueba.

#### 4.2.16.4. Pregunta 4 Puntúame




Esta cuestión se hizo con la finalidad de ver el grado de aceptación que podría tener en el mercado de *Android Market*, y por mayoría de votos la aplicación es de buen agrado y pueda que tenga éxito.

Ya que las personas se centran en que está del todo bien la aplicación.

| pregunta 4 | |
|---|---|
| Excelente | 6 |
| Bien | 17 |
| Regular | 10 |
| Mal | 8 |
| | |
| Mediana: | 9 |

Ilustración 223 - Resultado de la encuesta, pregunta 4

#### *4.2.16.5. Pregunta 5 Otros – Recomendaciones*

En las tablas anteriores, he resaltado algunas recomendaciones bastante agradables y que podrían ser de muy buenas ideas para mejorar este proyecto.

Propongo aquí algunas de estas.

1) **poder incluir tus propias rutinas.**
2) **linkear tus propias tablas con el progreso.**
3) **falta poder ver el dibujo de los ejercicios**
4) **más ejercicios.**
5) **Mejorar diseño y dietas.**
6) **tendrías de poner si quieren ganar masa muscular.**

La respuesta 2, de poder linkear las rutinas con las imágenes de cómo se hace, es una de las mejores ideas que creo que me aportaría más a este proyecto, así que, esta sería una de las primeras ideas a mejorar en las futuras evoluciones de este proyecto Android.

Luego se podrían poner más ejercicios o agregar un apartado de dietas.

# 5. PRUEBAS UNITARIAS

Hacer un plan de pruebas unitarias es muy importante, ya que de esta manera, evaluamos si lo que hacemos en nuestras funciones o métodos se hace bien y sin errores. Así que, Eclipse cuenta con un Framework de Junit para hacer pruebas unitarias para código desarrollado en java, pero este plugin no nos sirve, ya que Eclipse usa la estructura Java, pero ciertamente no es java, ya que antes necesita traducir este código a un lenguaje que pueda reconocer Android, así que este método no nos vale. Es por eso, que Android trae su propio test de pruebas, basta con ir a: **File → New → Other → Android → Android Test Project**. Y creamos un proyecto nuevo que nos ayuda a hacer este tipo de pruebas. Damos un nombre al proyecto, asociamos el proyecto al cual queremos hacer las pruebas y finalizar.

Este test es basado en el *framework* de ***Junit***, solo que para *Android*.

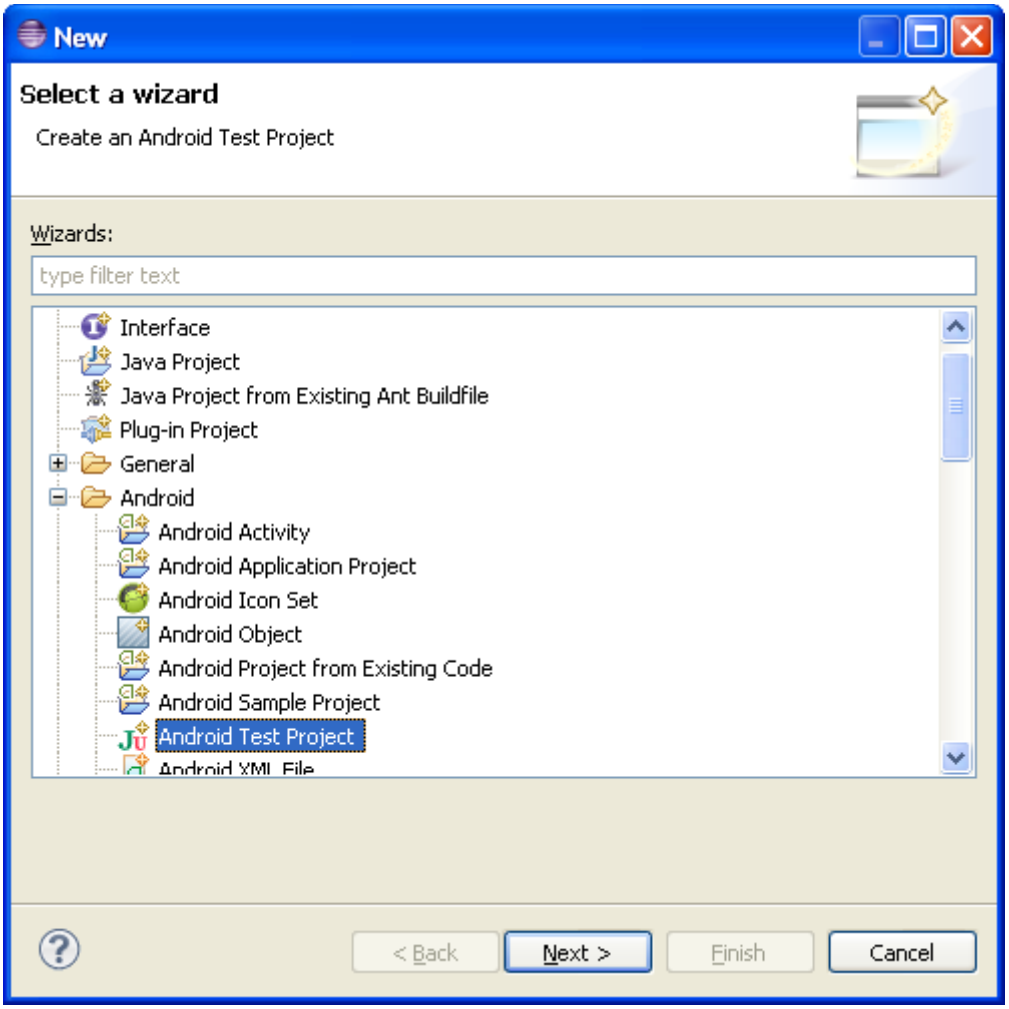



Ilustración 224 - Forma para crear pruebas unitarias en Android

Una vez tenemos creado este proyecto de pruebas, basta con hacer las oportunas operaciones sobre las funciones que queremos comprobar y ver los resultados.

Para este test, hemos usado las funciones de crear, leer y escribir sobre un fichero, ya que estas opciones son importantes, porque son las que destacan a esta aplicación, y por consiguiente, necesito que estas estén en perfecto funcionamiento.

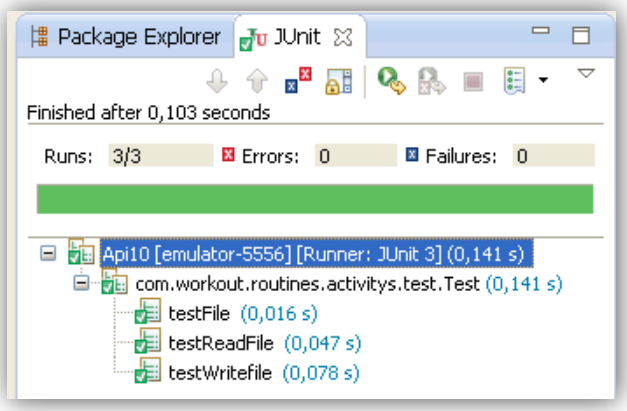



Ilustración 225 - Ejecución del JUnit con los métodos testeados a OK

```
public class Test extends TestCase {

    Context c;
    FileManager test = new FileManager();

    public void testWritefile() {
        // Creamos y escribimos sobre un fichero
        test.writeFile("prueba.txt", "Hola mundo !!");
    }

    public void testFile() {
        // devolvemos la ruta de un fichero
        test.ficheroCSV();
    }

    public void testReadFile() {
        // leemos sobre un fichero
        test.readFile(c, "prueba.txt");
    }

}
```

Ilustración 226 - Métodos probados en el JUnit para ser testeados

Las capturas anteriores, muestran que estas tres funciones trabajan a la perfección, y por tanto no tenemos que preocuparnos de que tengan algún fallo a la hora de trabajar con ellas. ***Junit*** para *Android* nos da el visto bueno.

Se hicieron más pruebas unitarias, pero estas fueron realizadas durante la programación, y se hicieron a través de trazas. Esta forma de hacer pruebas es un poco

antigua, pero es la que más usada durante el desarrollo, dada su sencillez. Así que, Android nos provee una clase que se llama ***Log***, la idea de esta clase es hacer un *System.out* pero más rápido y sin esperar a que la app esté completamente cargada.

Los resultados del log los podemos ver en la ventana ***LogCat*** de *Eclipse*.

Para definir el *Log*, basta con ejecutar la siguiente instrucción:

```
Log.d("TAG", "Mensaje a mostrar");
```

En esta instrucción se puede ver que usa como método estático el d, (Debug), se escogió este, ya que la salida que muestra por pantalla es de color azul, y para las búsquedas es más fácil de identificar y de encontrar posibles fallos esperados o inesperados, pero podríamos utilizar más funciones y que por consiguiente nos proporcione salidas de texto con su color especifico.

Como ocurre con otros frameworks de logging, en Android los mensajes de log se van a clasificar por su criticidad, existiendo así varias categorías (ordenadas de mayor a menor criticidad):

1. Error (Color Rojo)
2. Warning (Color Amarillo)
3. Info (Color Verde)
4. Debug (Color Azul)
5. Verbose (Color Negro)

Para cada uno de estos tipos de mensaje, existe un método estático independiente que permite añadirlo al log de la aplicación. Así, para cada una de las categorías anteriores tenemos disponibles los métodos **e(), w(), i(), d()** y **v()** respectivamente.

Ahora voy a mostrar un ejemplo de pruebas con la clase Log, para ello he utilizado la función **insertar_bd**() que se encuentra dentro de mi Clase **SQLiteBD**, donde sus parámetros son serie, ejercicio, peso, grupo, entrenamiento y pestaña. Siendo todos estos de tipo *String*. Así que, voy a comprobar que cada vez que hacemos una inserción sobre la tabla entrenamientos, esta se ejecuta bien, y que los datos llegan conforme lo he planeado.

La siguiente captura muestra cómo he creado los *Logs* en *Java*, siendo estos iguales en su salida, pero utilizando diferentes métodos, para ver la diferencia entre estos, aunque la única diferencia es en el color de sus textos.

```
Log.e("TAG Error    -> BD", "Mensaje : "+serie+" - "+ejercicio+
        " - "+peso+" - "+grupo+" - "+entrenamiento+" - "+pestaña);

Log.w("TAG Warning -> BD", "Mensaje : "+serie+" - "+ejercicio+
        " - "+peso+" - "+grupo+" - "+entrenamiento+" - "+pestaña);

Log.i("TAG Info     -> BD", "Mensaje : "+serie+" - "+ejercicio+
        " - "+peso+" - "+grupo+" - "+entrenamiento+" - "+pestaña);

Log.d("TAG Debug    -> BD", "Mensaje : "+serie+" - "+ejercicio+
        " - "+peso+" - "+grupo+" - "+entrenamiento+" - "+pestaña);

Log.v("TAG Vervose -> BD", "Mensaje : "+serie+" - "+ejercicio+
        " - "+peso+" - "+grupo+" - "+entrenamiento+" - "+pestaña);
```

Ilustración 227 - Logs para las diferentes salidas

Ahora procedemos a ver su salida por el terminal ***LogCat*** de *Eclipse*, cuando hacemos la operación de insertar. Para ello usamos el siguiente ejemplo, donde insertamos en nuestro terminal y vemos que los datos son insertados correctamente.


Ilustración 228 - Prueba de los elementos pasados al método insertar_bd()

Para corroborar tal información, procedemos a ir al terminal *LogCat* y ver si efectivamente son los datos que hemos enviado a la BD.

La siguiente imagen nos muestra información del tipo de *Log* usado, la fecha en que fue ejecutado, el *PID*, etc. Este ayuda a comprobar que la información es verdadera.

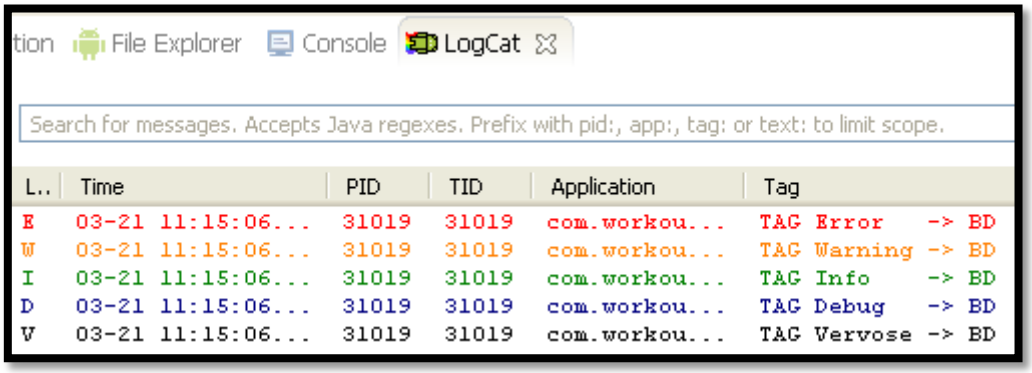



Ilustración 229 - Salidas de la Clase Log con sus distintos métodos

La siguiente captura enseña los datos que hemos querido insertar para este ejemplo, y vemos que corresponde con lo que he querido enseñar a la hora de utilizar la clase Log.


Ilustración 230 - Comprobamos que son los mismos elementos introducidos

# 6. PRUEBAS DE CARGA

Esta aplicación se subió a Android Market, por lo tanto, se tuvo que pagar una cuota de inscripción para poder publicar aplicaciones. Así que una vez subida esta aplicación, *Google Android* nos concede gráficos donde podemos ver la evolución y aceptación de esta aplicación, además, se muestran cuántas descargas ha tenido y cuántos de sus usuarios continúan usando la aplicación en cuestión.

Las descargas en total son de 851 y subiendo, y solo 212 personas continúan usando esta aplicación, lo que significa que la tienen instalada en sus dispositivos móviles. La siguiente imagen muestra tal resultado.

| PRECIO | INSTALACIONES ACTIVAS / TOTALES | VALORACIÓN MEDIA / TOTAL | ERRORES Y ANRS |
|---|---|---|---|
| Gratuita | 212 / 851 | ★ 3,50 / 4 | |

Ilustración 231 - Prueba de carga del servidor de Google Play Developer

La siguiente ilustración muestra las instalaciones activas en dispositivos de Versión de *Android*.

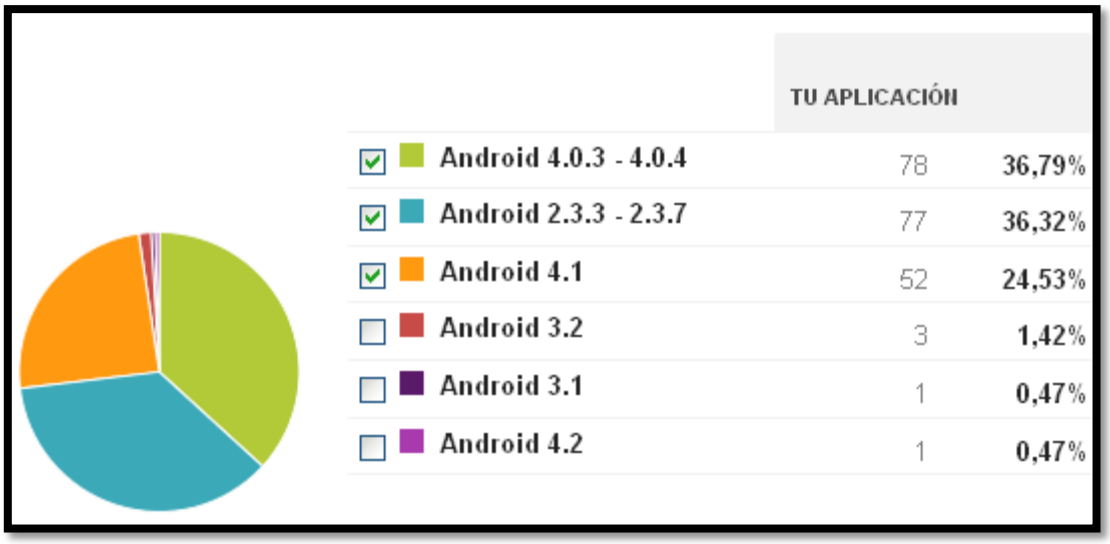



Ilustración 232 - Instalaciones activas en cada plataforma

Por otra parte voy a mostrar las estadísticas de la descargas por país, esto viene bien para saber a las zonas donde hemos llegado y a los cuales debemos prestar más atención.

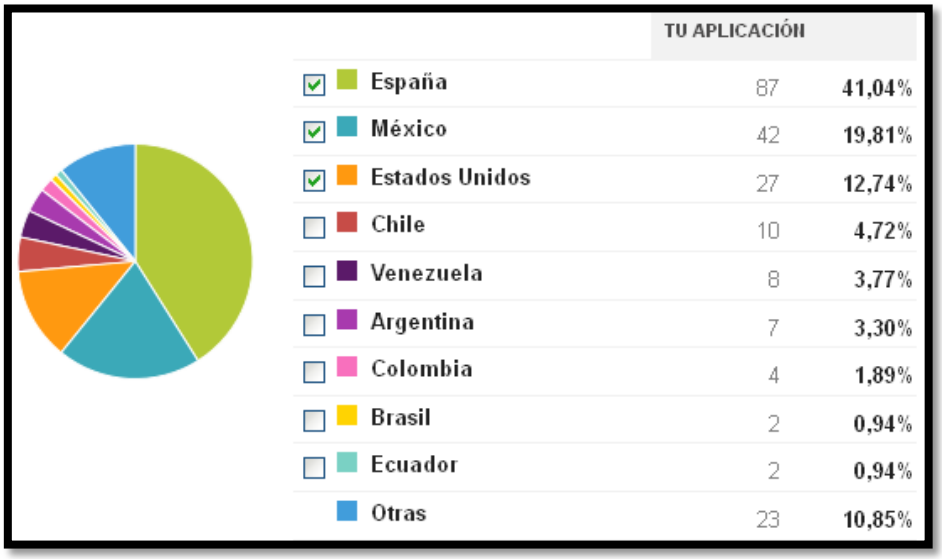



Ilustración 233 - Países donde más han descargado esta App

Se ve que en España, México y poco más de estados unidos, es en donde hemos tenido más aceptación, ya que esta gráfica lo que nos quiere decir es en qué países se ven más interesados por este tipo de aplicación, que corresponde a la categoría de deporte y salud.

Además, *Google* nos recomienda cuáles son los países donde se usan más aplicaciones relacionadas con los deportes. Para ello, la mayor referencia que tenemos es Estados Unidos.


Ilustración 234 -Recomendación de Google sobre demanda de aplicaciones que van de deportes

La siguiente gráfica muestra el idioma en que se usa esta app. Se debe recordar que esta aplicación tiene la compatibilidad de trabajar con dos idiomas, y como se puede ver, el resultado es que los hispanohablantes son los que más usan esta *App*.

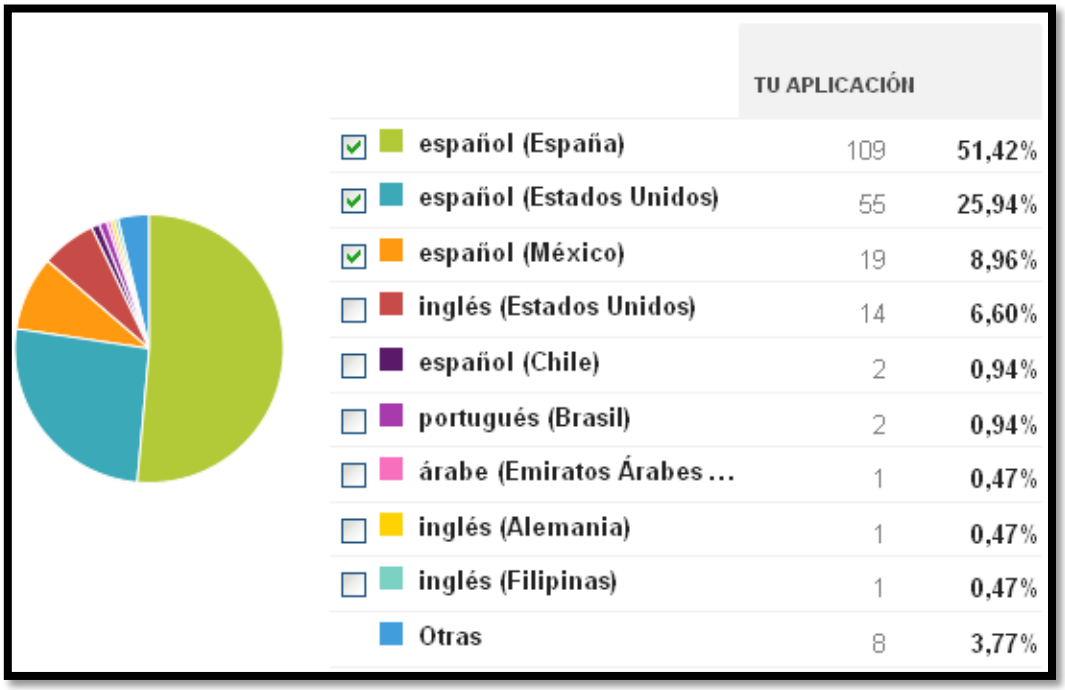



Ilustración 235 - Idiomas en que utilizan esta app

La siguiente gráfica muestra las instalaciones activas en los dispositivos, mostrando la versión que tienen instaladas, y se puede ver, que ha sido a partir de la versión 5, que es en donde prácticamente estaba finalizada esta aplicación, y que ya contaba con algunos cambios y un nuevo diseño y presentación, que es en donde ha tenido mayor aceptación.


Ilustración 236 – La versión de mi App que más se han descargado

## 6.1. Prueba de carga para el servidor externo

Para este apartado se ha considerado no hacer tal prueba de carga, ya que este servidor no es de mi propiedad. Además, las *web host* están preparadas para aceptar múltiples conexiones a la vez y estos son tolerantes a fallos.

Por otra parte, no considero que una multitud de personas puedan llegar a conectarse a mi página web y tampoco que todos estos envíen la encuesta a mi servidor de base de datos *MySQL*.

# 7. PRESUPUESTO

Para este proyecto, he trabajado con los siguientes lenguajes de programación: JAVA, XML, PHP, SQL y HTML, siendo de más usado JAVA y de menos uso HTML.

Por tanto, para este presupuesto voy a calcular las líneas de código totales para el caso de programación Java, ya que para el lenguaje de programación *Android* dependía de una herramienta Pallete Gráfica, la cual me diseñaba todos los XML necesarios para el diseño de la aplicación.

Antes de explicar cuánto costaría el proyecto "Workout Routines", voy a explicar cuánto costaría la estimación de este proyecto, para ello me valgo de los modelos de Punto de Función y Cocomo II en especial, ya que permite estimar el esfuerzo, el costo y la duración.

## 7.1. Calculo De Los Puntos De Función

Para el cálculo de los puntos de función sin ajustar, tenemos que usar la tabla siguiente que los calcula. Aunque los N valores hay que contarlos dependiendo del interés de cada funcionalidad.
Para hacer estos cálculos me valgo un documento en *EXCEL*, que nos dio la profesora de la signatura de **PLANIFICACIÓN DE PROYECTOS INFORMÁTICOS**.

| Tipo de función de usuario | Nivel de complejidad | Nº | * | Peso | = | Total |
|---|---|---|---|---|---|---|
| EI | Baja | 12 | | 3 | | 36 |
| | Media | | | 4 | | |
| | Alta | | | 6 | | |
| EO | Baja | 23 | | 4 | | 92 |
| | Media | | | 5 | | |
| | Alta | | | 7 | | |
| EQ | Baja | 23 | | 3 | | 69 |
| | Media | | | 4 | | |
| | Alta | | | 6 | | |
| ILF | Baja | 4 | | 7 | | 28 |
| | Media | 1 | | 10 | | 10 |
| | Alta | | | 15 | | |
| EIF | Baja | 1 | | 5 | | 5 |
| | Media | | | 7 | | |
| | Alta | | | 10 | | |
| **Número de Puntos Función sin ajustar:** | | | | | | **240** |

| Factores de Complejidad | Grado | |
|---|---|---|
| 1. Comunicaciones de datos | | no |
| 2. Procesamiento distribuido | | no |
| 3. Objetivos de rendimiento | 3 | |
| 4. Configuración de explotación | 3 | |
| 5. Tasas de transacción rápidas | 1 | |
| 6. Entrada de datos en línea | | no |
| 7. Eficiencia para el usuario final | 2 | |
| 8. Actualización de datos en línea | 1 | |
| 9. Procesamiento complejo | 3 | |
| 10. Reusabilidad | 3 | |
| 11. Facilidad de instalación | 1 | |
| 12. Facilidad operacional | | no |
| 13. Instalaciones Múltiples | | no |
| 14. Facilidad de Cambios | 3 | |
| **Total FC** | **20** | |

Total FC = Suma total de todos los factores

PUNTOS FUNCIÓN = PUNTOS FUNCIÓN SIN AJUSTAR * (0'65 + 0'01 * TOTAL FC ) = **204**

NOTA: Rellenar las celdas amarillas

Ilustración 237 - Cálculo de los puntos de función

$$PFA = PFSA * (0{,}65 + 0{,}01 \times \sum_{i=1}^{14} FC_i)$$

Ilustración 238 - Formula de los Puntos de Función

Para el cálculo de los puntos de función sin ajustar, tuve que evaluar y valorar la complejidad de todas las ventanas de la aplicación, para ello tuve que contar los Tipos de datos elementales (DET), los tipos de ficheros referenciados (FTR), los tipos de registro elementales (RET), las entradas (EI), las salidas (EO), las consultas (EQ), los ficheros lógicos internos (ILF) y los ficheros lógicos externos (EIF).

Luego hay que comparar estos valores entre ellos, para que nos dé una complejidad (peso) por cada ventana de la aplicación analizada.

Después sumar este peso con su correspondiente atributo, ya sea un EI, EO, EQ, ILF o un EIF, y ponerlo en la tabla Excel que nos lo calcula automáticamente aplicando la formula antes propuesta. Ilustración 238.

## 7.2. Modelo COCOMO II

Para este desarrollo, debo tener en cuenta los puntos de función sin ajustar (***PFSA***), calculados en el apartado anterior y con valor de 240. Este dato los tenemos que multiplicar por el valor de relación entre el Lenguaje De Implementación De Código Fuente Por PFSA correspondiente con JAVA, que vale 53 y luego dividirlo entre 1000, para que este valor se trasforme en miles de líneas de código (***KSLOC***). Este valor será nuestro tamaño para la fórmula de esfuerzo.

Para la estimación de este proyecto he usado el modelo Postarquitectónico de COCOMO. Para este apartado también me he ayudado de un EXCEL el cual me calcula el esfuerzo con el nivel Postarquitectónico, con tablas que calculan los factores de escala y la estimación del multiplicador M.

Además para el cálculo del coste monetario he tomado en cuenta que un programador trabaja 22 días laborales (**Mes**) a 8€ la hora.

Ahora voy a mostrar tres capturas del Excel, en donde muestro paso a paso como se ha calculado el esfuerzo, el coste por personas mes y el coste del proyecto final.

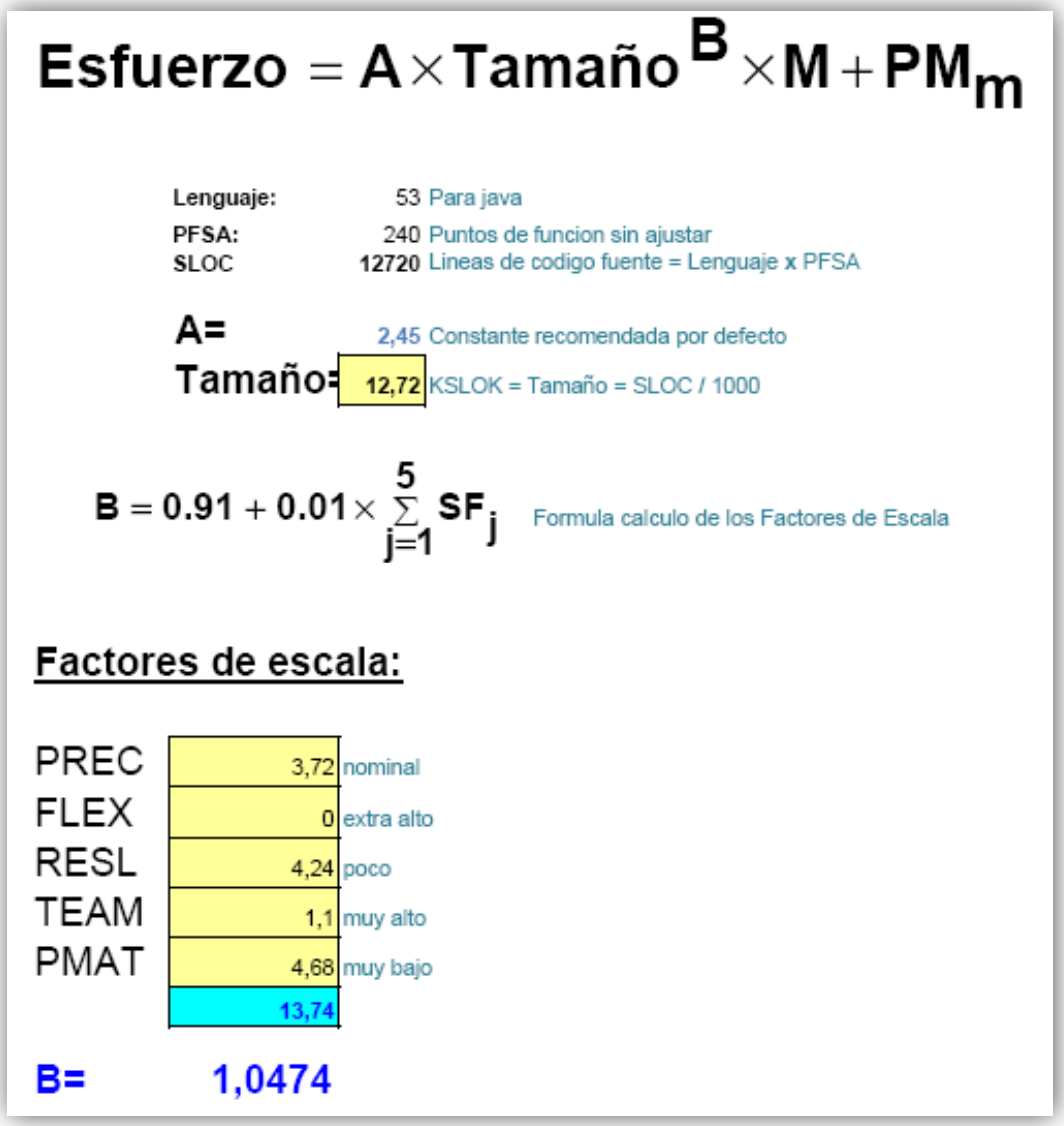
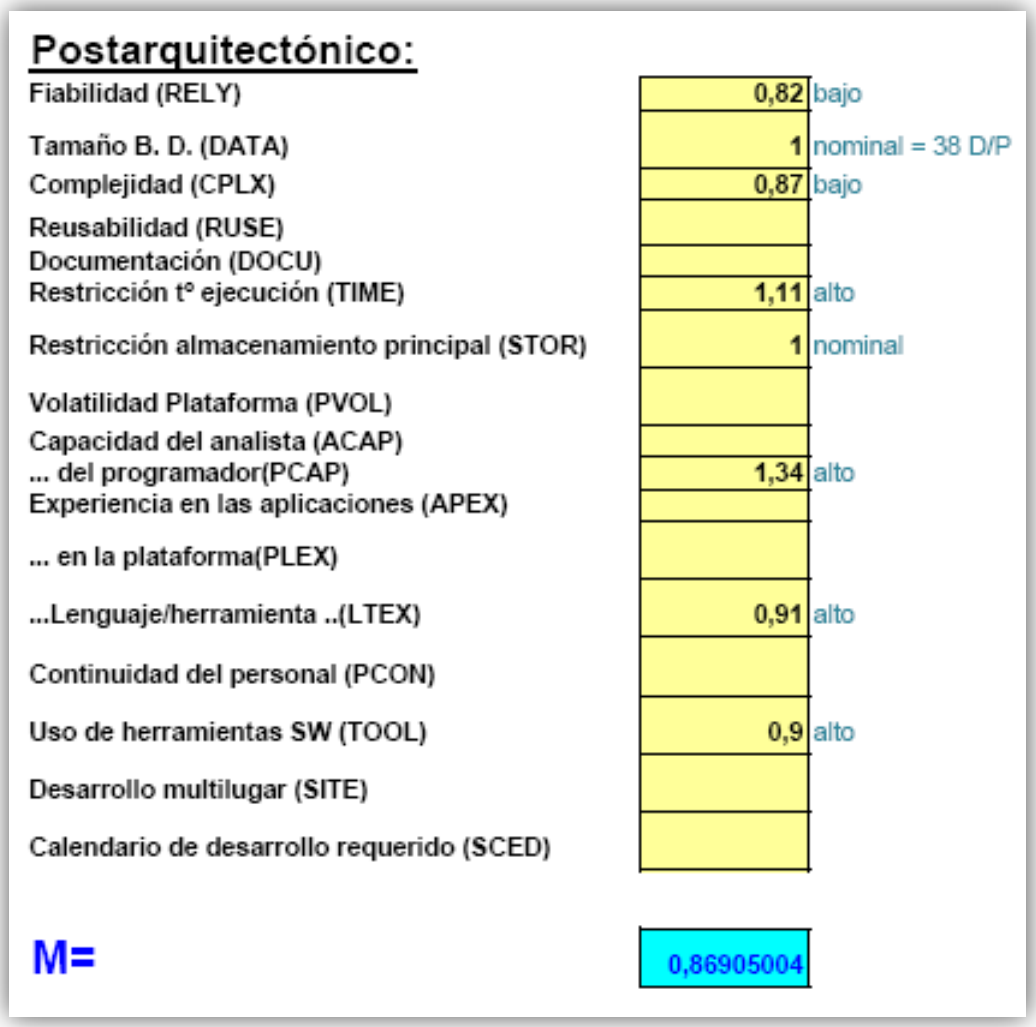



Ilustración 239 - Calculo de los factores de escala

En esta parte lo que intento es calcular el valor de la variable Tamaño.

Además de calcular los factores de escala (**B**), ya que estos valores son necesarios para la fórmula del esfuerzo.

Por otra parte, el valor A es 2.45, ya que es por defecto.



| Postarquitectónico: | | |
|---|---|---|
| Fiabilidad (RELY) | 0,82 | bajo |
| Tamaño B. D. (DATA) | 1 | nominal = 38 D/P |
| Complejidad (CPLX) | 0,87 | bajo |
| Reusabilidad (RUSE) | | |
| Documentación (DOCU) | | |
| Restricción t° ejecución (TIME) | 1,11 | alto |
| Restricción almacenamiento principal (STOR) | 1 | nominal |
| Volatilidad Plataforma (PVOL) | | |
| Capacidad del analista (ACAP) | | |
| ... del programador(PCAP) | 1,34 | alto |
| Experiencia en las aplicaciones (APEX) | | |
| ... en la plataforma(PLEX) | | |
| ...Lenguaje/herramienta ..(LTEX) | 0,91 | alto |
| Continuidad del personal (PCON) | | |
| Uso de herramientas SW (TOOL) | 0,9 | alto |
| Desarrollo multilugar (SITE) | | |
| Calendario de desarrollo requerido (SCED) | | |
| M= | 0,86905004 | |

Ilustración 240 - Calculo de los multiplicadores

El multiplicador **M**, que es él nos define el nivel **Postarquitectónico**, el cual lo calculamos a través de unas tablas con sus respectivos valores nominales, dependiendo del grado de conformidad con dada apartado tratado.

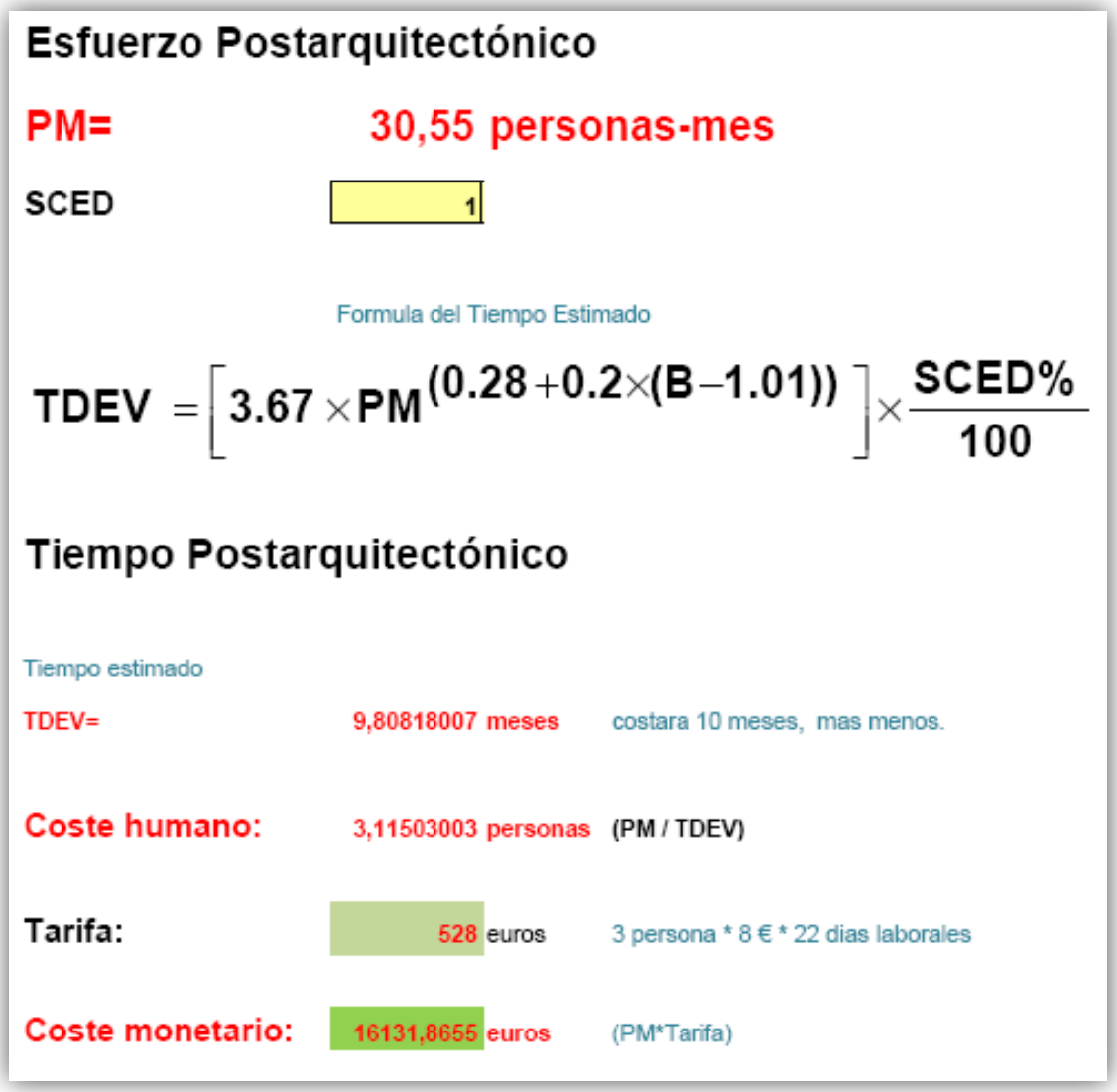



Ilustración 241 - Calculo del Esfuerzo, Coste Humano y el Coste Monetario

Por último calculamos el esfuerzo (PM), que se mide en personas-mes.

Como muestra la gráfica anterior, este valor es 30.55, lo que quiere decir es que necesitamos 35 personas más o menos trabajando en conjunto para terminar este trabajo en un mes exacto.

Ahora, con la fórmula del tiempo estimado (TDEV) calculamos el tiempo en calendario en meses, y obtenemos que nos costara 9.80 meses más o menos.

Después de saber esto, podremos saber cuántas personas necesitamos para realizar este proyecto, para eso dividimos el esfuerzo (PM) entre el (TDEV) y obtenemos que equivale a 3.11 personas. Yo he tomado 3 personas trabajando para este proyecto.

Con estos datos, ya sabemos que tres personas pueden terminar este proyecto en 9 meses más o menos. Ahora falta calcular su coste por persona y por proyecto.

La tarifa total por los tres programadores es de 528 €, ya que estos trabajarán 22 días laborales a 8 euros por persona al día.

El coste del proyecto "Workout Routines" es de 16.131,87 €. Que es el resultado de medir el esfuerzo por persona, más una serie de características que definen lo grande del proyecto y las habilidades o desventajas de los desarrolladores. Estos valores fueron tomados teniendo en cuenta que soy novato en este tipo de proyectos.

- Es por tanto, que el proyecto final costara **16.131,87 €.**
- Con un tiempo de finalización de **9 meses**, desarrollando **tres personas**.
- Cada desarrollador cobrará **1200 €** mes.
- Descontando el salario. El Neto parcial es: **12.531,87 €**.
- No hay que olvidar que a este Neto parcial, habrá que descontar los gastos más comunes, como la luz, el agua, el papel, las licencias de software si se requieren, impresoras, tintas, local, etc. Esto quiere decir, que habrá que subir un poco el valor del proyecto.
- Para dejar un margen de error entre las ganancias y las pérdidas, este proyecto costará alrededor **15.000 €.**

Estos datos serían los dados al cliente, en el caso que tuviera que dar un presupuesto del desarrollo de este tipo de proyecto, con esta nueva tecnología.

## 7.3. Presupuesto Real

Para este apartado, he tenido en cuenta que un programador de media suele cobrar unos 10 € por hora, y teniendo en cuenta que este proyecto ha costado 3 meses y 15 días (105 días) en desarrollarse, trabajando unas 30 horas a la semana, dando un total de 450 Horas trabajadas.

Además solo ha intervenido un programador, que soy yo.

Y todo esto da como resultado un total de proyecto:

- **Total del proyecto** = 450 h * 10 € = **4.500 €**

- Pero a este valor hay que sumar un valor de ganancia, ya que luego hay que descontar lo que son gastos comunes, como el combustible, las comidas, gasto energético, el pago de acceso para *Android Developer*, etc.

- No tengo en cuenta licencias de software, ya que son gratuitas (Desarrollo *Android - JAVA*) o la mayoría son proporcionadas por la universidad, tampoco el equipo utilizado.

- Así que, para tener un margen de ganancia aceptable, podría subir el valor de esta aplicación a **8.000 €**. para tener un margen de ganancia es de **3.500 €**.

Además con *Google AdMob*, el portal publicidad de visitas para las aplicaciones en *Android*, puedo obtener más ganancias de lo que quiero obtener, ya que porcada 100 solicitudes, estos me dan como ganancia 1 cent de dólar.

Y las ganancias obtenidas por usar esta herramienta alrededor de dos meses y medio son de 0.28 $, tal cual como muestro en la siguiente imagen.


**Ilustración 242 – Estadísticas de ganancia de la publicidad de AdMob**

# 8. CONCLUSIONES Y LÍNEAS FUTURAS.

Al final, se ha obtenido una aplicación que es capaz de ayudar y guiar a las personas cuando están entrenando en un gimnasio, ya sea, porque son novatos y no tienen idea de cómo se realiza cierto ejercicio o porque son personas con algo de experiencia y que quieren llevar sus entrenamientos a todos los lugares posibles, por ejemplo, para comparar su rutina con otras personas.

Esta aplicación es completamente funcional, ya que es capaz de instalarse en **2219** dispositivos diferentes y software superior o igual a la versión **2.3.3 de *Android***.

La siguiente imagen muestra algunos dispositivos admitidos, solamente unos cuantos, porque la lista es muy extensa.

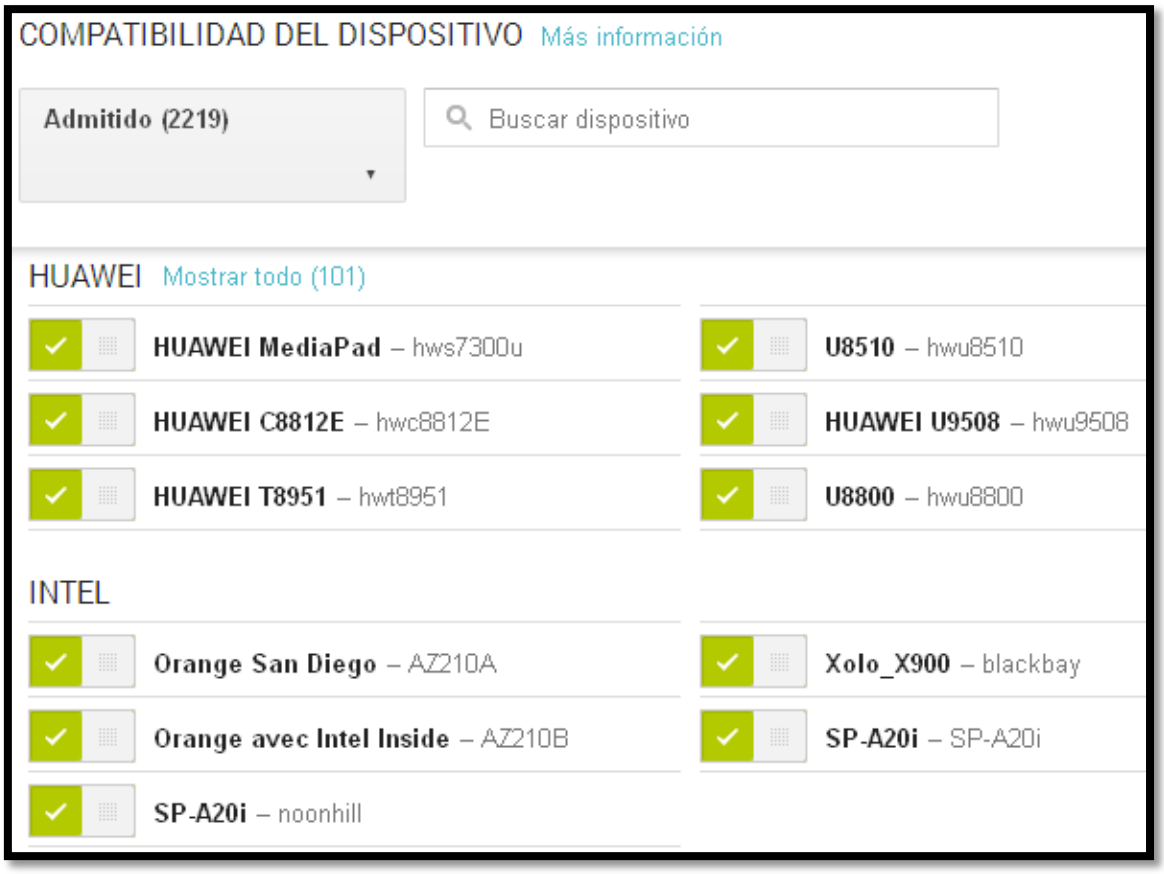



Ilustración 243 - Móviles capaces de soportar la App Workout Routines

Además, esta aplicación es bilingüe, lo que permite que esté en inglés o español y de esta manera podamos llegar a más personas, no solo a las de habla hispana.
Aunque esta acción podría expandirse a más idiomas, ya que se ha preparado para este cambio sea lo más fácil posible, por si se quiere adaptar ya sea apara alemán o francés.

Por otra parte, al hacer esta aplicación, he aprendido mucho de cómo llevar este tipo de proyectos, no solo por haber aprendido el lenguaje *Android*, sino también, de cómo llevar a cabo una buena planificación y estructuración de lo que se quiere diseñar y construir, idea válida para cualquier proyecto software.

También, he aprendido que hay que adaptarse a las necesidades de los clientes, ya que ellos son los usuarios finales, y que hay que saber llegar a la gran mayoría de ellos.

Todos los objetivos están cumplidos, tal como se han especificado, ya que la idea es hacer un proyecto completo, partiendo de las necesidades de las personas durante un entrenamiento en un gimnasio.

Además, he aprendido a valorar un proyecto, ya sea por su coste monetario, duración o tamaño, ya que para ello, me valgo de los métodos de análisis de puntos de función y COCOMO II enseñados en la asignatura de Planificación y Proyectos Informáticos

Como mejora para este proyecto, me he planteado la idea de ampliarlo para las personas que les gusta correr. La mejora sería agregar un mapa de Google, donde se trace la ruta que sigue cada atleta, y con estos datos, calcular cuánto le ha costado llegar de un lugar a otro y ver si hay mejoras con las diferencias de los tiempos de un día con el anterior.

Además, me había planteado agregar más ejercicios, ya que al parecer los 40 ejercicios que tenemos para hacer rutinas de entrenamientos musculares no son suficientes. Esto es una recomendación que me habían hecho en la encuesta.

Otra recomendación es linkear los ejercicios con las formas en que se ejecutan estos, esto es para minimizar las búsquedas que los usuarios tienen que hacer. La nueva mejora será que el acceso sea directo entre la tabla de entrenamientos con las nombres de los ejercicios musculares que existen.

Otra mejora podría ser la inclusión de dietas, ya que esto es fundamental para ganar masa muscular y perder peso, aunque para ello tendría que hablar con especialistas de la nutrición, que me ayuden a estructurar esto a nivel general, porque las dietas son personales y dependen de la constitución de cada persona.

# 9. BIBLIOGRAFÍA

[1] API Android Desarrollador: http://developer.android.com/index.html

[2] IDE Eclipse: http://www.eclipse.org/

[3] Ingresos publicitarios AdMob: http://www.google.es/ads/admob/

[4] Consola Desarrollador Google Play: https://play.google.com/apps/publish/

[5] Android Ya: http://www.javaya.com.ar/androidya/index.php?inicio=0

[6] Tutoriales Androideity: http://androideity.com/

[7] Video tutoriales Edu4Java: http://www.edu4java.com/android.html

[8] Hosting, Hostinger: http://www.hostinger.es/

[9] Guía AFreeChart: http://code.google.com/p/afreechart/

[10] Tutoriales Vogella: http://www.vogella.com/android.html

[11] Video tutoriales YouTube: http://www.youtube.com/user/0utKast

[12] Tutorial web: http://www.androidhive.info/

[13] Ejercicios de Culturismo y Fitness:

http://fitnessypesas.webcindario.com/listadoejer.htm

[14] Plugin Eclipse para la creación de diagramas UML, Soyatec:

http://www.soyatec.com/euml2/

[15] Plugin Eclipse para la creación de diagramas UML, ObjectAid:

http://www.objectaid.com/home

[16] Tutorial web android, EduMobile: http://www.edumobile.org/android/

[17] Página de ejercicios en Ingles, netfit: http://www.netfit.co.uk/

[18] Foro de programadores: http://stackoverflow.com/

## 10. ANEXO

# Manual de Usuario

Aplicación Workout Routines

## VISUALIZAR RUTINAS PREDETERMINADAS

Para empezar a usar esta aplicación, es recomendable que eches un vistazo a las rutinas ya creadas por nosotros, esto te guiara para que puedas hacerte a la idea de cómo ir construyendo una rutina que se acomode a tus necesidades y no a las de otra persona. Para ello tienes que estar en el ***Menú Principal*** y pulsar ***Entrenamientos.***


Una vez seleccionado esta opción, después tendrás que seleccionar la pestaña que dice ***Predeterminado***.


En esta encontraras rutinas diseñadas para procurar la menor fatiga muscular posible por cada entrenamiento seleccionado, recuerda que tienes tres posibles, ya sea para ganar masa **muscular**, **tonificarte** o simplemente ganar **resistencia**. Además, te indicamos las series que debes hacer por cada ejercicio seleccionado.

Cuando estés entrenando, procura utilizar un peso adecuado, ya que este dependerá de la fuerza que tengas.

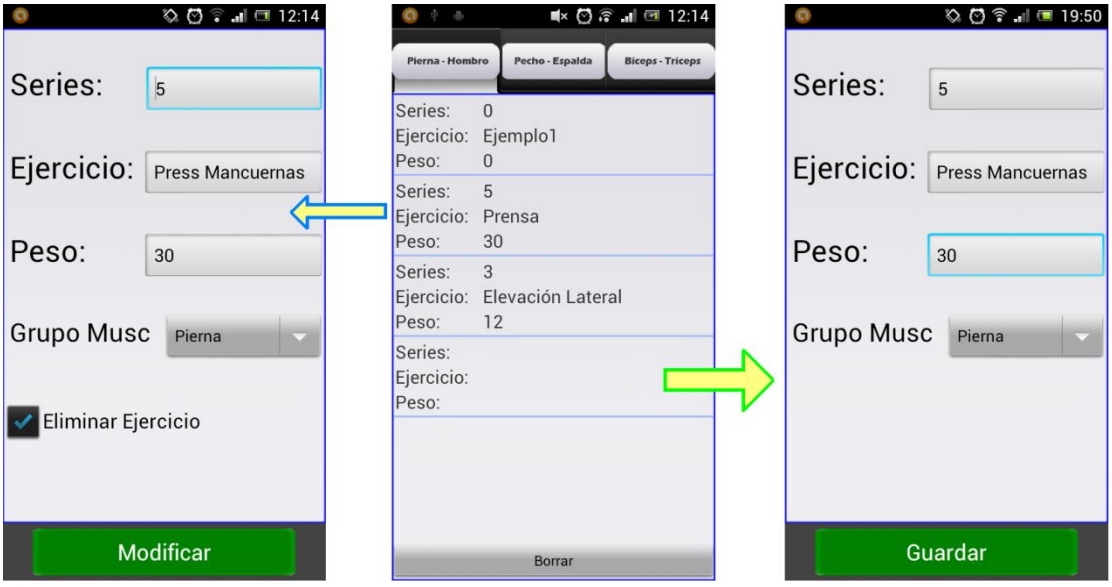
## INSERTAR UN ENTRENAMIENTO NUEVO

Una vez que ya te hayas acostumbrado a una rutina, intenta construir una rutina que solamente sea para ti.
Estando en el ***Menú Principal***, ve al botón ***Entrenamientos*** y selecciona la pestaña ***Personalizado***, en ella podrás ver un banner de imágenes y tres botones, los cuales fueron diseñados para que puedas diferenciar y categorizar tus entrenamientos

Una vez dentro, te saldrá una pantalla como esta.


Puedes seleccionar cualquier botón para almacenar tus rutinas, solo recuerda para que lo necesitas.

Una vez dentro, podrás modificar las series, el ejercicio y el peso asociado a este entrenamiento. Pulsando en modificar o guardar.

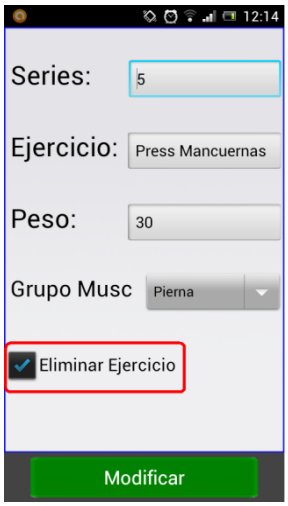


## MODIFICAR UN ENTRENAMIENTO NUEVO

Estando en la pestaña de **Entrenamientos Personalizados**, selecciona el ejercicio que quieras modificar, allí puedes cambiar el peso, el nombre del ejercicio o las series a hacer. Solo basta con cambiar lo que quieras y dar al botón modificar, de esta manera tus datos estarán actualizados. Esta herramienta se puede apreciar en la imagen anterior

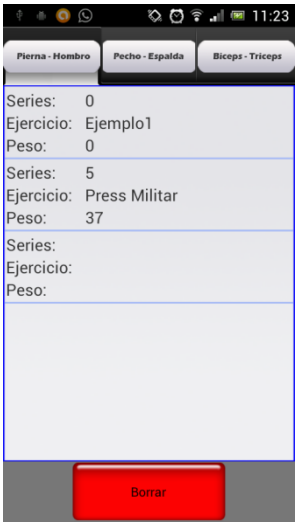
## ELIMINAR UN ENTRENAMIENTO

Para eliminar un entrenamiento es necesario tener al menos un ejercicio creado, para eliminar solo basta con seleccionar el ejercicio a eliminar y marcar la casilla de eliminar ejercicio, y pulsar el botón de modificar y listo. Con esto ya tendríamos el ejercicio eliminado de la rutina.



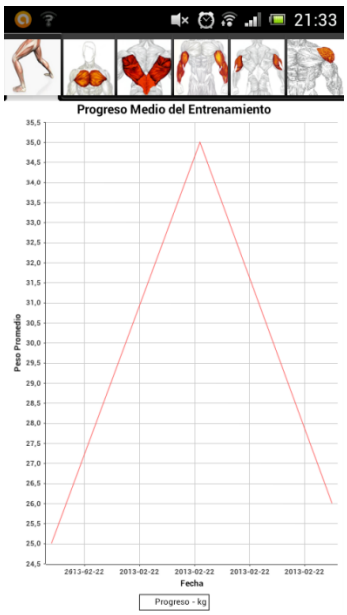

## ELIMINAR TODOS LOS ENTRENAMIENTOS

Para eliminar todos los entrenamientos de golpe, solo tienes que pulsar el botón rojo que indica Borrar, al hacer esta acción, te saldrá un mensaje de confirmación, ya que eliminara todos los datos existentes en la BD.



## VISUALIZAR RENDIMIENTO

Para visualizar tu progreso, debes posicionarte en el ***Menú Inicio*** y pulsar el botón de ***Progreso***, en este te saldrán seis pestañas, las cuales están indicadas por un grupo muscular trabajado. Selecciona la de tu interés y miras cual ha sido tu progreso. Recuerda crear un entrenamiento personalizado, ya que si no, tu progreso será Cero.




## COMPARTIR LA APP

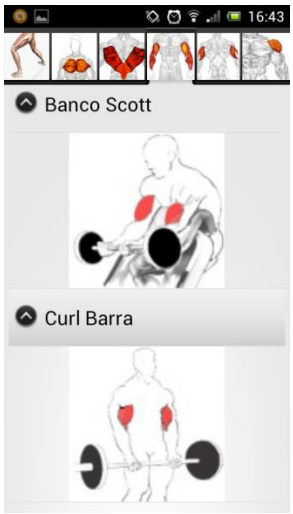
Esta opción está diseñada para que compartas esta App con tus amigos en las redes sociales, para ello ve al Menú Principal y selecciona el botón que es encuentra en la parte superior derecha de esta pantalla. En ella te saldrá una lista con las posibles redes sociales. Selecciona la de tu interés y pulsa a aceptar.

## COMO EJECUTAR LOS EJERCICIOS CORRECTAMENTE

Esta herramienta está diseñada para guiar a las personas noveles durante su entrenamiento.

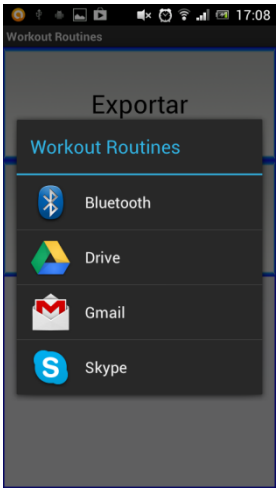
La idea consiste en que puedas ver las diferentes formas de entrenar cada musculo, y de aprender el nombre con que se le conoce.

Ve al Menú Inicio, pulsa el botón Ejercicios, en este apartado podrás ver una gran variedad de ejercicios, diferenciados por grupo muscular, aquí podrás ver el nombre de cada ejercicio, con su correspondiente imagen de cómo ejecutarlo de forma segura.




## EXPORTAR ENTRENAMIENTO

Para exportar un entrenamiento, debes tener habilitado el Bluetooth, junto con la persona a la que quieres enviar la rutina y estar vinculados. Una vez hecho esto, lo siguiente es ir al menú Inicio de la aplicación y pulsar el botón de exportar, este automáticamente cargara toda tu rutina, para enviarla por Bluetooth con las personas a la que elijas de tus dispositivos vinculados.

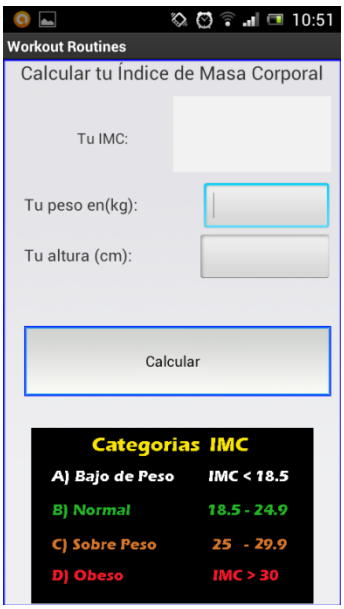


## IMPORTAR ENTRENAMIENTO

Para importar el entrenamiento, debes tener esta App instalada. Luego ve al Botón de Exportar – Importar y pulsa el Botón de ***Importar***, automáticamente esta opción cargará toda la rutina en tu aplicación y podrás usarla en cualquier momento.


## CALCULA TU ÍNDICE DE MASA CORPORAL

Para calcular tu IMC (Índice de Masa Corporal), debes introducir tu **Altura** (cm) y tu **Peso** (kg), y después pulsar el botón de calcular, este de manera automática te dará un índice, el cual, luego debes comparar con las categorías del IMC y mirar tu estado de salud con respecto al porcentaje de grasa corporal acumulada.



# PROYECTO DE FIN DE CARRERA

## APLICACIÓN ANDROID
## "WORKOUT ROUTINES"

**Ronald Sandoval Sandoval**

**Tutor: José Javier Astrain Escola**

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

# OBJETIVOS

- ✓ **Facilitar el uso de tablas de entrenamientos en los gimnasios, haciendo portable su rutina y pueda ser compartida entre los usuarios.**
  - Aprender a desarrollar aplicaciones en plataforma Android, para dar servicios a un sinfín de dispositivos móviles (*Smartphone*).
  - Aprender a integrar bases de datos, con interfaz usuario–servidor y conexiones a servidores externos.
  - Aprender a usar e implantar *APIs* externas, como la de *AFreeChart*, *Google AdMob* o el propio *SDK* de *Android*.

# PROBLEMA A RESOLVER

**Los entrenadores de los gimnasios asignan rutinas de entrenamientos a las personas que lo deseen en una hoja impresa.**

* Las personas suelen perder o olvidar estas hojas.
* Es por eso que he previsto la necesidad de hacer una aplicación para móviles.
* Con esto nos aseguramos de que las personas no extravíen sus rutinas y por ende tengan que improvisar, con el agravante de que todo el entrenamiento sea en vano.

# SOLUCIÓN PROPUESTA

➢ **LOS DISPOSITIVOS MÓVILES SON PARTE DE NUESTRO DÍA A DÍA**

* Elaborar una interfaz sencilla.
* Listas dinámicas donde el usuario pueda crear, modificar o eliminar sus ejercicios.
* Mostrar imágenes de ejecución de los entrenamiento.
* Gráficas de progreso.
* Facilidad de exportar e importar la tabla de entrenamientos.
* Que sea bilingüe, inglés o español.
* Cálculo del índice de la masa corporal (IMC).

# ESTADO DEL ARTE

* Algunas aplicaciones suelen ser gratuitas.
* Aplicaciones totalmente en inglés.
* Menús complicados.
* Entrenamientos ya definidos.
* Desventaja de algunas aplicaciones como la de *Adidas* o *Nike.*

# METODOLOGÍA PROPUESTA

* Metodología Proceso Unificado (UP/RUP) .

  - Iterativo e Incremental.
  - Asegurar un software de calidad.
  - Software ajustado a las necesidades de los usuarios finales.
  - Conocimiento amplio del producto a desarrollar.

# ANÁLISIS

## Requisitos Funcionales

* Debe ser compatible para la gran mayoría de dispositivos móviles.
* Ofrecer una tabla de entrenamiento predeterminada y una rutina que pueda ser personalizada.
* Mostrar imágenes de ejecución de los entrenamientos, diferenciadas por grupo muscular.
* Facilitar el intercambio de los entrenamientos.
* Mostrar como progresan los usuarios a través de gráficas.

# ANÁLISIS

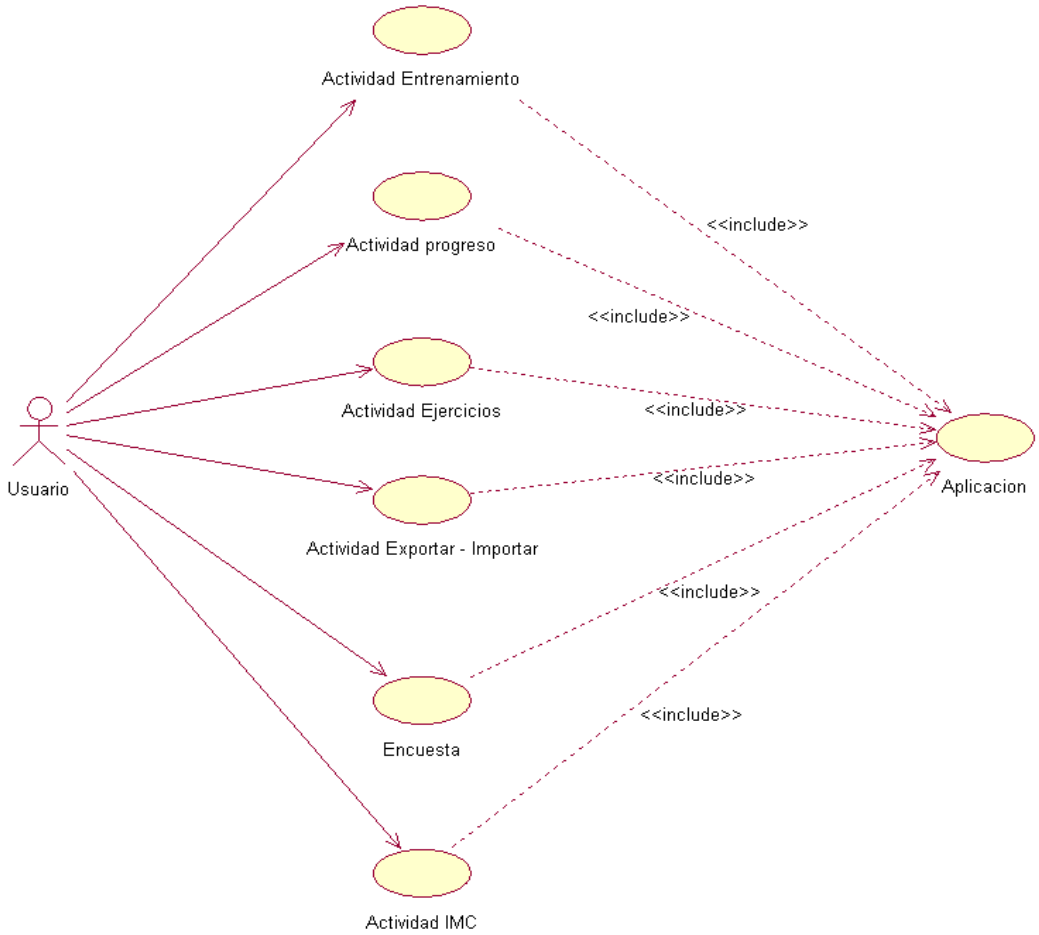
## Requisitos No Funcionales

* La información almacenada sea lo más clara posible.
* El diseño debe estar pensado para personas de una edad comprendida entre los 16 y los 40 años.
* Ofrecer publicidad a la aplicación.
* Informar de los errores en caso de que se produzcan.
* Facilitar la búsqueda y descarga.

# ANÁLISIS

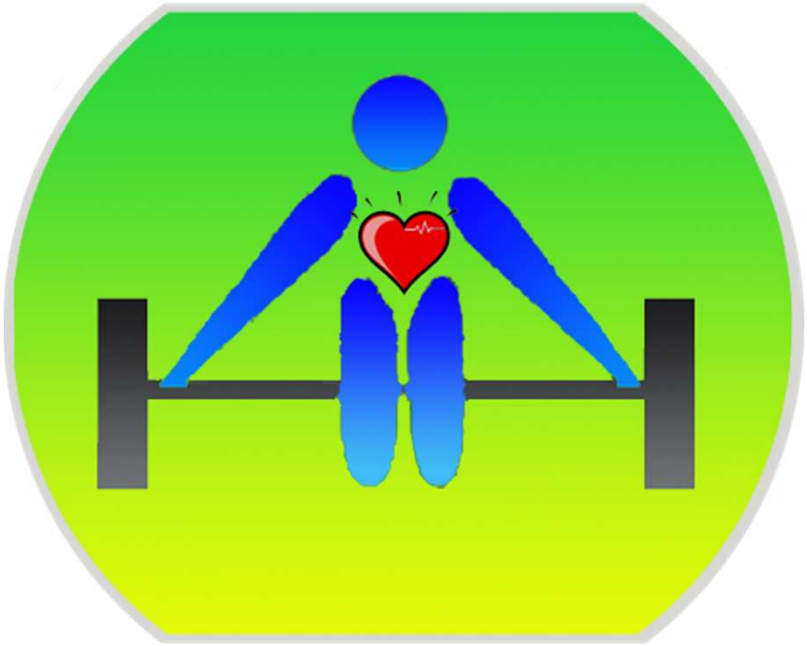
## Caso de Uso General

* Usuarios interactúan con la aplicación.



# DISEÑO

## LOGOS



Logo de la aplicación


Nombre del producto

# DISEÑO

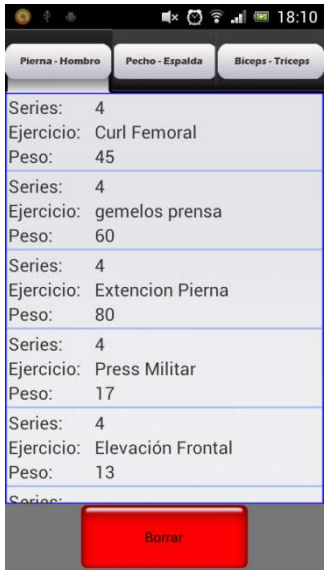
## Menú Principal


Sin publicidad

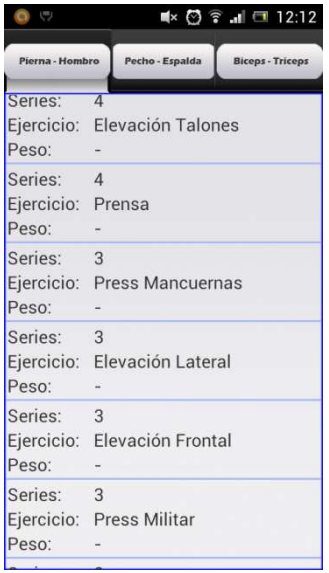

Con publicidad

# DISEÑO

## Entrenamiento



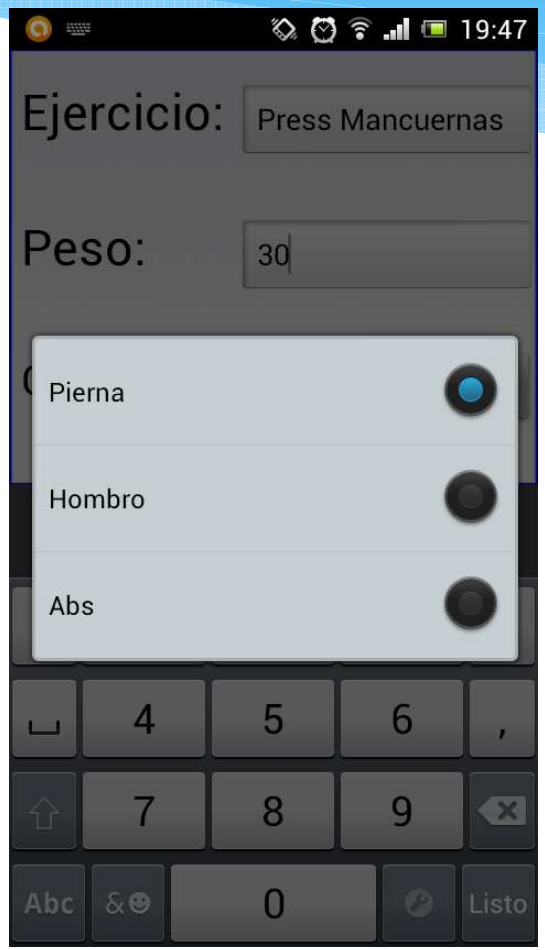


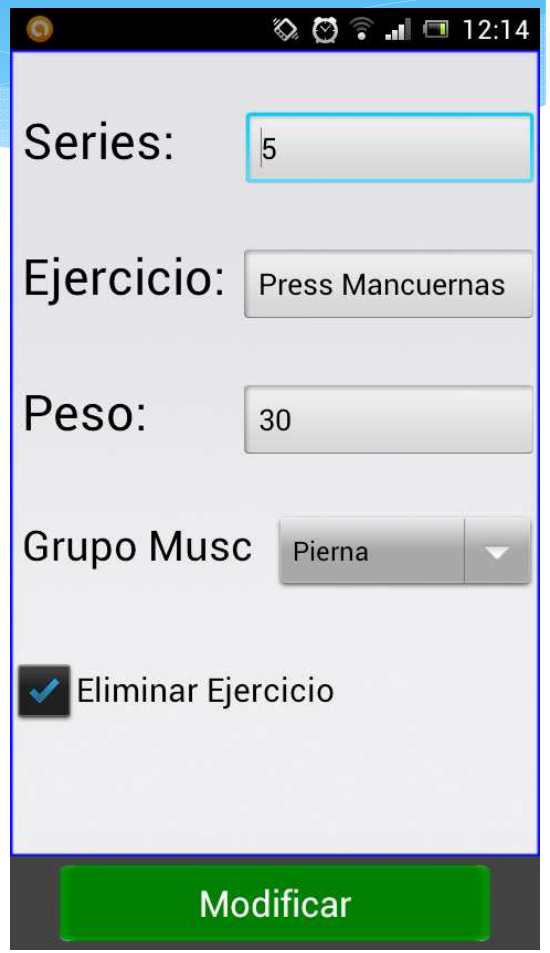

**Entrenamiento Personalizable**

**Entrenamiento Predefinido**

# DISEÑO

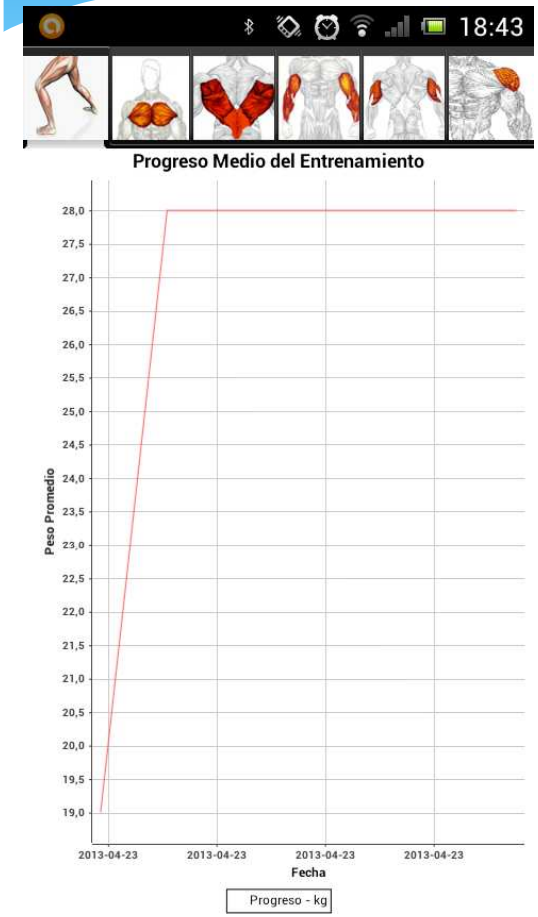
## Crear o Modificar

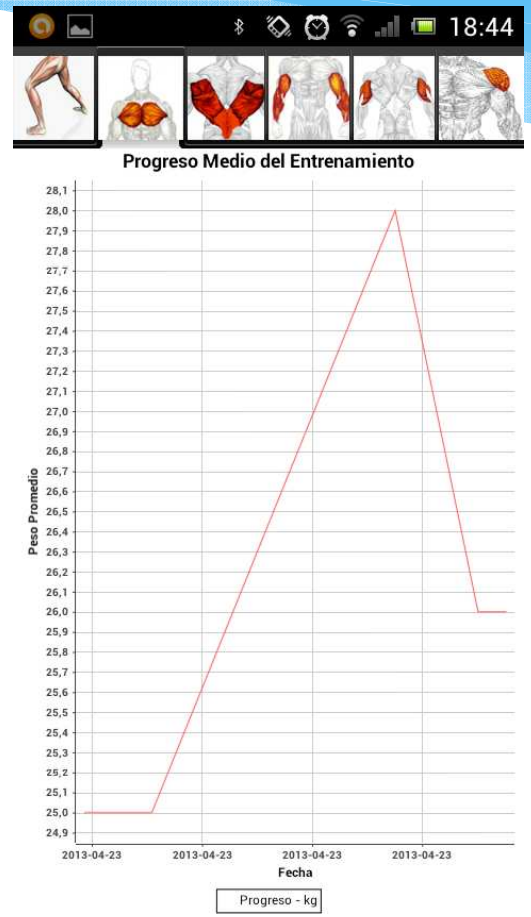



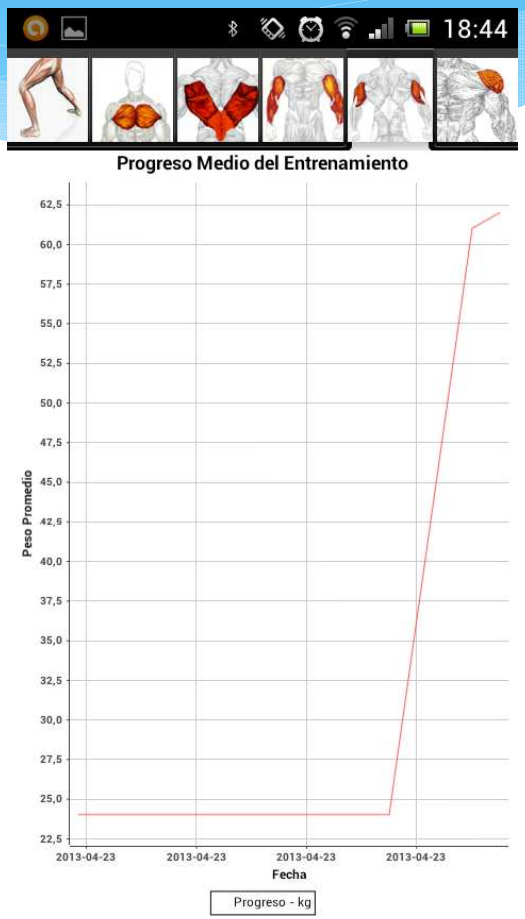



# DISEÑO

## Progreso

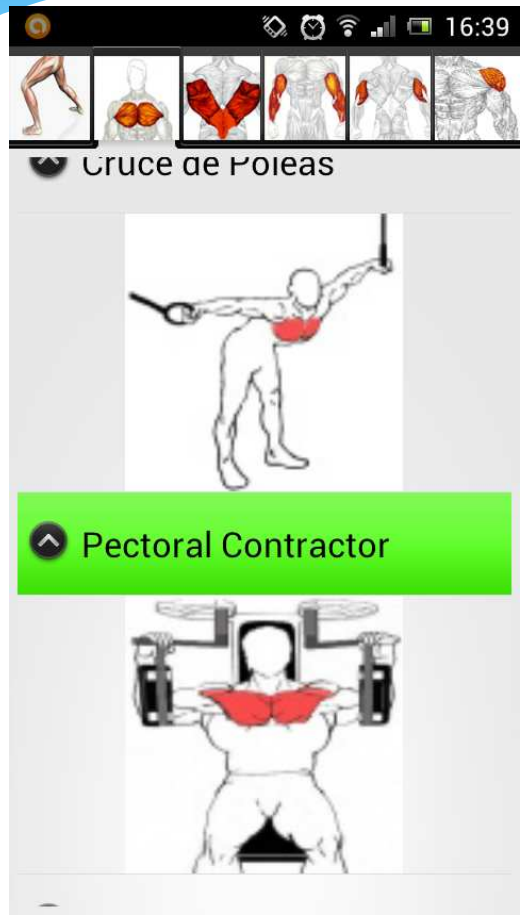


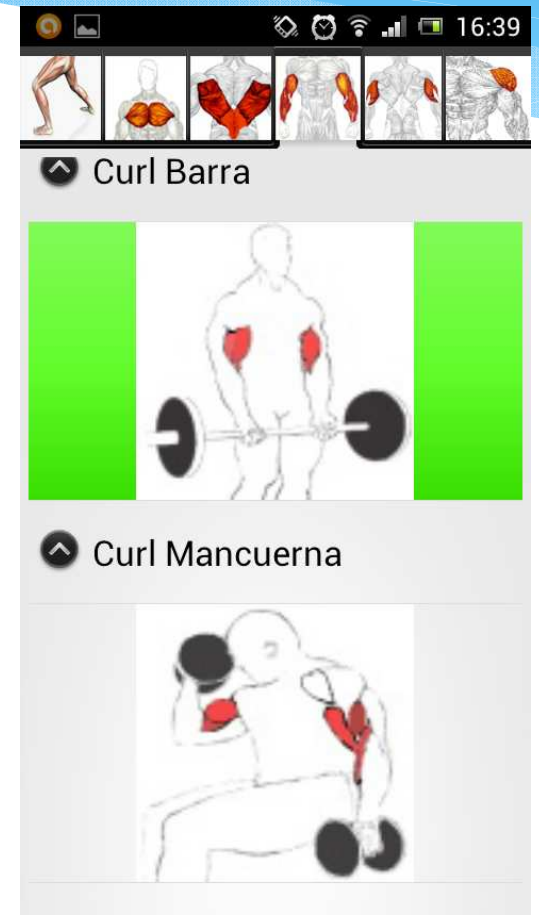



# DISEÑO

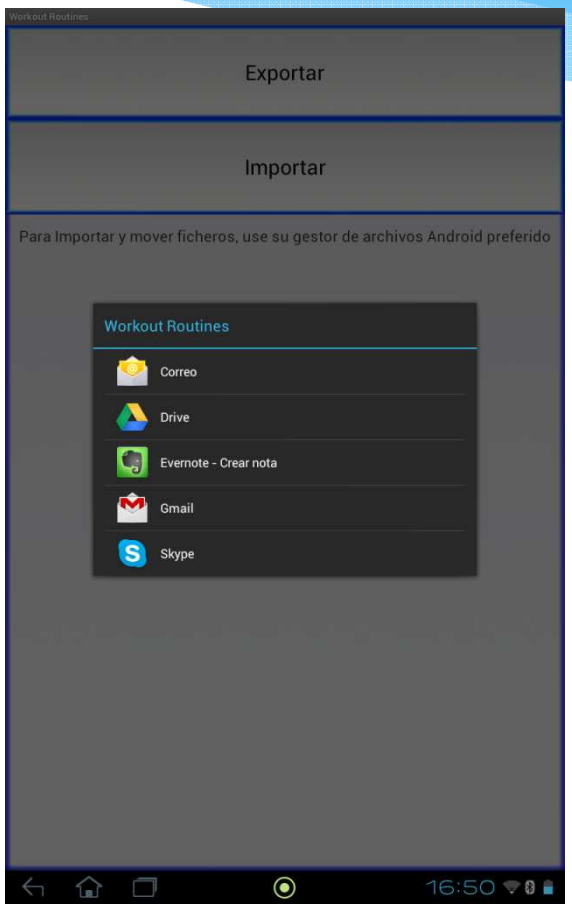
## Formas de ejecutar los ejercicios

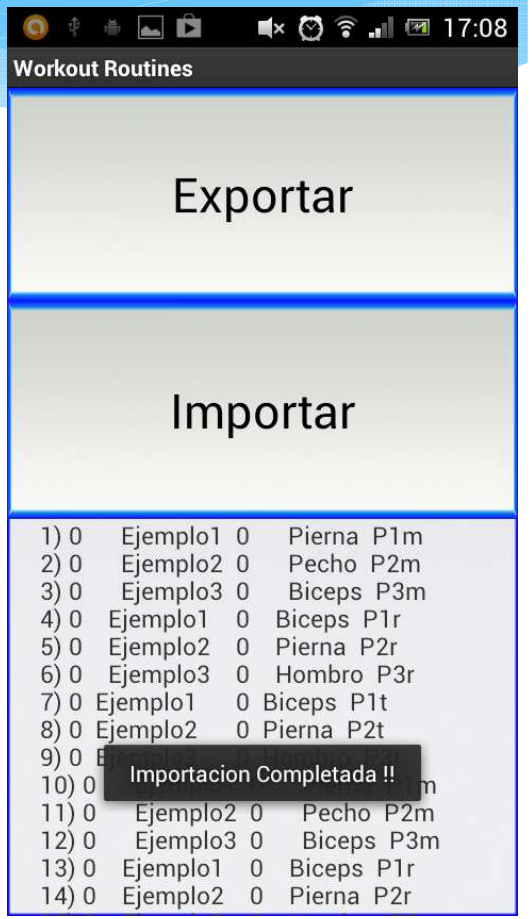



# DISEÑO

## Exportar - Importar

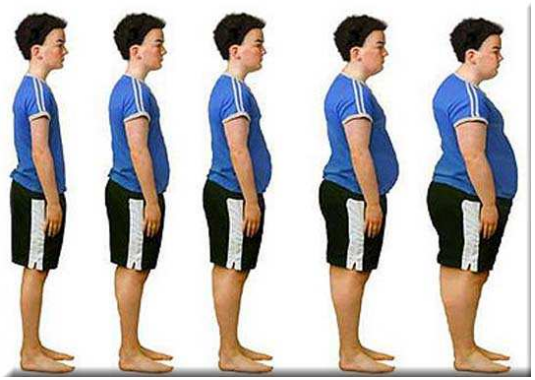
# DISEÑO


## Encuesta

# DISEÑO

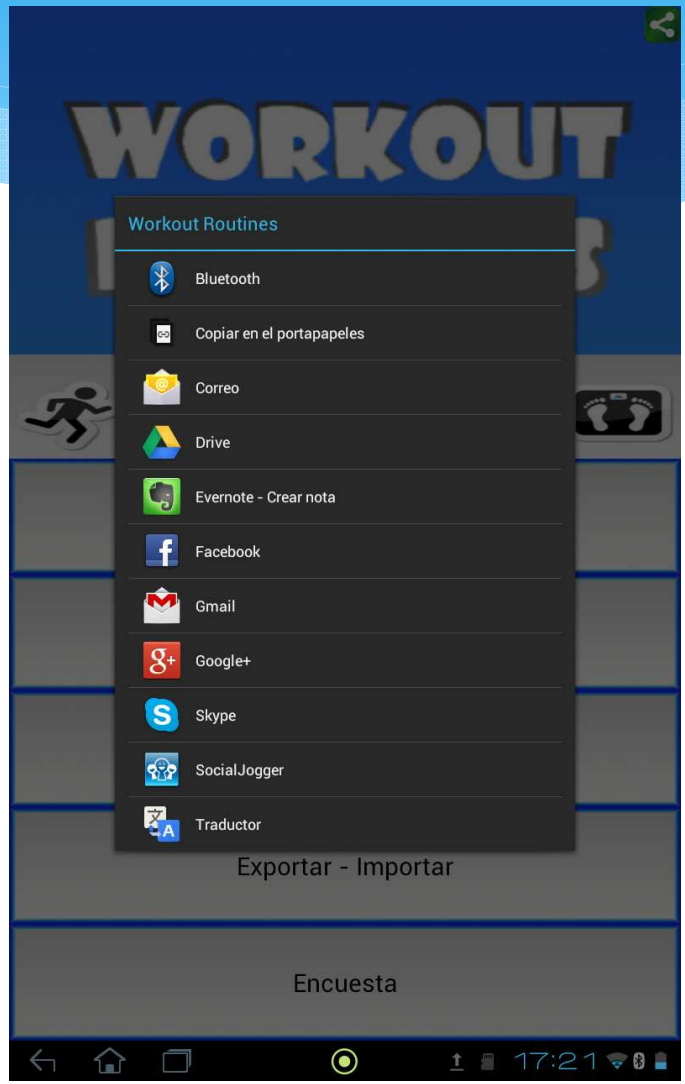
## Calculo del Índice de Masa Corporal

$$\mathrm{IMC} = \frac{peso(kg)}{altura^2(m)}$$

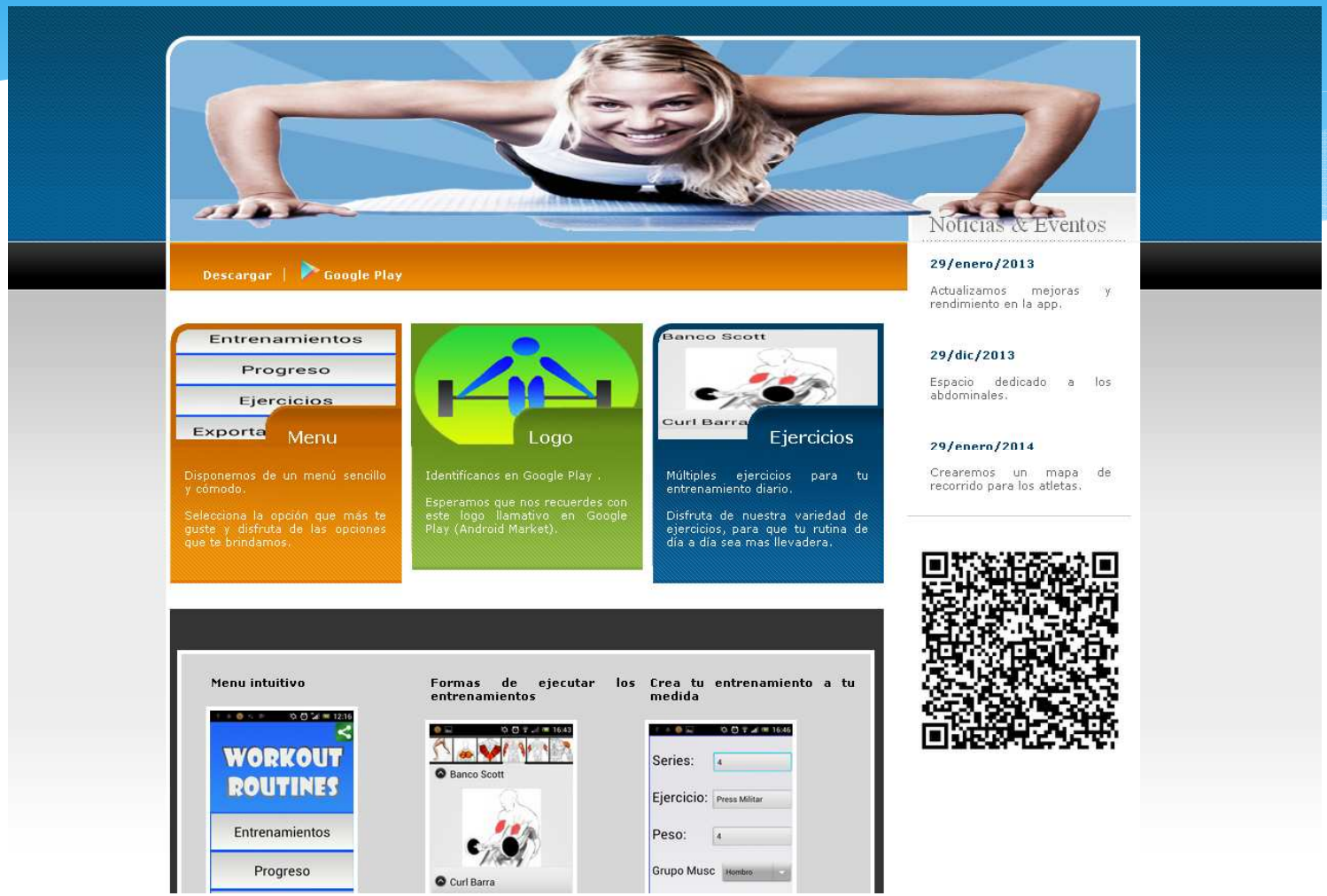


# DISEÑO

## Opción Compartir

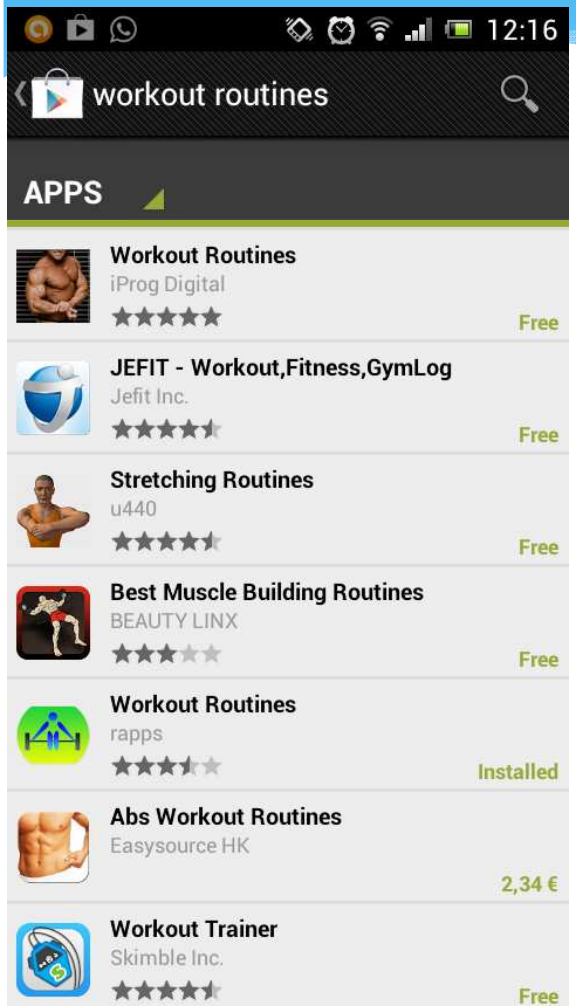


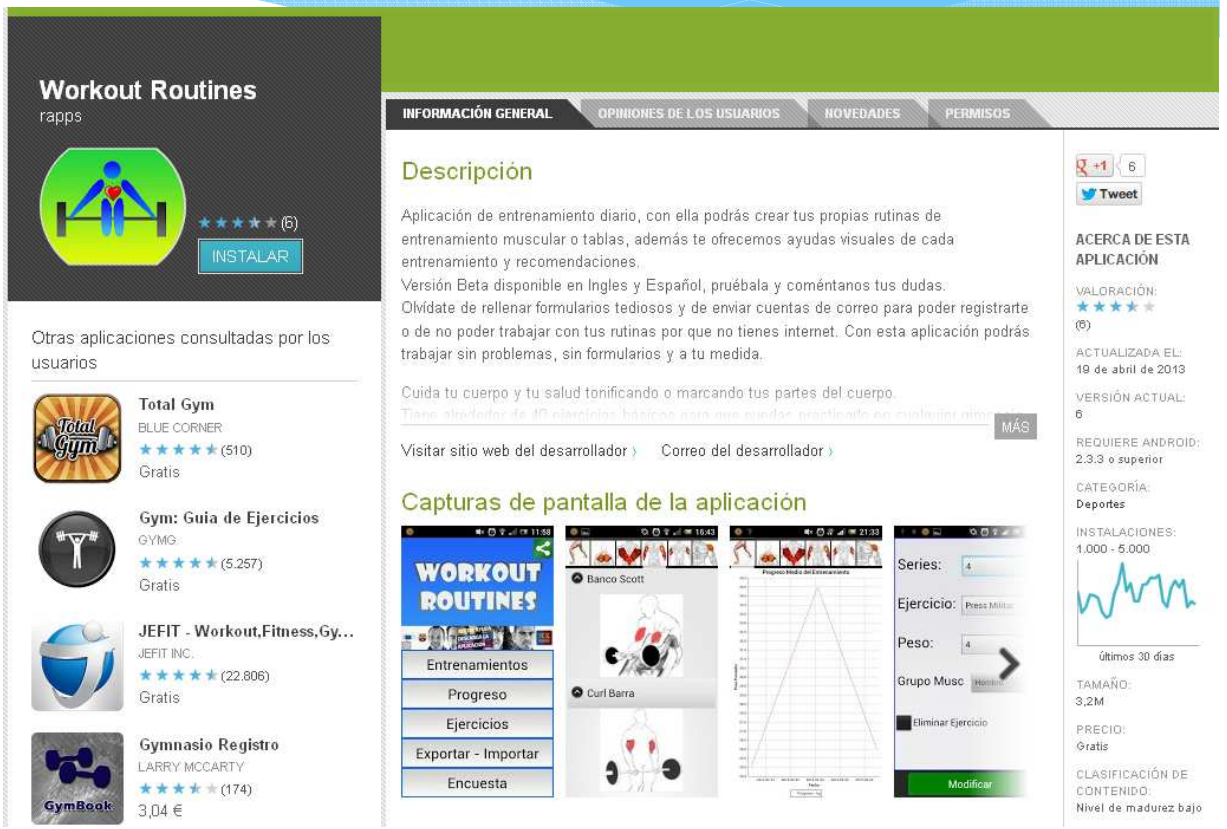
# PÁGINA WEB




www.workoutroutines.hol.es

# GOOGLE PLAY



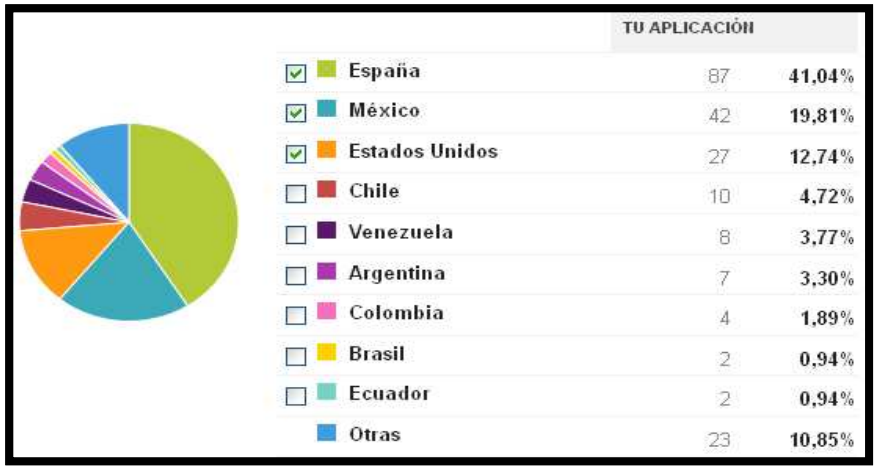
# PRUEBAS

- Datos ofrecidos por la Consola de Desarrollador de Google

| PRECIO | INSTALACIONES ACTIVAS / TOTALES | VALORACIÓN MEDIA / TOTAL | ERRORES Y ANRS |
|---|---|---|---|
| Gratuita | 212 / 851 | ★ 3,50 / 4 | |

Datos carga del servidor Google Play Developer


Instalaciones activas en cada plataforma



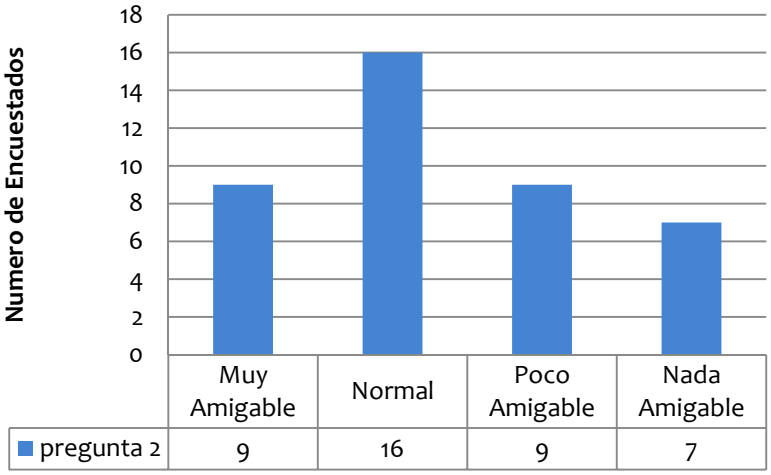

Países donde más han descargado esta App

# PRUEBAS

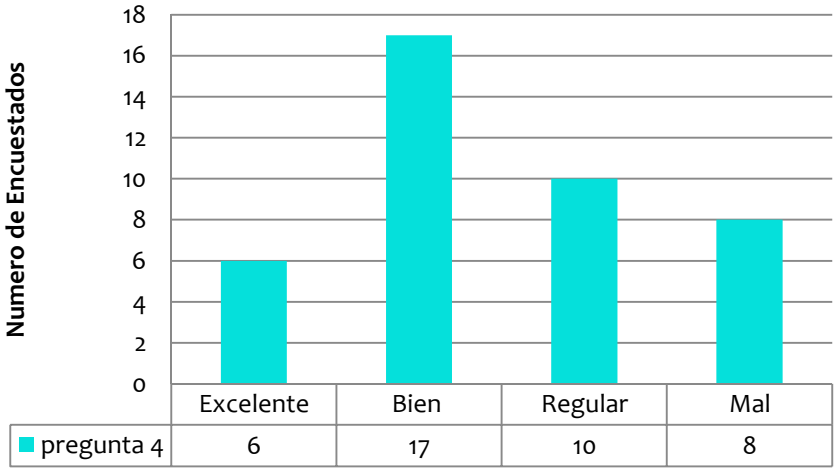
## Resultados de la encuesta

- ✓ Personas encuestadas, 41.
- ✓ La aplicación es fácil de usar.
- ✓ Tendrá muy buena aceptación.

### ¿ Tiene un diseño Intuitivo ?




### Puntúame



# CONCLUSIONES

* Se ha obtenido una aplicación capaz de ayudar y guiar a las personas durante su entrenamiento.
* Esta aplicación es compatible en más **2219** dispositivos diferentes, con un software superior o igual a la versión **2.3.3 de *Android***.
* Todos los objetivos están cumplidos.
* He aprendido que hay que adaptarse a las necesidades de los clientes y de cómo llevar este u otro tipo de proyectos.
* Aprendizaje en desarrollo de aplicaciones Android.

# LÍNEAS FUTURAS

* Agregar más ejercicios.
* Incluir un apartado de dietas.
* Ampliar la aplicación para que se adapte a los deportistas que les gusta correr.
* Crear atajos, con las rutinas y los ejercicios.
* Incluir un apartado sólo con tablas de Abdominales.

# PROYECTO DE FIN DE CARRERA

## APLICACIÓN ANDROID
## "WORKOUT ROUTINES"

**Ronald Sandoval Sandoval**

**Tutor: José Javier Astrain Escola**

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa