Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación Android para ayudar a controlar los gastos personales

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Caramés Edo, Vicente

Tutor: Fons Cors, Joan

2015-2016

Resumen

Un hito indiscutible en la evolución de los teléfonos móviles fue la aparición de los Smartphone, estos unen las características de un teléfono móvil y un ordenador. Su rápida expansión ha supuesto que hoy en día casi todo el mundo lleva un mini ordenador en su bolsillo. Como estudiante de Grado en Informática y apasionado del desarrollo software no pude evitar pensar que tipo de aplicación podría ser interesante teniendo en cuenta que esta se utilizaría en un dispositivo que el usuario iba a llevar encima en todo momento.

Así pensé en una aplicación para gestionar los gastos diarios, que será el caso de estudio a desarrollar en este proyecto. Algunos gastos corrientes como préstamos, agua, electricidad, etc.. se controlan fácilmente a través de la página web de las entidades bancarias, sin embargo otro tipo de gatos comunes como salidas a restaurantes, almuerzos, cine, peluquería, gastos del vehículo y en general gastos menores que no son recurrentes son más difíciles de controlar y a menudo representan una parte importante de nuestro sueldo. En este tipo de gastos es en los que se centra la aplicación.

Así el objetivo principal debe ser permitir al usuario introducir los gastos e ingresos de una forma sencilla y rápida y posteriormente mostrarle toda esta información de una manera ordenada y fácil de interpretar.

Palabras clave: móviles, aplicación, control de gastos.

Abstract

An indisputable milestone in the evolution of mobile phones was the emergence of the Smartphone, they bring together the features of a mobile phone and a computer. Its rapid expansion has meant that today almost everyone carry a mini computer in their pocket. As a student of Degree in computer science and passionate of the software development I could not avoid thinking what kind of application could be interesting considering that it would be used in a device that the user would carry at all moment.

Then I thought on an application to manage daily expenses, which will be the case of study in this project. Some current expenses such as mortgage, water, electricity and so on are easily controlled through the website of the banks. However other common expenses as restaurants, cinema, hairdresser, vehicle and generally low non-recurring expenses are more difficult to control and often they represent an important part of our salary. It is in this type of expenses in which the application is focused.

Then the main objective must be to enable end users record their expenses in an easy and fast way and display the information in an easy way to analyze.

Keywords : mobile, application, control expenses.

Tabla de contenidos

1. Introducción

1.1 Motivación

La realización del presente Trabajo Final de Grado (TFG) me permite finalizar los estudios de Grado en Ingeniería informática y me da la oportunidad de poner en práctica los conocimientos adquiridos durante mis estudios.

Llevar a cabo un proyecto práctico que te permita desarrollar los conocimientos adquiridos en las distintas asignaturas del grado es algo estimulante y que recomendaría a cualquier alumno, además es algo tangible que da un valor añadido a tu CV. En mi caso he optado por desarrollar una aplicación nativa para la plataforma Android pero bien podría haber sido cualquier otra cosa ya que durante el grado adquieres conocimientos que te permiten trabajar sobre áreas muy diversas.

1.2 Objetivo

El objetivo de este trabajo consisten en proporcionar al lector una visión general sobre las distintas plataformas de desarrollo software para dispositivos móviles y tabletas, así como de las diferentes herramientas, lenguajes de programación y enfoques que se pueden adoptar a la hora de desarrollar aplicaciones para estos dispositivos. Una vez el lector tenga una visión general sobre las distintas opciones entre las que se puede optar, motivaré porqué elegí el desarrollo nativo sobre la plataforma Android. A partir de ahí nos centraremos en los detalles concretos de dicha plataforma y explicaré todos los pasos que he realizado para desarrollar la aplicación iBudget, desde el diseño conceptual, bocetado de la interfaz de usuario y finalmente la implementación del código.

iBudget es una aplicación que comencé a desarrollar hace unos dos años con el único objetivo de aprender a programar para Android. El desarrollo de las primeras versiones lo realice sin seguir ninguna metodología ni patrones de arquitectura software, tampoco utilice técnicas de diseño centrado en el usuario. El resultado de esto fue una aplicación funcional pero con muchas carencias, tanto en la estructura del código como en la interfaz de usuario, también carecía de algunas funcionalidades importantes. Así decidí aprovechar el TFG para rehacer la aplicación, planteándome el seguir una metodología de desarrollo, reestructurar, hacer refactorización de código y aplicar técnicas de diseño centrado en el usuario. También he aprovechado para añadir funcionalidades que me parecen importantes. Esto tuvo como resultado un aumento en el número de descargas de la aplicación y una mejor valoración por parte de los usuarios.

Me gustaría destacar que no hice la aplicación siguiendo ningún tutorial que utilizase esa idea como caso de estudio, la idea fue propia y los libros y demás documentación que utilice fueron solo para aprender como se programa para Android.

En mi caso no disponía de ningún conocimiento previo sobre esta plataforma así que empecé adquiriendo (El Gran Libro de Android) publicado por Jesús Tomás profesor de la UPV. Me fue muy útil para empezar desde cero y es algo que aconsejo a cualquiera que desee iniciarse en el desarrollo de aplicaciones para una plataforma que le sea totalmente desconocida. Un libro bien estructurado te proporciona una visión general y la base necesaria para empezar el trabajo de desarrollo, una vez tienes esa base y comprendes en líneas generales las peculiaridades de la plataforma elegida puedes recurrir a los foros y ejemplos on-line que siempre están más actualizados y tratan con más detalle cada uno de los problemas concretos que te van surgiendo.

Al final obtuve como resultado una aplicación sólida y con mucha funcionalidad en la que todavía continuo trabajando y ampliando

Puesto que todavía sigo trabajando en la aplicación, al final de este trabajo presentaré unas posibles mejoras y/o ampliaciones, algunas de ellas me dará tiempo a realizarlas a la vez que escribo esta memoria y otras simplemente las detallaré pero posiblemente no tendré tiempo de implementarlas.

1.3 Metodología y plan de trabajo

Puesto que el objetivo del proyecto ha sido mejorar y ampliar una aplicación ya existente, he decido dividir el trabajo en 3 etapas bien diferenciadas que paso a enumerar:

1. Rediseñar la interfaz de usuario. He rediseñado completamente todas las pantallas de la aplicación. Para esto he empleado la técnica persona, intentando conseguir así una interfaz más intuitiva, atractiva y usable. También he cambiado la forma de navegar entre dichas pantallas basándome en la técnica persona y el diseño de usabilidad.
2. Rediseñar la lógica de la aplicación. Con esto he pretendido obtener un código más legible y más ordenado, así como usar de forma más correcta las estructuras de datos que ofrece Java. He utilizado técnicas de modelado orientado a objetos. También he intentado conseguir que sea más sencillo ampliar el código para ofrecer nuevas funcionalidades.
3. Añadir nuevas funcionalidades. Al utilizar la técnica persona he identificado nuevas necesidades de usuario que sería interesante implementar. En el apartado de conclusiones explicaré brevemente cuales son estas nuevas funcionalidades.

He seguido una metodología de desarrollo ágil, iterando sobre cada uno de estos puntos hasta obtener el resultado esperado. En cada una de las iteraciones he incluido una fase de planificación, análisis de requisitos, codificación y pruebas. Las pruebas las he realizado manualmente, probando cada una de las funcionalidades de la aplicación e intentando emular todas las posibles acciones que puede realizar un usuario. La parte de pruebas no la incluiré en la memoria.

2. Contexto tecnológico

2.1 Contexto tecnológico actual de las aplicaciones para móviles

Aunque este trabajo se centra en una aplicación nativa para Android me gustaría tratar brevemente los distintos tipos de aplicaciones que existen para teléfonos móviles. El motivo de esto es que actualmente estoy realizando unas prácticas de empresa en SAP Dublín y me he dado cuenta de lo importante que es a nivel comercial elegir que herramientas y tecnologías son las más adecuadas a la hora de desarrollar aplicaciones, tanto es así que SAP optó por invertir mucho dinero y esfuerzos en desarrollar SAPUI5 (SAP User Interface for HTML5) que es un conjunto de librerías basadas en JavaScript y HTML5 que los desarrolladores pueden utilizar para crear aplicaciones web de escritorio y para móviles. Una características importante de las aplicaciones hechas con UI5 es que son responsive es decir, se adaptan automáticamente a los diferentes tamaños de pantalla y además pueden convertirse en aplicaciones hibridas utilizando un plugin de PhoneGap, las aplicaciones hibridas tienen una serie de ventajas de las que hablaré más adelante. Junto con esto también han desarrollado SAP Fiori que son unas guias de diseño de interfaz de usuario, UI5 proporciona muchas herramientas para facilitar el desarrollo de aplicaciones que cumplan esas guías, con esto se asegura que todas las aplicaciones de SAP proporcionen una misma experiencia de usuario. La interfaz de usuario resultante se adapta con facilidad a tamaños de pantalla tan diversos como los de móviles, tabletas y ordenadores de escritorio.

2.1.1 Comparativa de las principales plataformas móviles

A continuación maestro una tabla en la que comparo las principales plataformas disponibles para el desarrollo de aplicaciones para móviles

	 **Android**	 **Apple**	 **Windows phone**	 **Blackberry**
Núcleo del SO	Linux	Mac OS X	Windows	QNX
Licencia de software	Libre y abierto	Propietaria	Propietaria	Propietaria
Lenguaje de programación	Java y C	Objective C y Swift	C#, Visual Basic y C++	C, C++ y Java
IDE de desarrollo	Eclipse y Android Studio	Cocoa	Visual Studio	Eclipse
Tienda de aplicaciones	Google play	App Store	Windows Marketplace	Blackberry World
Coste de publicar aplicaciones	$25 una sola vez	$99 al año	$99 para empresas y $19 para cuentas individuales	Sin coste

Tabla 1. Comparativa de las principales plataformas móviles

2.1.2 Tipos de aplicaciones para móviles

A la hora de desarrollar aplicaciones para móviles lo primero que tenemos que decidir es si queremos desarrollar una aplicación nativa, hibrida o una aplicación web optimizada para móviles. En la siguiente figura muestro un breve resumen de las características de cada una de ellas y a continuación explicaré más en profundidad las ventajas e inconvenientes de cada una de las tres opciones.

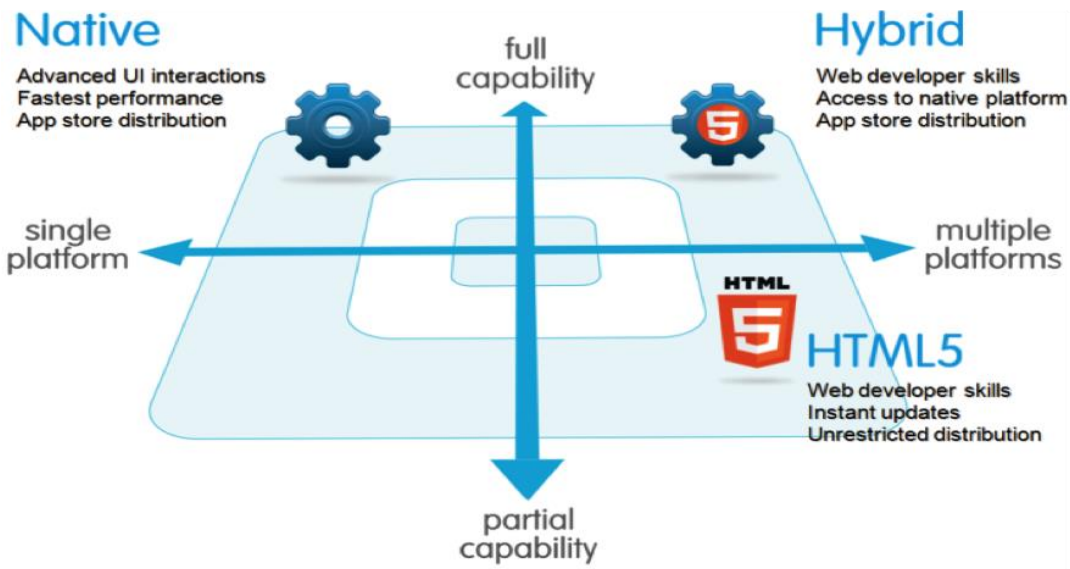


Figura 1. Muestra los tres tipo de aplicaciones para móviles que se utilizan en la actualidad

2.1.2.1 Aplicaciones nativas

Un ejemplo es el caso de estudio utilizado en este proyecto. Estas aplicaciones deben desarrollarse utilizando el lenguaje de programación y las herramientas que te permiten cada una de las plataformas.

- **Ventajas**: Puedes utilizar todas las funcionalidades de los SDK de cada una de las plataformas. Tienen mejor rendimiento y te permiten acceder a todos los recursos del dispositivo de forma nativa. Pueden publicarse en las tiendas de aplicaciones. Proporcionan una experiencia de usuario acorde con la plataforma sobre la que se utilizan.
- **Desventajas**: Solo pueden ejecutarse en la plataforma para la que se han desarrollado, el código no es reutilizable para otras plataformas. Se requieren diferentes habilidades/lenguajes de programación/herramientas para cada plataforma concreta.

2.1.2.2 Aplicaciones hibridas

Están experimentando un gran crecimiento en los últimos años. Imaginemos una empresa que necesita desarrollar una aplicación para todas las plataformas existentes. La primera fase sería realizar un diseño conceptual y bocetos de la interfaz de usuario abstrayéndose por el momento de la plataforma, pues bien, esto sería lo único que podríamos reutilizar, el resto del trabajo debería hacerse una vez por cada una de las plataformas que pretendamos alcanzar. Esto supone poner a trabajar a cuatro equipos diferentes de desarrolladores, cada uno de ellos con distintas habilidades y conocimientos concretos de cada plataforma. Y todavía peor, una vez hecha la aplicación nos quedaría el mantenimiento y las posteriores ampliaciones ya que raramente una aplicación permanece en su primera versión. Aquí nos encontraríamos con el problema de tener que mantener y ampliar una aplicación por cada una de las plataformas objetivo. Las aplicaciones hibridas aparecieron para solucionar este problema y presentan casi todas las ventajas de las aplicaciones nativas.

Estas aplicaciones se ejecutan sobre un contenedor nativo, esto es un punto de entrada a la aplicación exactamente igual que las nativas, que es llamado por el sistema operativo cuando el usuario inicia la aplicación, además también tienen la misma estructura de directorios que las aplicaciones nativas. Una vez tenemos esto hay dos enfoques diferentes:

A) Encapsular toda la aplicación en un visor web

El truco está en que una vez se inicia la aplicación se utiliza un Visor Web que es un componente nativo disponible en todas las plataformas y emula a un navegador web. A partir de ahí toda la aplicación puede desarrollarse utilizando las mismas herramientas que para la programación web como HTML 5, JavaScript, y CSS, con la ventaja de que podemos acceder a los recursos del dispositivo como GPS, contactos, almacenamiento interno, cámara, etc..

Hay varios frameworks que te permiten el desarrollo de este tipo de aplicaciones uno a destacar es PhoneGap que te permite el acceso a recursos locales del dispositivo a través de plugins, una ventaja de este framework es que si no existe un plugin para acceder a algún recurso que tu aplicación necesita hay mucha documentación a cerca de cómo crear tus propios plugins de una forma no excesivamente complicada.

- **Ventajas**: Tenemos una sola aplicación y un solo código que mantener, aunque hay que escribir un poco de código nativo para crear el contenedor se trata de unas pocas líneas y hay mucha documentación sobre como hacerlo sin necesidad de conocimientos concretos de cada plataforma. Tienes acceso a los recursos del dispositivo a través de plugins . Pueden publicarse en las tiendas de aplicaciones al igual que las nativas.
- **Desventajas**: No son tan eficientes como las aplicaciones nativas, aunque últimamente se está avanzando mucho en esto y según que tipo de aplicación sea no representa un problema. Se crean dependencias con los frameworks utilizados para el desarrollo, si no existe un plugin para acceder a alguna funcionalidad del dispositivo tienes que crearlo tu mismo o esperar a que alguien lo haga.

B) Desarrollar en un solo lenguaje y que este sea traducido a los lenguajes nativos de las diferentes plataformas

En este enfoque hay que destacar la plataforma Xamarin, se trata de una herramienta para los desarrolladores de aplicaciones móviles que te permite desarrollar toda la aplicación programando en C# y el código se traduce para que pueda ser ejecutado en Android, iOS y Windows Phone. Lo que hace Xamarin no es traducir el código escrito en C# a java o objective C , sino que lo compila de forma que pueda ser ejecutado en dichas plataformas. Las aplicaciones hechas con Xamarin pueden considerarse casi nativas.

- Xamarin.iOS utiliza una técnica conocida como AOT que es el acrónimo en inglés de Ahead Of Time, se trata de compilar un lenguaje de alto nivel como C# a código máquina. En este caso lo que hace Xmarin.iOS es traducir el código escrito en C# a ARM binario que es compatible con los dispositivos de Apple. A diferencia de la técnica conocida como JIT esta traducción se hace en tiempo de compilación de ahí el nombre Ahead Of Time.
- Xamarin.Android utiliza la técnica JIT que es el acrónimo de Just In Time, que consiste en compilar código intermedio a código máquina en tiempo de ejecución, esto por supuesto puede producir problemas de rendimiento.
- En cuanto a Windows Phone no tenemos ningún problema ya que por defecto se pueden programar aplicaciones nativas utilizando C#.

Xamarin puede incorporarse a Visual Studio y si quieres trabajar en un sistema operativo diferente a Windows puedes utilizar Xamarin Studio que está basado en Visual Studio. Al iniciar un proyecto debes elegir el tipo de plataforma Android/iOS/Windows Phone para la que quieres crear el proyecto. La interfaz de usuario debe de hacerse una vez para cada plataforma, lo que resulta en una interfaz totalmente nativa, Xamarin incorpora los Widgets disponibles en todas las plataformas. En cuanto al código de la lógica de negocio escrito en C# Xamarin te permite compartirlo entre proyectos creados para las diferentes plataformas, en media puedes compartir el 70% del código. Sigues teniendo que hacer una aplicación para cada plataforma pero la mayor parte del código es igual en todas ellas.

También tienes la opción de desarrollar la interfaz de usuario utilizando Xamarin.Forms, se trata de componentes o Widgets que Xamarin te proporciona para crear la interfaz de usuario, después Xamarin los convierte a componentes nativos de cada plataforma, el único problema es que no existen Xamarin.Forms para emular todos los componentes de las tres plataformas, pero si son suficientes para tu aplicación más del 90% del código que utilices será compatible con las tres plataformas.

Comparando Xamarin con el otro tipo de aplicaciones hibridas que utilizan el enfoque web encontramos las siguientes ventajas y desventajas:

- **Ventajas**: Podemos programar con C# que es un lenguaje muy sólido y eficiente. Puedes utilizar Visual Studio que es uno de los mejores entornos de programación que existe. La interfaz de usuario es nativa con lo que proporciona la misma experiencia de usuario que las aplicaciones nativas. Son más eficientes y proporcionan acceso al 100% de los recursos del dispositivo.

- **Desventajas**: Solo abarcan las tres plataformas mencionadas y las interfaces de usuario no son responsive. Tienes que crear una interfaz de usuario por cada plataforma, aunque en muchos casos basta con utilizar los Xamarin.Forms con lo cual solo creas una interfaz de usuario.

Comparando Xamarin con las aplicaciones nativas:

- **Ventajas**: Puedes llegar a reutilizar más del 90% del código.
- **Desventajas**: Puedes encontrarte con pequeños bugs que no sucederían en las aplicaciones nativas. Las nuevas API's de cada plataforma tardan algún tiempo en estar disponibles. Aunque están muy cerca no llegan a ser tan eficientes como las nativas debido a los enlaces y referencias que se tienen que producir entre el framework .NET y el sistema operativo que se quiera alcanzar y al hecho de tener que utilizar las técnicas AOT o JIT mencionadas anteriormente.

2.1.2.3 Aplicaciones Web

Se trata de aplicaciones web como las que utilizamos en nuestro ordenador de escritorio solo que optimizadas para teléfonos móviles. No pueden compararse con el tipo de aplicaciones tratadas en los puntos anteriores ya que no existe una instalación local, sino que se ejecutan en un servidor y se accede a ellas a través de una url. No requieren instalación, no pueden publicarse en las tiendas de aplicaciones y no pueden accede a los recursos del teléfono.

2.2 Conclusiones y motivación de porque elegí Android como plataforma

Una vez comparadas las diferentes plataformas existentes y los diferentes tipos de aplicaciones podemos hacernos una idea de lo difícil que resulta tomar una decisión a la hora de desarrollar una aplicación a nivel comercial. Normalmente las empresas cuentan con personas experimentadas en el desarrollo de aplicaciones para móviles y tabletas que les ayudan a decidir que enfoque tomar, idealmente debe de tratarse de profesionales con experiencia que hayan trabajado en proyectos tomando diferentes enfoques y se hayan topado con los pros y los contras de cada uno de ellos. Tomar una buena decisión en este sentido resulta crucial ya que una vez escogido un enfoque dar marcha atrás supondría desperdiciar casi todo el trabajo hecho hasta ese momento.

En mi caso al plantear este proyecto como un ejercicio docente la decisión fue más sencilla. La elección de hacer una aplicación nativa para Android fue algo natural ya que las principales herramientas de desarrollo las aprendemos en la universidad:

- Lenguaje de programación: Java
- IDE de desarrollo: Eclipse
- Base de datos: SQLite

Otras ventajas a destacar es que al ser herramientas abiertas y que se enseñan en todas o la mayoría de universidades del mundo hay una gran comunidad de desarrolladores que eligen Android como plataforma, esto hace que haya muchísima documentación online a parte de la documentación oficial que Google mantiene actualizada, infinidad de ejemplos en Github y foros muy activos en los que te resuelven las dudas. También es posible encontrar libros on-line gratuitos. En especial quiero destacar StackOverflow como uno de los mejores foros no solo para Android si no para resolver dudas sobre muchas otras plataformas y lenguajes de programación.

En los siguientes puntos me centro en la plataforma Android y en los pasos que seguí para desarrollar la aplicación iBudget.

2.3 La plataforma Android

Podemos definir una plataforma de desarrollo como un entorno software común en el cual se realiza la programación de un grupo definido de aplicaciones, todas ellas con ciertas características comunes impuestas, precisamente por dicha plataforma. Suelen estar ligadas a un sistema operativo, a un lenguaje de programación y a unas interfaces de programación (API's por sus siglas en inglés).

2.3.1 El sistema operativo Android

Android está basado en linux y fue desarrollado inicialmente para dispositivos móviles por Android Inc. en Palo Alto, California, antes del año 2005. En Julio de 2005 la empresa fue adquirida por Google que es su actual propietario. En la siguiente tabla podemos ver las versiones actuales de Android, sus niveles de API y la cuota de distribución actual de cada una de las versiones en el mercado. Estos datos han sido obtenidos de la documentación oficial que Google pone a disposición de los desarrolladores. Los datos son muy recientes, fueron recogidos durante un periodo de 7 días que finalizo el 4 de abril de 2016 y no se muestran las versiones que a dicha fecha cuentan con una cuota de distribución menor del 0.1%. Como vemos Android 3.0 Honeycomb prácticamente a desaparecido del mercado, por lo que no se muestra en esta tabla.

Versión	Nombre comercial	API	Distribución
2.2	Froyo	8	0.1%
2.3 – 2.3.7	Gingerbread	10	2.6%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	2.2%
4.1.x	Jelly Bean	16	7.8%
4.2.x		17	10.5%
4.3		18	3.0%
4.4	KitKat	19	33.4%
5.0	Lollipop	21	16.4%
5.1		22	19.4%
6.0	Marshmallow	23	4.6%

Tabla 2. Cuota de distribución y nivel de api de las diferentes versiones de Android

A fecha de hoy Google no recomienda desarrollar para versiones de API inferiores a la 10. En los dispositivos que tengan instalada una determinada versión de Android solo se podrán utilizar las funcionalidades disponibles en su correspondiente versión de API e inferiores. La práctica general es configurar tu proyecto para utilizar la máxima versión de API disponible y utilizar las librerías de soporte que Google pone a disposición de los desarrolladores para que tu aplicación pueda ejecutarse en dispositivos con versiones de API inferiores.

2.3.2 Fundamentos de una aplicación Android

Una aplicación Android puede construirse a partir de cuatro componentes fundamentales. Cada uno de estos componentes representa un punto de entrada a través del cual el sistema puede interactuar con nuestra aplicación. No todos los componentes son puntos de entrada con los que pueda interactuar el usuario y algunos dependen unos de otros, pero cada uno juega su papel para definir el comportamiento general de nuestra aplicación. Tampoco es necesario implementar los cuatro componentes, esto dependerá de las necesidades de cada aplicación.

Hay cuatro componentes, cada uno de ellos tiene su propio ciclo de vida y sirve a diferentes propósitos:

1. **Activity**: Se implementa como una clase de Java heredando de Activity que es un componente presente en todas las versiones de API de Android. A partir de ahí puedes utilizar y sobre escribir todos los métodos de la clase Activity en tu clase Java. Generalmente una actividad representa una pantalla individual en tu aplicación y permite que tu clase despliegue una UI compuesta de vistas y responda a eventos. La navegación entre las diferentes pantallas se suele hacer (salvo que utilices fragmentos) iniciando nuevas actividades a través de los intentos.
2. **Services**: Un Servicio es una aplicación que se ejecuta en segundo plano por un largo periodo de tiempo y no despliega una UI. Por ejemplo un Servicio podría ser un programa que reproduce archivos mp3 desde una lista de música, mientras el usuario realiza otras actividades o podríamos utilizar un servicio para traer datos que la aplicación necesite desde internet sin bloquear la interfaz de usuario. Otros componentes como las actividades pueden iniciar servicios y dejarlos corriendo en segundo plano o interactuar con ellos. Igual que sucede con las actividades podemos usarlos heredando de la clase Service.
3. **Content providers**: Los proveedores de contenido se utilizan para compartir datos entre aplicaciones. A través de ellos varias aplicaciones pueden acceder a un conjunto de datos e incluso modificarlos, siempre que tengan permiso para hacerlo. Se pueden implementar heredando de ContentProvider y deben implementar un conjunto estándar de API's que permiten a otras aplicaciones realizar transacciones.

4. **Broadcast receivers**: Son componentes destinados a recibir y responder ante eventos globales generados por el sistema, como avisos de batería baja, una llamada entrante, etc. También pueden capturar y responder a eventos producidos por otras aplicaciones. No disponen de una interfaz de usuario aunque pueden crear notificaciones en la barra de acciones. Se implementan heredando de la clase BroadcastReceiver.

Tres de estos cuatro componentes (las actividades, los servicios y los receptores de difusiones) se activan a través de un mensaje asíncrono llamado Intent. Los intentos sirven para unir estos tres componentes diferentes en tiempo de ejecución, uno de sus usos principales es iniciar y terminar actividades.

Un aspecto a destacar del diseño de Android y que facilita mucho las cosas a los desarrolladores es que cualquier aplicación puede iniciar un componente de cualquier otra aplicación instalada en el dispositivo. Esto se traduce en que si por ejemplo queremos que nuestra aplicación haga una fotografía, seguramente ya tendremos otra aplicación instalada que realice dicha función, así no necesitamos implementar código para interactuar con la cámara del dispositivo, si no que basta con llamar a la aplicación que realiza dicha función. Esto podemos hacerlo utilizando un intento para desde nuestra aplicación iniciar la actividad que hace una foto en la aplicación que maneja la cámara. Sin embargo por motivos de seguridad para que nuestra aplicación pueda hacer esto deberemos especificar un permiso en el manifiesto.

Algo muy importante a la hora de empezar a programar es conocer el ciclo de vida de las actividades, ya que en más de una ocasión necesitaremos sobre escribir los métodos que son llamados cuando la actividad pasa por cada una de las diferentes fases desde su creación hasta su destrucción. En la siguiente imagen muestro el ciclo de vida de una actividad.

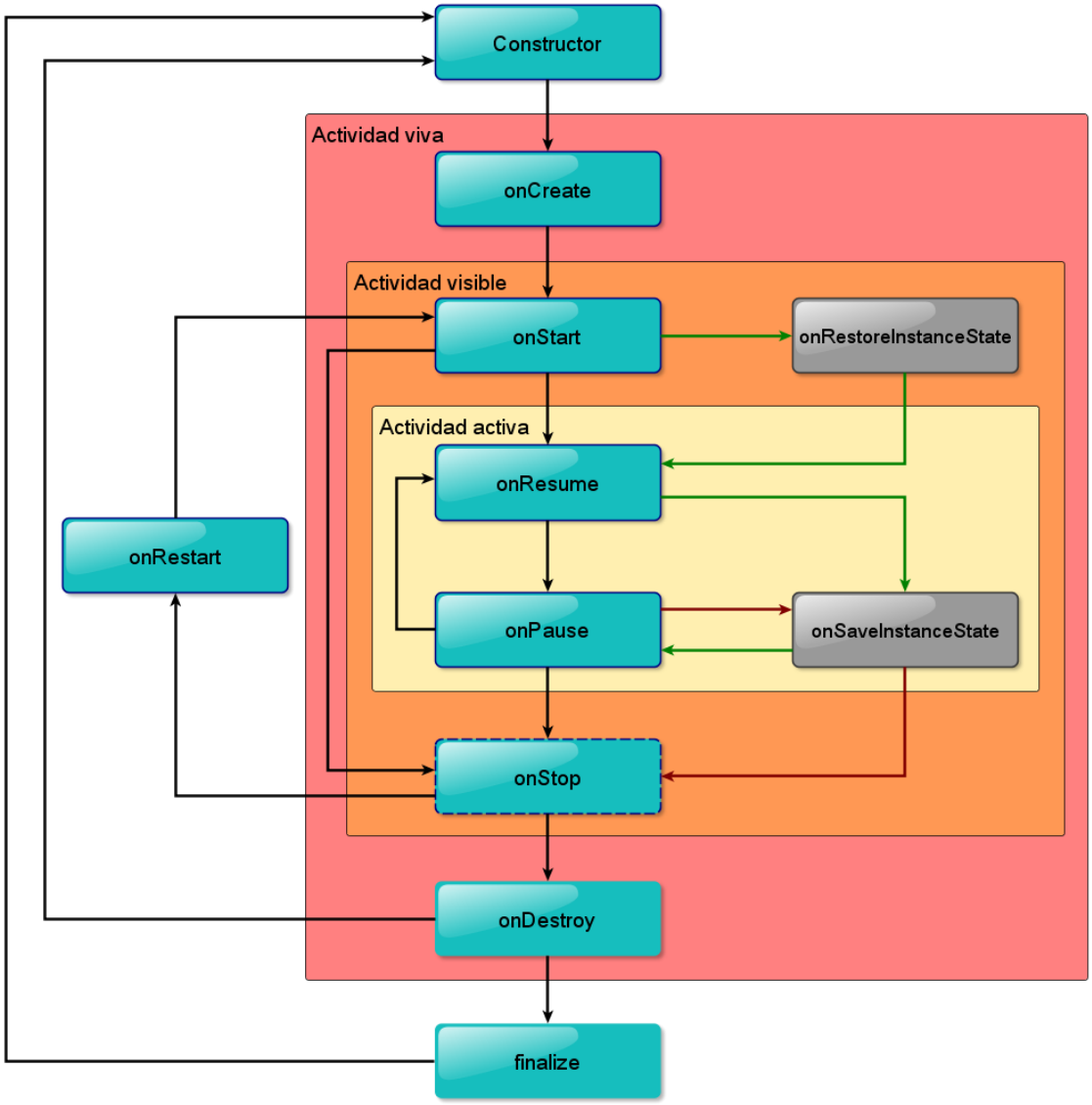
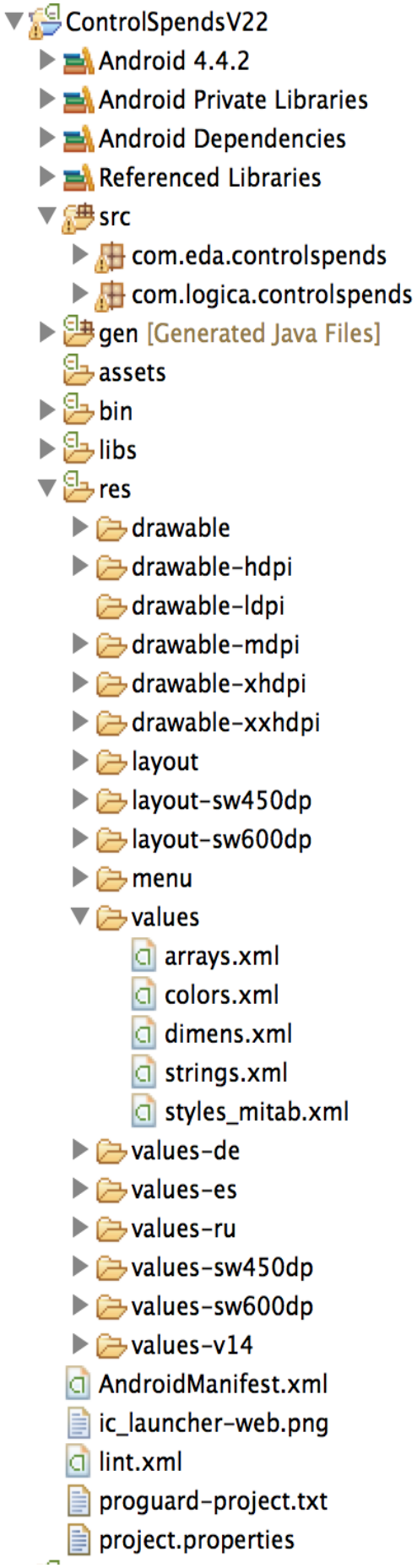


Figura 2. Ciclo de vida de la clase Activity

2.3.3 Estructura de directorios de una aplicación Android

En un principio Google eligió eclipse como entorno de desarrollo, pero posteriormente creo Android Studio y a fecha de hoy es el framework oficial que recomienda para el desarrollo de aplicaciones, sin embargo todavía sigue actualizando y manteniendo el SDK para eclipse. Actualmente muchos desarrolladores se han pasado a Android Studio, pero yo he preferido utilizar eclipse debido a que en la universidad he trabajado mucho con este framework y me resulta muy familiar. A continuación muestro en una captura de pantalla la estructura de directorios que crea eclipse cuando seleccionas crear un proyecto Android.



src
Contiene el código escrito por el desarrollador en Java. En mi caso he creado dos sub directorios uno para las estructuras de datos y otro para la lógica de negocio

gen
Contiene código autogenerado por eclipse. Son clases java con identificadores para manejar cada uno de los recursos que vamos añadiendo en la aplicación como imágenes, vistas, degradados en xml, etc. Este código no se debe modificar manualmente.

res/drawable
Contiene recursos como imágenes y degradados o colores definidos en xml. Podemos usar los directorios con sufijo –xxx para poner recursos optimizados a las diferentes resoluciones de pantalla como hdpi, ldpi, mdpi, xhdpi y xxhdpi, cuando se instale la aplicación solo se instalará la carpeta drawable y aquella cuyo sufijo se corresponda con la resolución de pantalla del dispositivo

res/layout
Contiene la interfaz de usuario definida en archivos xml. Como en el caso anterior podemos utilizar sufijos, en este caso para manejar los diferentes tamaños de pantalla.

res/menú
Contiene los strings que aparecerán en los menus que definamos en la aplicación.

res/values
Se utiliza para definir diferentes valores y permite utilizar sufijos para diferenciar entre idiomas, tamaños de pantalla y niveles de API. Todo lo que sea por defecto debemos ponerlo en la carpeta values que es la que se cargará cuando las configuraciones del dispositivo no coincidan con ninguno de los sufijos utilizados.

res/values/strings.xml y **res/values/arrays.xml**
Aquí deberíamos poner todo el texto estático que aparezca en nuestra aplicación. Se usan sufijos para referenciar los diferentes idiomas, en mi caso la aplicación está traducida al alemán, ruso y español y el idioma por defecto es inglés.

res/values/colors.xml
Aquí definimos colores en formato hexadecimal

res/values/dimens.xml
Podemos definir diferentes dimensiones para los componentes visuales de la aplicación y utilizar los sufijos para decidir que archivo se cargará dependiendo del tamaño de pantalla del dispositivo

res/values/styles
Definimos estilos para los componentes visuales, estos estilos pueden depender del nivel de API que soporte cada dispositivo.

El archivo manifest
Es sumamente importante, en él debemos definir todas las actividades que utilicemos en la aplicación y especificar cual de ellas será el punto de entrada, es decir, la que se ejecutará cuando el usuario pulse el icono de la aplicación. También debemos definir los permisos que solicitamos para acceder a los recursos del teléfono. En el momento de la instalación se solicitará al usuario que acepte dichos permisos.

Trataré con más detalle alguno de estos puntos en el capítulo de implementación.

3. Caso de estudio: aplicación para el control de gastos personales

Cuando tienes en mente desarrollar una aplicación para cubrir una necesidad o resolver un problema específico sueles tener una idea general y bastante vaga de lo que realmente quieres hacer. Un buen punto de partida es ver las opciones ya disponibles en el mercado para intentar descubrir posibles nichos que dichas aplicaciones no cubran. Actualmente existen muchas aplicaciones orientadas a las finanzas personales, algunas de ellas son muy completas y cuentan con interfaces de usuario muy atractivas, pero al principio pueden ser difíciles de utilizar para personas poco experimentadas en las nuevas tecnologías, otras son más simples pero no cuentan con una buena interfaz de usuario y ofrecen poca funcionalidad. Así yo opté por un enfoque intermedio, priorizando la facilidad de uso pero sin sacrificar funcionalidades que considero importantes. Una vez tuve claro esto comencé con el diseño de la aplicación dividiéndolo en dos partes, diseño de la interfaz de usuario y diseño de la lógica de la aplicación. A continuación desarrollo cada una estas partes.

3.1 Análisis de necesidades de usuario

Puesto que hace poco curse la asignatura Desarrollo Centrado en el Usuario, decidí utilizar la técnica persona para obtener tanto las funcionalidades que la aplicación debía ofrecer como los diseños de la interfaz de usuario.

Además hace unas pocas semanas tuvimos un curso sobre diseño de interfaces en SAP Irlanda y la primera parte del curso consistió en crear una Persona y escenarios para un caso de uso que nos dieron de ejemplo, la forma de hacerlo que nos explicaron era muy similar a como se ve en la asignatura Desarrollo Centrado en el Usuario.

3.1.1 Persona

La técnica Persona consiste en crear un usuario ficticio que represente a un grupo de personas o colectivo al que va dirigido tu aplicación. Esto ayuda a empatizar con dicho colectivo, lo que a su vez te permite tomar mejores decisiones a la hora de diseñar la interfaz de usuario y especificar los requisitos de la aplicación que te has propuesto desarrollar. Existen varias tipologías de Persona, la más utilizada y la única que voy a utilizar en este trabajo es la persona primaria. La persona primaria debe representar al grupo de usuarios a los que va dirigida la aplicación, en mi caso son usuario con un nivel socioeconómico medio-bajo y con habilidades medias o bajas en el uso de nuevas tecnologías.

Mónica Vidal

Biografía

- Tiene 34 años
- Es extrovertida y le gusta el trato con la gente
- Nació y vive en valencia en un piso situado en las afueras que comparte con una amiga
- Mónica es soltera y no tiene hijos
- Trabaja de dependienta en una zapatería situada en el centro de Valencia
- Nivel de estudios medio, no llego ha estudiar una carrera universitaria
- No le entusiasman las nuevas tecnologías, sin embargo si que utiliza habitualmente su Smartphone, en especial WhatsApp para comunicarse con sus amigos así como redes sociales, principalmente Facebook
- En su casa utiliza un ordenador portátil principalmente para navegar por internet con el objetivo de planificar algún viaje y eventualmente entra en su Facebook
- Como trabaja en el centro de vez en cuando aprovecha para quedar a tomar algo con sus amigos al salir de trabajar y el fin de semana suele ir a visitar sus padres y quedar con amigos para realizar salidas nocturnas, ir al cine o a algún concierto
- Tiene tarjeta de crédito pero para compras menores prefiera pagar en efectivo
- Le gusta salir a correr entre 2 y 3 días a la semana y viajar durante sus vacaciones

Objetivos

- Aunque es consciente de que con su sueldo en la zapatería no puede ahorrar mucho dinero si que le gustaría tener unos pequeños ahorros para el futuro y al mismo tiempo poder planear más viajes con sus amigos
- No suele comprar cosas caras y el compartir piso le ayuda a ahorrar, pero tiene una vida social bastante activa y suele gastar dinero en pequeñas cosas pero de forma bastante frecuente. A Mónica le gustaría tener un mayor control de en que gasta su dinero y así poder planificar mejor la forma de ahorrar
- Próximamente tiene pensado realizar un curso de liderazgo y gestión de personal con el objetivo de poder ascender en su trabajo
- Esta planificando hacer un viaje con una amiga el próximo verano por lo que necesita ahorrar algo de dinero extra, así como encontrar un destino que no sea excesivamente caro

3.1.2 Escenarios y bocetos de la interfaz de usuario

Escenario general: Primer contacto con la aplicación

Mónica está descansando en el sofá de su casa, tiene su Smartphone y está mirando las fotos de un viaje que una amiga ha puesto en Facebook. Le viene a la cabeza el viaje que está planeando para este verano y que no lleva mucho dinero ahorrado. A Mónica le gustaría saber en que se le va el dinero. De repente se le ocurre entrar a Google Play y buscar una aplicación para el control de gastos. Encuentra iBudget y se la descarga. A primera vista le parece sencilla e intuitiva de modo que empieza a navegar entre las cinco pantallas principales de la aplicación, la forma de hacerlo le resulta sencilla y rápidamente es capaz de deducir lo que puede hacer en cada una de ellas. A Mónica le parece una aplicación práctica y fácil de usar por lo que se propone utilizarla para controlar mejor sus gastos.

Escenario general: Introducir un gasto/ingreso

Mónica acaba de salir de trabajar y está con unos amigos tomando una cerveza y unas tapas en un bar cercano a su trabajo. Ya han terminado y cada uno se dispone a pagar su parte, Mónica recuerda que ahora tiene una aplicación que le ayuda a controlar y administrar mejor su dinero, así que saca su Smartphone abre la aplicación iBudget y en menos de 30 segundos introduce el gasto que acaba de tener.

Escenario general: Visualizar listados de gastos/ingresos

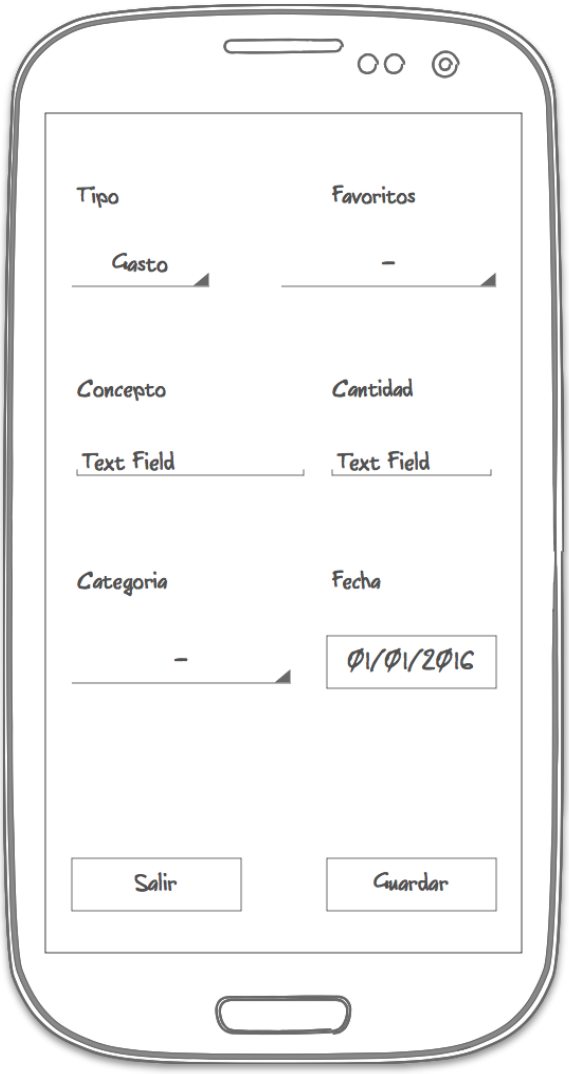
Mónica va en el metro de vuelta a su casa, le han propuesto quedar a cenar ese fin de semana pero piensa que quizás sería mejor quedarse en casa y ahorrar un poco más de dinero para su viaje. Abre la aplicación iBudget e inmediatamente puede ver cuanto lleva gastado este mes y en que lo ha gastado. Ve que sus gastos están siendo algo más elevados de lo planeado, así que decide que este fin de semana se quedará en casa.

Escenario general: Introducir objetivos de gasto

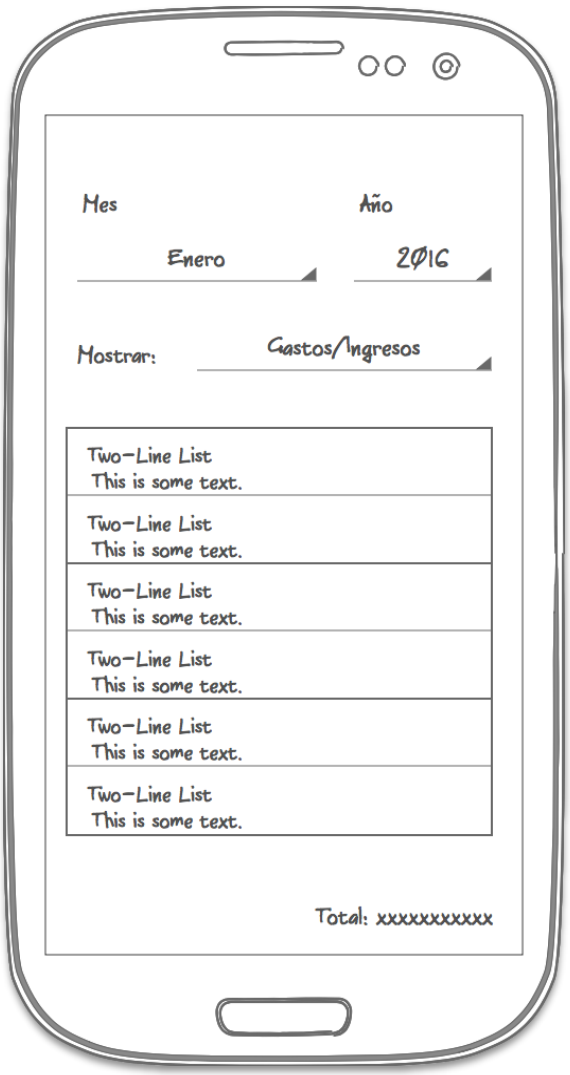
Mónica está en casa repasando sus cuentas mensuales, desafortunadamente no está consiguiendo ahorrar todo lo que le gustaría. Se da cuenta de que cada vez que queda con sus amigos gasta mucho en cenas y copas, así que decide poner un objetivo de gasto de 100 euros al mes en estos conceptos, esto le ayudará al mismo tiempo a realizar otras actividades más sanas.

A partir de estos escenarios e incluyendo algunas ideas propias he diseñado los siguientes bocetos utilizando la herramienta ***ninjamock*** que es muy adecuada para una primera fase de bocetado, en la que los bocetos son todavía bastante abstractos.

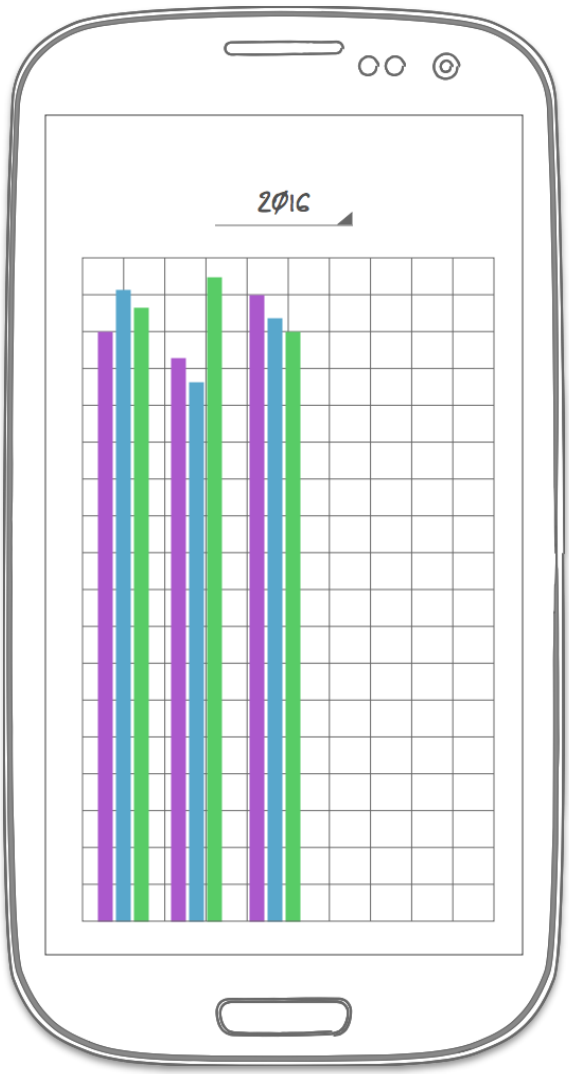
Pantalla para introducir gasto/ingreso



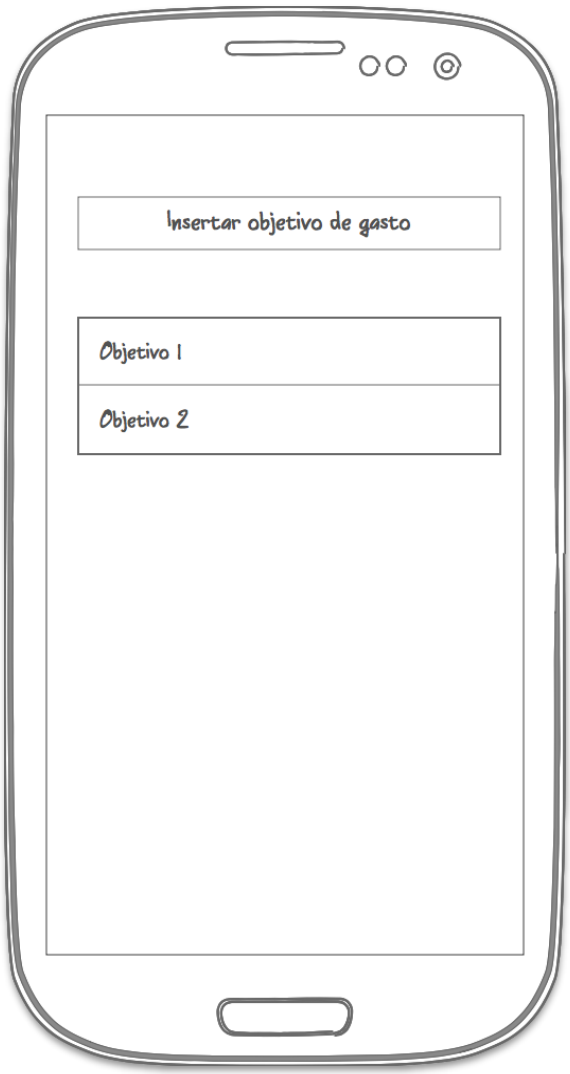
Pantalla para ver los listados de gastos/ingresos



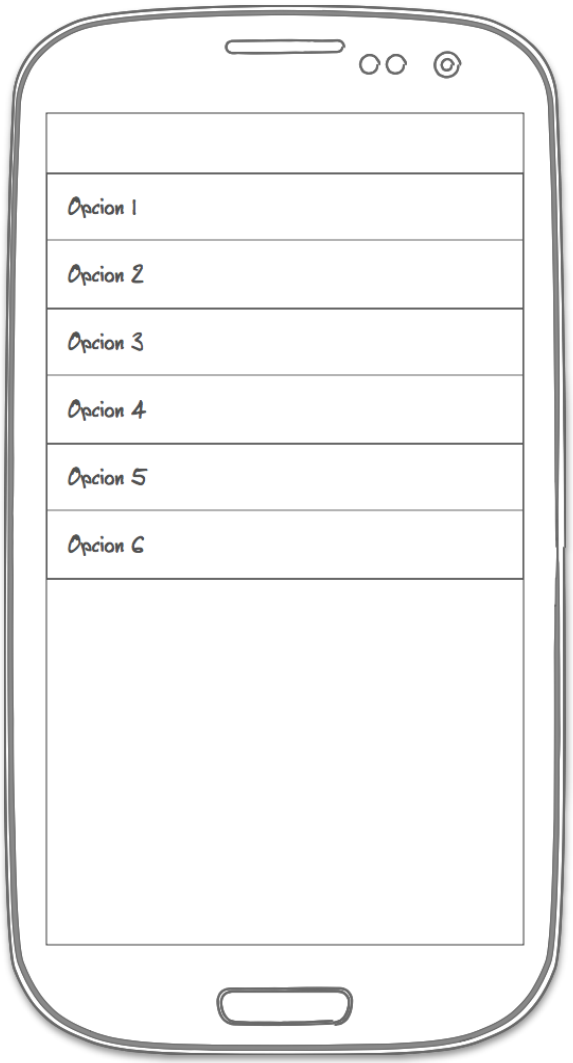
Pantalla con una gráfica anual de gastos/ingresos



Pantalla para introducir y ver objetivos de gasto



Pantalla con diferentes opciones



Escenario intermedio: Primer contacto con la aplicación

Mónica está descansando en el sofá de su casa, tiene su Smartphone y está mirando las fotos de un viaje que una amiga ha puesto en Facebook. Le viene a la cabeza el viaje que está planeando para este verano y que no lleva mucho dinero ahorrado. A Mónica le gustaría saber en que se le va el dinero. De repente se le ocurre entrar a Google Play y buscar una aplicación para el control de gastos. Encuentra iBudget y se la descarga. A primera vista le parece sencilla e intuitiva y comienza a navegar entre las diferentes pantallas deslizando el dedo sobre la pantalla del teléfono, esto le recuerda a la forma de navegar por las pantallas de WhatsApp una aplicación que utiliza muy a menudo. Así explora las cinco pantallas principales de la aplicación. A Mónica le parece una aplicación práctica y sencilla de usar por lo que se propone utilizarla para controlar mejor sus gastos.

Escenario intermedio: Introducir un gasto/ingreso

Mónica acaba de salir de trabajar y está con unos amigos tomando una cerveza y unas tapas en un bar cercano a su trabajo. Ya han terminado y cada uno se dispone a pagar su parte, Mónica recuerda que ahora tiene una aplicación que le ayuda a controlar y administrar mejor su dinero, así que saca su Smartphone y abre la aplicación iBudget, la primera pantalla que le aparece es la de introducir gastos/ingresos, introduce rápidamente los datos correspondientes al gasto que acaba de realizar, presiona el botón guardar y sale de la aplicación.

Escenario intermedio: Visualizar listado de gastos/ingresos

Mónica va en el metro de vuelta a su casa, le han propuesto quedar a cenar ese fin de semana pero piensa que quizás sería mejor quedarse en casa y ahorrar un poco más de dinero para su viaje. Abre la aplicación iBudget y va a la pantalla con el título "Entradas" donde puede ver un listado detallado de todos los gasto del mes ordenados por fecha, así como el balance de gastos/ingresos. También puede ver un resumen de cuanto ha gastado en cada concepto y en cada categoría que le ayuda a identificar cuales son sus principales gastos. Como ve que este mes ya ha gastado algo más de lo previsto decide quedarse en casa el sábado por la noche, de ese modo podrá ahorra más dinero y de paso salir a correr el domingo por la mañana.

Escenario intermedio: Introducir objetivos de gasto

Mónica está en casa repasando sus cuentas mensuales, desafortunadamente no está consiguiendo ahorrar todo lo que le gustaría. Se da cuenta de que cada vez que queda con sus amigos gasta mucho en cenas y copas, así que decide poner un objetivo de gasto de 100 euros al mes en estos conceptos, esto le ayudará al mismo tiempo a realizar otras actividades más sanas. Abre la aplicación y crea una nueva categoría que llamará "Cenas y Copas" que englobará todos estos gastos, seguidamente se va a la pantalla "Objetivos" y presiona el botón "Insertar objetivo de gasto", inmediatamente se abre un dialogo donde define dicho objetivo. Una vez rellenados los campos correspondientes pulsa "Aceptar" lo que le lleva de nuevo a la pantalla "Objetivos" donde ahora puede ver los detalles de dicho objetivo así como una barra de progreso que irá avanzando según vaya introduciendo gastos en la categoría "Cenas y Copas".

Escenario específico: Primer contacto con la aplicación

Mónica está descansando en el sofá de su casa, tiene su Smartphone y está mirando las fotos de un viaje que una amiga ha puesto en Facebook. Le viene a la cabeza el viaje que está planeando para este verano y que no lleva mucho dinero ahorrado. A Mónica le gustaría saber en que se le va el dinero. De repente se le ocurre entrar a Google Play y buscar una aplicación para el control de gastos. Encuentra iBudget y se la descarga. A primera vista le parece sencilla e intuitiva. Empieza a recorrer las principales pantallas deslizando su dedo de derecha a izquierda sobre el teléfono, esto le recuerda la forma de navegar entre las pantallas de WhatsApp una aplicación que utiliza muy a menudo. La pantalla inicial de la aplicación es un formulario que le permite introducir gastos/ingresos, crear una lista de categorías en las que posteriormente se agruparán las entradas así como una lista de entradas frecuentes, seguidamente ve una pantalla que le permite ver un listado de todos los gastos/ingresos del mes en curso, permitiendo seleccionar diferentes meses y años, después encuentra una pantalla con una gráfica inicialmente vacía que le permitirá contrastar visualmente los gastos/ingresos de los diferentes meses del año en curso. A continuación una pantalla con el título "Objetivos" y un botón donde lee "Insertar objetivo de gasto", y por último la pantalla opciones done puede ver una lista con diferentes opciones interesantes como crear entradas recurrentes, exportar a Excel, etc.. A Mónica le parece una aplicación práctica y sencilla de usar por lo que se propone utilizarla para controlar mejor sus gastos.

Escenario específico: Introducir un gasto/ingreso

Mónica acaba de salir de trabajar y está con unos amigos tomando una cerveza y unas tapas en un bar cercano a su trabajo. Ya han terminado y cada uno se dispone a pagar su parte, Mónica recuerda que ahora tiene una aplicación que le ayuda a controlar sus gastos, así que saca su Smartphone y abre la aplicación iBudget, la primera pantalla es un formulario que le permite introducir gastos/ingresos, en el campo concepto escribe merienda con los amigos, en el campo cantidad pone 4 euros y selecciona la categoría Bar/Restaurante como categoría de dicho gasto, seguidamente presiona el botón guardar y el botón salir para cerrar la aplicación.

Escenario específico: Visualizar listado de gastos/ingresos

Mónica va en el metro de vuelta a su casa, le han propuesto quedar a cenar ese fin de semana pero piensa que quizás sería mejor quedarse en casa y ahorrar un poco más de dinero para su viaje. Abre la aplicación iBudget y va a la pantalla con el título "Entradas" donde puede ver un listado detallado de todos los gasto del mes ordenados por fecha, así como el balance de gastos/ingresos. En la parte superior de la pantalla tiene 3 listas desplegables para seleccionar el mes y el año que quiera visualizar y otra para seleccionar el tipo de listado , por defecto están seleccionados el mes y año en curso y la opción de listado detallado de los correspondientes gastos/ingresos. Como ve que este mes ha gastado más de lo que pensaba selecciona la opción ver resumen de gastos por concepto y se da cuenta que ha gastado mucho en salidas con los amigos así que decide quedarse en casa el sábado por la noche, de ese modo podrá ahorra más dinero y de paso salir a correr el domingo por la mañana.

Escenario específico: Introducir objetivo de gasto

Mónica está en casa repasando sus cuentas mensuales, desafortunadamente no está consiguiendo ahorrar todo lo que le gustaría. Se da cuenta de que cada vez que queda con sus amigos gasta mucho en cenas y copas, así que decide poner un objetivo de gasto de 100 euros al mes en estos conceptos, esto le ayudará al mismo tiempo a realizar otras actividades más sanas. Abre la aplicación y crea una nueva categoría que llamará "Cenas y Copas" que englobará todos estos gastos, seguidamente se va a la pantalla "Objetivos" y presiona el botón "Insertar objetivo de gasto", inmediatamente se abre un dialogo que le permite definir el objetivo, escoge no gastar más de 100 euros en la categoría "Cenas y Copas", por último define la fecha inicio y fin de dicho objetivo. Una vez rellenados estos campos pulsa "Aceptar" lo que le lleva de nuevo a la pantalla "Objetivos" donde ahora puede ver los detalles de dicho objetivo así como una barra de progreso que irá avanzando según vaya introduciendo gastos en la categoría "Cenas y Copas".

Con los escenarios intermedio y específico definidos ya tenemos en mente una idea más concreta de lo que queremos hacer, lo que nos permite realizar unos bocetos más específicos de la aplicación. Estos segundos bocetos los he realizado con la herramienta Justinmind ya que la utilizamos en la asignatura Desarrollo Centrado en el Usuario y me gustó la forma de trabajar con ella. Justinmind te permite realizar bocetos para una aplicación web que se vaya a visualizar en ordenadores de escritorio, así como para aplicaciones nativas que se vayan a ejecutar en tabletas y teléfonos Android y iOS. Cuando creas un nuevo prototipo debes seleccionar una de estas opciones lo que te permite en tiempo de diseño utilizar vistas o Widgets propios de la plataforma seleccionada. En mi caso como tengo claro que la aplicación va a ser exclusivamente para Android he realizado el prototipo utilizando los Widgets disponibles en dicha plataforma, esto es de gran ayuda cuando posteriormente comienzas a implementar la interfaz de usuario. A continuación muestro los bocetos definitivos que utilizaré como guía para implementar la interfaz de usuario.

Pantalla para introducir gasto/ingreso

Pantalla para ver los listados de gastos/ingresos

Pantalla con una gráfica anual de gastos/ingresos

Pantalla para introducir y ver objetivos de gasto

Pantalla con diferentes opciones

3.2 Análisis de requisitos

Del análisis anterior y añadiendo algunas ideas propias he determinado que la aplicación debe ofrecer las siguientes funcionalidades mínimas:

- Introducir gastos e ingresos especificando para cada uno de ellos un concepto, una categoría, un importe y la fecha en la que se ha realizado el gasto o se ha producido el ingreso.
- Posibilidad de editar y eliminar las entradas previamente introducidas.
- El usuario debe poder visualizar los gastos e ingresos previamente introducidos en forma de listados que podrán obtenerse bien para un mes específico o para todo un año, así como el balance del mes o año seleccionados. Cada entrada del listado mostrará el concepto, la categoría, la fecha y el importe del correspondiente gasto o ingreso.
- Un resumen de gastos/ingresos agrupados por categoría. Dicho resumen podrá corresponder a un solo mes o a todo un año .
- Un resumen de gastos/ingresos agrupados por concepto. Igualmente podrán corresponder a un solo mes o a todo un año.
- Un gráfico anual de gastos/ingresos.
- El usuario podrá introducir objetivos de ahorro, del tipo no gastar más de una determinada cantidad en un concepto o categoría durante un cierto periodo de tiempo y visualizar el progreso de dicho objetivo.
- Se podrán programar gastos/ingresos recurrentes. Estos podrán ser diarios, semanales, quincenales, mensuales, bimestrales, trimestrales, semestrales o anuales. El usuario escogerá la fecha de inicio y fin de los gastos/ingresos recurrentes.
- Se podrá exportar a Excel los gastos/ingresos previamente introducidos, tanto de todo el periodo como seleccionando los correspondientes a un periodo concreto.
- Se ofrecerá la opción de solicitar una contraseña de inicio de sesión.
- Se podrá elegir un formato de fecha ya que usuarios de diferentes países pueden preferir formatos de fecha diferentes.
- Se podrá elegir entre crear respaldos de los datos almacenados manualmente o que la aplicación cree un respaldo automáticamente cada vez que se inserte un nuevo gasto/ingreso.
- Se podrá compartir y valorar la aplicación en Facebook.

Otras funcionalidades que considero interesantes al margen de las que se desprenden de los casos de uso expuestos anteriormente y que tendré en cuenta a la hora de realizar el diseño conceptual de la lógica de la aplicación son:

- El usuario podrá organizar sus gastos e ingresos en diferentes cuentas
- Se podrán adjuntar facturas escaneadas como comprobantes de cada uno de los gastos e ingresos

4. Modelado y diseño de la lógica de negocio

Aquí solamente voy a incluir un diagrama de clases propio de una primera fase de modelado UML donde muestro las principales clases que utilizaré en la aplicación y la relación que deben guardar entre ellas, la implementación en Java de dichas clases y el diseño de la base de datos.

4.1 Modelado orientado a objetos. Diagrama de clases

Un modelo conceptual trata de especificar detalladamente el problema del mundo real al que nos enfrentamos. Es el primer paso en el diseño de la lógica de una aplicación y está desprovisto de consideraciones de implementación. Posteriormente se realiza el diseño extendiendo y refinando el modelo previamente creado. El diseño ya considera tanto el entorno como el lenguaje de programación que se empleará en la implementación. El diagrama de clases es uno de los elementos esenciales en la fase de modelado. También se suelen utilizar diagramas de secuencia para derivar los métodos u operaciones de cada clase, sin embargo voy a omitir este paso ya que en mi aplicación las operaciones más interesantes se encuentran en las actividades que son los controladores encargados de manejar las vistas, y eso lo mostraré en la parte de implementación. En el diagrama de clases muestro solo los atributos de cada una de las clases ya que las operaciones que se pueden aplicar sobre dichas clases son los típicos getters y setters.

Como podemos ver en el diagrama de la siguiente página existen dos relaciones entre clases, la relación de especialización/generalización y la de composición.

- Relación de especialización/generalización: La generalización consiste en que dado un conjunto de clases que tengan en común atributos y métodos se puede sacar un factor común y crear una clase más general (superclase) a partir de las iniciales (subclases). Y la especialización es precisamente la relación inversa. En la práctica y en los lenguajes de programación orientados a objetos, esto se implementa utilizando la herencia.
- La relación de composición: Es un tipo específico de agregación, semánticamente significa “Esta compuesto por“. En este tipo de relación la eliminación del compuesto implica siempre la eliminación de las partes.

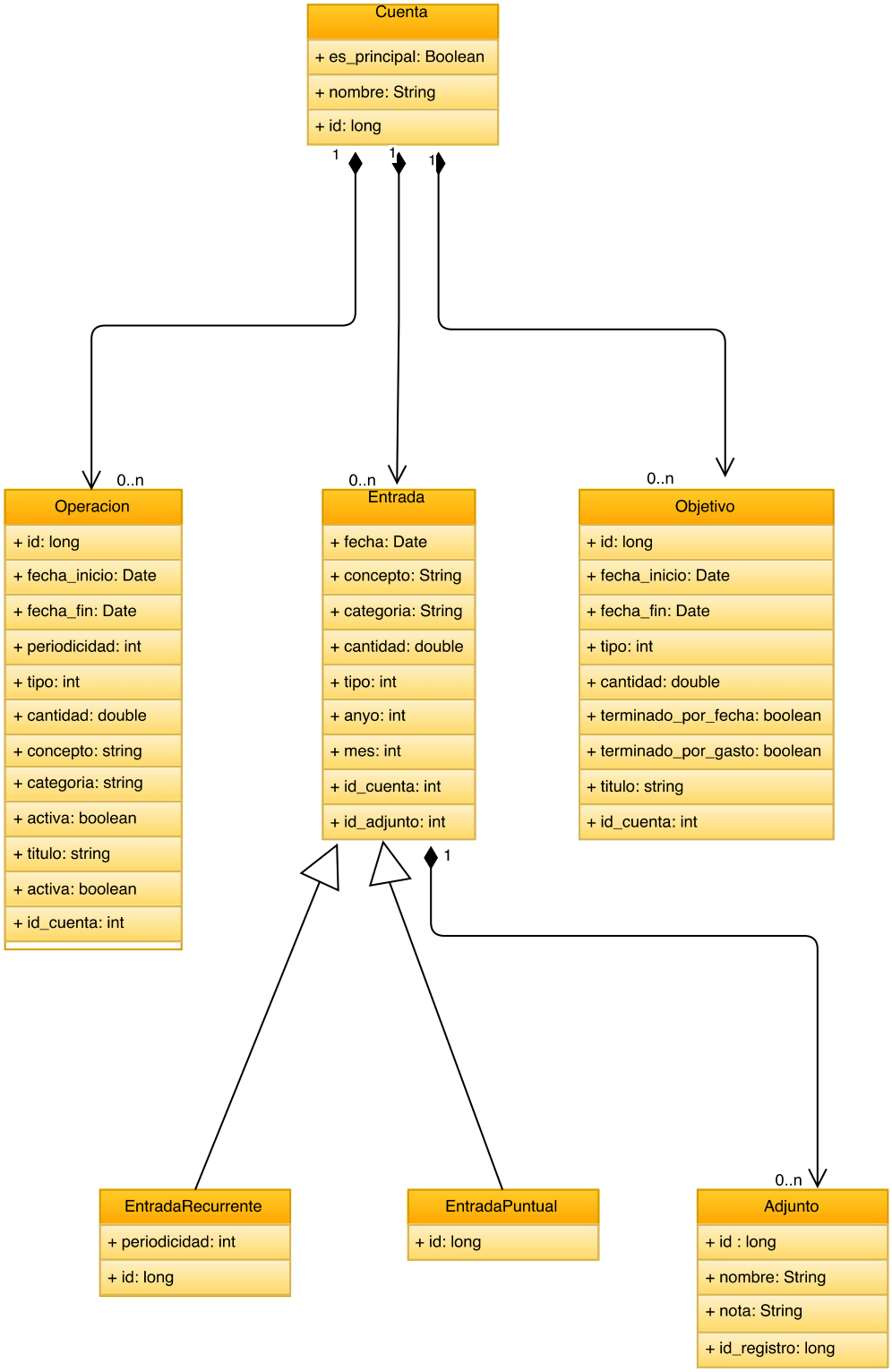


Figura 3. Diagrama de clases

El diagrama muestra 4 relaciones de composición, una cuenta está compuesta por operaciones (una operación sirve para programar entradas recurrentes), entradas y objetivos y a su vez cada entrada puede estar compuesta por un documento adjunto. La cardinalidad de esta relación es de 0 a N elementos, es decir una cuenta podría tener 0 o N entradas, objetivos y operaciones y una entrada puede tener de 0 a N documentos adjuntos.

Por otra parte la relación de especialización/generalización, nos indica que tanto las entradas recurrentes como las puntuales tiene atributos y métodos comunes que implementaremos en una superclase llamada Entrada.

4.2 Clases diseñadas en Java

Como salida del diagrama de clases obtenemos el diseño de las clases ya implementadas en un lenguaje de programación específico, en este caso Java. Al igual que en el diagrama omitiré los métodos de las clases ya que son los típicos getters y setters.

La clase Cuenta

Aquí bien podría haber optado por utilizar 3 estructuras de datos por ejemplo ArrayList que contuvieran las entradas, objetivos y operaciones correspondientes a cada cuenta. Sin embargo he optado por manejar esto directamente desde la base de datos, y asociar cada uno de estos objetos con su correspondiente cuenta utilizando el identificador de la clase cuenta, así cuando el usuario seleccione una determinada cuenta solo se le mostrarán las entradas, objetivos y operaciones cuyo id_cuenta sea igual al id de la cuenta seleccionada.

```
public class Cuenta {

	private boolean es_principal;
	private String nombre;
	private long id;

	public Cuenta(boolean es_principal, String nombre, long id){
		this.es_principal = es_principal;
		this.nombre = nombre;
		this.id = id;
	}

}
```

La clase Objetivo

```
public class Objetivo {
	private Date fechaInicio;
	private Date fechaFin;
	private String tipo;
	private String nomDeCatOCon;
	private double cantidadLimite;
	private boolean terminadoPorFecha ;
	private boolean terminadoPorGasto ;
	private long id;
	private long id_cuenta;

	public Objetivo(Date fI, Date fF, String tip, String nomDeCatOCon,
			double cantLim, long id_cuenta) {
		this.fechaInicio = fI;
		this.fechaFin = fF;
		this.tipo = tip;
		this.nomDeCatOCon = nomDeCatOCon;
		this.cantidadLimite = cantLim;
		this.id_cuenta = id_cuenta;
		this.terminadoPorFecha = false;
		this.terminadoPorGasto = false;
	}
```

La clase Operación

Esta clase se encarga de crear las entradas recurrentes según los parámetros que indique el usuario.

```
public class Operacion {
	private long id;
	private Date fechaInicio;
	private Date fechaFin;
	private String concepto;
	private String categoria;
	private int tipo;
	private double cantidad;
	private boolean activa;
	private int periodicidad;
	private String titulo;
	private long id_cuenta;
	private int identificador_antiguo;

	public Operacion(Date fechaInicio, Date fechaFin, String concepto,
			String categoria, int tipo, double cantidad, boolean activa,
			int periodicidad, String titulo) {
		this.fechaInicio = fechaInicio;
		this.fechaFin = fechaFin;
		this.concepto = concepto;
		this.categoria = categoria;
		this.tipo = tipo;
		this.cantidad = cantidad;
		this.activa = activa;
		this.periodicidad = periodicidad;
		this.titulo = titulo;
	}
```

La clase Entrada

Esta clase representa los gastos/ingresos que introduzca el usuario.

```
public class Entrada implements Comparable<Entrada>{
	private Date fecha;
	private String concepto;
	private String categoria;
	private int tipo;
	private double precio;
	private long id_cuenta;
	private int periodicidad;

	public Entrada(Date fecha, String concepto, String categoria, int tipo, double
precio, int periodicidad, long id_cuenta) {

		this.fecha = fecha;
		this.concepto = concepto;
		this.categoria = categoria;
		this.tipo = tipo;
		this.precio = precio;
		this.periodicidad = periodicidad;
		this.id_cuenta = id_cuenta;
	}
```

La clase EntradaPuntual

Representa los gastos ingresos que el usuario introducirá manualmente y es una especialización de la clase Entrada.

```
public class EntradaPuntual extends Entrada  {

	private long id;
	private boolean automatico;
	private long id_adjunto;
	private int anyo;
	private int mes;

public EntradaPuntual(Date f,String con, String cat, int tip, double pre, boolean
automatico, int periodicidad, long id_cuenta, long id_adjunto ){

		super(f, con, cat,tip, pre,periodicidad, id_cuenta);

		this.id_adjunto = id_adjunto;
		this.automatico = automatico;
	}
```

La clase EntradaRecurrente

Representa los gastos/ingresos recurrentes que el usuario haya definido, igualmente es una especialización de la clase Entrada.

```
public class EntradaRecurrente extends Entrada{

	private long id;
	private long id_operacion;

	public EntradaRecurrente(Date f,String con, String cat, int tip, double pre, int
periodicidad, long id_operacion, long id_cuenta){

		super(f, con, cat,tip, pre,periodicidad, id_cuenta);

		this.id_operacion = id_operacion;
	}
```

La clase Adjunto

Esta clase representa una fotografía que el usuario podrá adjuntar a cada gasto/ingreso junto con una nota.

```
public class Adjunto {
	private long id;
	private long entrada_id;
	private String nombre;
	private String nota;

	public Adjunto(long entrada_id, String nombre, String nota){
		this.entrada_id = entrada_id;
		this.nombre = nombre;
		this.nota = nota;
	}
```

4.3 El esquema de la base de datos

A partir del diseño de las clases es fácil deducir como serán las tablas en las que se almacenarán los objetos de dichas clases. Para proporcionar persistencia en Android tenemos SQLite, se trata de un motor para bases de datos SQL muy sencillo de utilizar como veremos en el punto de implementación. Aquí muestro el esquema de mi base de datos:

```
db.execSQL("CREATE TABLE cuentas (" +
        "_id INTEGER PRIMARY KEY," +
        " nombre TEXT," +
        " es_principal BOOLEAN)");
```

```
db.execSQL("CREATE TABLE objetivos (" +
        "_id INTEGER PRIMARY KEY," +
        " finicio long," +
        " ffin long," +
        "tipo TEXT, nomdecatocon TEXT," +
        " cantlimite DOUBLE," +
        " id_cuenta INTEGER)");
}
```

```
db.execSQL("CREATE TABLE EntradasPuntuales (" +
      "_id INTEGER PRIMARY KEY," +
      " fecha INTEGER," +
      " concepto TEXT," +
      " categoria TEXT," +
      " tipo INTEGER," +
      " precio DOUBLE," +
      " automatico BOOLEAN," +
      " periodicidad INTEGER," +
      " id_cuenta INTEGER," +
      " id_adjunto INTEGER," +
      " year INTEGER," +
      " month INTEGER)");
```

```
db.execSQL("CREATE TABLE EntradasRecurrentes (" +
      "_id INTEGER PRIMARY KEY," +
      " fecha INTEGER," +
      " concepto TEXT," +
      " categoria TEXT," +
      " tipo INTEGER," +
      " precio DOUBLE," +
      " periodicidad INTEGER," +
      " idoperacion INTEGER," +
      " id_cuenta INTEGER," +
      " id_adjunto INTEGER)");
```

```
db.execSQL("CREATE TABLE operaciones (" +
      "_id INTEGER PRIMARY KEY," +
      " fechaini INTEGER," +
      " fechafin INTEGER," +
      " concepto TEXT," +
      "categoria TEXT," +
      " tipo INTEGER," +
      " cantidad DOUBLE," +
      " activa BOOLEAN," +
      " periodicidad INTEGER," +
      " titulo TEXT," +
      "id_cuenta INTEGER," +
      " id_antigua INTEGER)");
```

```
db.execSQL("CREATE TABLE adjunto (" +
      "_id INTEGER PRIMARY KEY," +
      " nombre TEXT, id_registro INTEGER," +
      " id_cuenta INTEGER," +
      "id_nota TEXT)");
```

5. Diseño de usabilidad

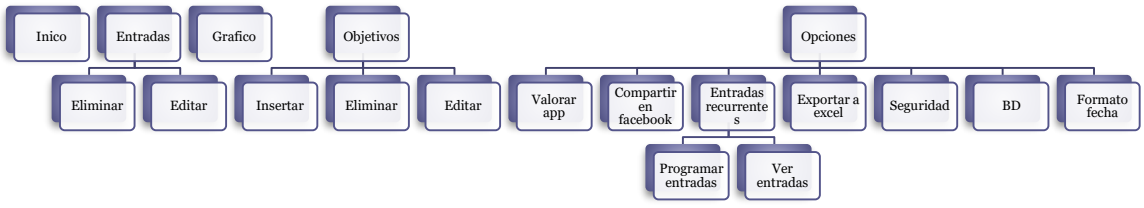
Una vez tuve claras las funcionalidades mínimas que quería ofrecer el siguiente reto fue como presentar todo esto al usuario, es decir como organizar la información y definir la forma de navegación entre las distintas pantallas de mi aplicación. Todo esto lo vimos en la asignatura DCU, concretamente en el Tema 3 de la asignatura y en el apartado Diseño de usabilidad, que cubre los siguientes aspectos clave:

- Arquitectura de la información
- Diseño de interacción
- Diseño gráfico
- Simplicidad

De los cuatro puntos solo voy a tratar aquí los dos primeros, que son los que me ayudaron a afrontar el reto que expongo en el párrafo anterior.

5.1 Arquitectura de la información

La arquitectura de la información tiene como objetivo organizar, clasificar, ordenar y estructurar los contenidos de un producto software, con el fin de que sus usuario pueden satisfacer sus necesidades con el menor esfuerzo posible. Una forma de llevar a cabo esta actividad es desarrollar un esquema organizativo. Yo he decidido realizar un esquema organizativo denominado “ambiguo” que consiste en realizar una organización por tareas. A continuación muestro el esquema que he seguido para organizar la información en mi aplicación.



Como vemos en el diagrama la información está organizada en 3 niveles. Las pantallas Inicio, Entradas, graficas, objetivos y opciones serán fragmentos que están contenidos en una TabHost Activity y según el tamaño del dispositivo que se utilice se mostrará como primera pantalla uno o los dos primeros fragmentos. Los siguientes dos niveles contienen diálogos y sub diálogos que el usuario puede ir abriendo según las tareas que quiera realizar, la idea es poner en los niveles superiores las tareas más frecuentes. Contando los diálogos la aplicación esta constituida por 19 pantallas.

5.2 Diseño de interacción

En el diseño de interacción se decide la forma en la que el usuario navegara por las distintas pantallas de la aplicación. En mi caso he elegido que las 5 pantallas principales sean fragmentos contenidos en una ViewPager que a su vez está contenida en un TabHost, esto permite que el usuario pueda navegar entre ellos deslizando el dedo sobre la pantalla o pulsando el tab correspondiente a la pantalla que desee visualizar. Desde cada una de las pantallas principales se puede acceder a diversos diálogos pulsando el botón correspondiente o manteniendo una pulsación larga sobre un determinado elemento con el cual se quieres interactuar.

5.3 Diseño gráfico y simplicidad

Este ha sido uno de los puntos más difíciles de tratar ya que normalmente no se suelen dar en la misma personas habilidades de programación y de diseño gráfico, en mi caso yo no me considero un buen diseñador gráfico, simplemente he intentado crear una interfaz de usuario coherente y lo más sencilla posible. También he intentado conseguir consistencia en el diseño, haciendo que las 5 pantallas principales tengan la misma apariencia y he elegido un tema diferente para el diseño de los diálogos. Siguiendo ciertos criterios y estándares puedes conseguir un resultado coherente y una IU intuitiva, pero crear algo original y atractivo requiere además de ciertas dotes para el diseño gráfico que yo no poseo, así considero que la interfaz de mi aplicación es intuitiva y está organizada de una forma lógica que permite al usuario realizar las tareas que desee lo más rápidamente posible, pero no he podido lograr algo atractivo e innovador. En las siguientes página muestro el resultado final de las 5 pantallas principales de mi aplicación. Si algún lector está interesado en explorar más a fondo la aplicación puede descargársela gratuitamente de Google Play desde el enlace que dejaré al final de este proyecto.

En la figura 4 muestro la pantalla inicial de la aplicación que sirve para introducir nuevas entradas, se trata de un fragmento contenido en un TabHost. Hago que este fragmento sea el primero que se muestre al usuario al iniciar la aplicación ya que considero que una de las tareas más corrientes será introducir nuevas entradas. La figura 5 se corresponde con el fragmento "Entradas" que utilizo para mostrar al usuario información sobre las entradas previamente introducidas. El usuario puede visualizar este fragmento desplazando el dedo sobre la pantalla. Lo pongo en segundo lugar ya que considero que a menudo los usuarios querrán ver el listado de gastos/ingresos a continuación de introducir una nueva entrada.

Figura 4. Pantalla Inicio de la aplicación para introducir un nuevo gasto/ingreso

Figura 5. Pantalla para visualizar las diferentes entradas

En las figuras 6 muestro un fragmento que contiene una gráfica anual de los gastos/ingresos y que se puede utilizar para comparar los resultados de los diferentes meses del año.

En la figura 7 se muestra el cuarto fragmento en el que podemos visualizar objetivos de gasto que previamente hemos introducido, así como introducir nuevos objetivos.

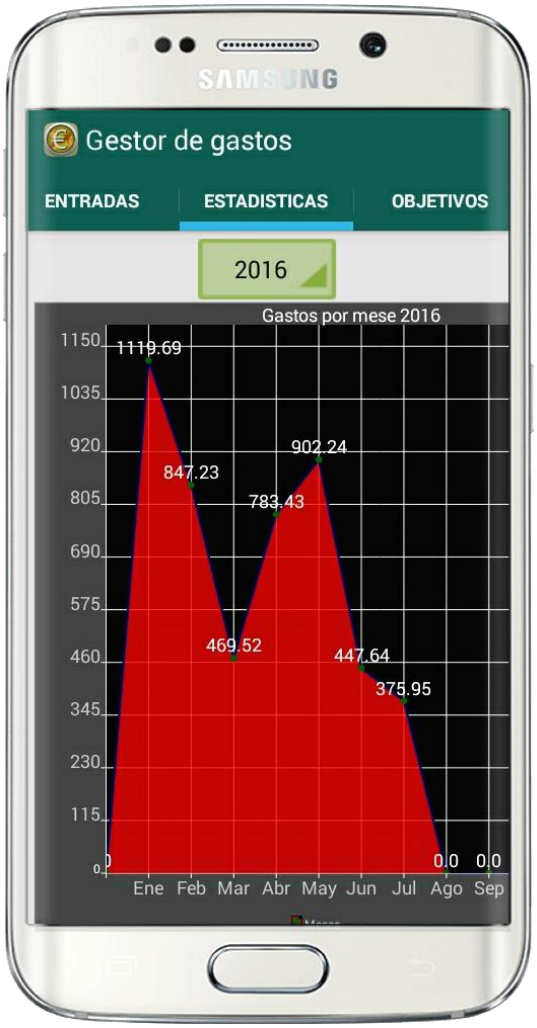


Figura 6. Grafica anual de gastos/ingresos

Figura 7. Pantalla para introducir objetivos de gasto/ahorro

Por último en la figura 8 muestro el último fragmento a través del cual se accede a 7 diálogos que permiten realizar las tareas descritas por los títulos de los mismos.

Figura 8. Pantalla con diferentes opciones

6. Implementación de la aplicación

En este punto voy a describir como fue el proceso de implementación de iBudget, centrándome en describir las partes de la implementación que considero más interesantes. Al final ha resultado una aplicación con 45 clases java que suman 9210 líneas de código para manejar la lógica de la aplicación, y 75 archivos xml que suman 5168 líneas de código que utilizo para crear la interfaz de usuario, traducción a otros idiomas y describir colores y gradientes. Debido a esto voy a omitir mucho código en el proyecto mostrando solo lo que considero más destacable.

A menudo se suele utilizar el tamaño de un proyecto medido en líneas de código para hacerse una idea de su complejidad. Para tal propósito existe un programa llamado cloc que está disponible de forma gratuita en http://cloc.sourceforge.net . Se trata de un programa bastante sencillo que se utiliza mediante ordenes de terminal, ya que no posee interfaz de usuario. La ventaja de cloc es que es capaz de diferenciar las líneas en blanco, los comentarios y el código escrito en múltiples lenguajes de programación. A continuación muestro el resultado de ejecutar cloc en el terminal de mi ordenador pasándole como argumento el directorio raíz de mi aplicación.

```
MacBook-Pro-vicente:Desktop vicente$ cloc ControlSpendsV22
     133 text files.
     128 unique files.
      13 files ignored.

github.com/AlDanial/cloc v 1.70  T=1.41 s (85.3 files/s, 15530.9 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                            45           1747           4615           9210
XML                             75            838            283           5168
-------------------------------------------------------------------------------
SUM:                           120           2585           4898          14378
-------------------------------------------------------------------------------
```

El desarrollo de la aplicación lo he realizado siguiendo el patrón de arquitectura de software modelo-vista-controlador, que consiste en lo siguiente:

- Vista: La forman los archivos XML que dibujan la interfaz de usuario. En el caso de mi aplicación las vistas son pasivas, es decir, no se comunican directamente con el modelo de datos sino que lo hacen a través del controlador.

- Controlador: Manejan los eventos que el usuario realiza a través de las vistas, consultan y modifican el modelo de datos. En Android son las actividades, fragmentos, adaptadores para listas, etc.. las que desempeñan este papel.

- Modelo: Es el modelo de datos que se utiliza para construir y actualizar la interfaz de usuario.

A continuación paso a explicar como he implementado las características más destacables de mi aplicación utilizando tanto API's de Android como API's de terceros.

6.1 API's de Android

Android se puede definir como un framework del lenguaje de programación Java, es decir, un esquema de desarrollo específico creado para la implementación de una aplicación sobre dicho lenguaje. La ventaja de utilizar un framework radica en que el desarrollador puede utilizar estructuras y servicios definidos en dicho framework en el lenguaje base en el que se desarrolla. Por supuesto las API's de Android ofrecen una amplísima variedad de servicios, estructuras y widgets. A continuación detallaré unos pocos que he utilizado en mi aplicación. Algunos los explicaré con más detalle y mostrando código ya que los considero más importantes y otros simplemente los mencionaré brevemente.

6.1.1 ViewPager combinada con ActionBar y Tabs

Se trata de una combinación muy utilizada en aplicaciones Android, un ejemplo de aplicación muy conocida que lo usa es WhatsApp. En mi aplicación es el elemento raíz a través del cual se empieza a navegar por las diferentes pantallas. La bondad de utilizar esta combinación es que ofrece al usuario dos formas diferentes de navegar entre los fragmentos, o bien pulsando el tab correspondiente o bien gracias a la ViewPager navegar deslizando el dedo sobre la pantalla. En las primeras versiones de mi aplicación utilizaba un simple tab para navegar entre las 5 pantallas y al comenzar el TFG decidí introducir el ViewPager, esto me llevo bastante trabajo ya que el controlador de cada una de las pantallas era una actividad y para utilizar el ViewPager tuve que convertirlas en fragmentos, lo cual no es trivial ya que funcionan de una forma un poco diferente. En la siguiente imagen trato de ilustrar como funciona esta estructura.

Figura 9. Estructura ActionBar + Tabs + ViewPager

1. Action bar o barra de acciones. En este caso contiene el título, el icono de mi aplicación y un menú overflow.
2. TabHost. Se trata de un contenedor de tabs, en cada tab se visualiza el título de uno de los fragmentos que contiene la ViewPager, así podemos conectar por código cada uno de los tabs con su correspondiente fragmento.
3. ViewPager. Contiene los 5 fragmentos y nos permite navegar entre ellos deslizando el dedo sobre la pantalla.
4. Fragmentos. Se adjuntan por código tanto a la ViewPager como al TabHost.

A continuación vamos a ver como se implementa esta estructura que constituye el núcleo y la mayor parte de la aplicación, voy a separar esta implementación en dos puntos, los archivos xml que forman la interfaz de usuario (Vista), el código java que maneja estas vistas (Controlador) y como se comunican los fragmentos contenidos en una ViewPager. El modelo de datos lo mostraré posteriormente en otro apartado.

6.1.1.1 Archivos xml

En Android la interfaz de usuario se construye al igual que en muchas otras plataformas utilizando el lenguaje xml. Básicamente tenemos contenedores y Widgets, los contenedores pueden contener Widgets u otros contenedores, esto basa en el llamado modelo de cajas. Los Widgets son objetos como botones, listas, calendarios, campos de texto, etc.. el SDK de Android nos proporciona una amplia gama de Widgets, y los contenedores son layouts que se utilizan para posicionar los Widgets en la pantalla de una forma conveniente.

Para crear la estructura ActionBar + Tabs + ViewPager utilizamos los siguientes archivos xml.

Archivo main_activity.xml

Aquí simplemente definimos la ViewPager que será el contenedor principal para los demás elementos que vamos a utilizar:

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

La ViewPager apareció en Android 3.0 Honeycomb , si se quiere utilizar este elemento en versiones anteriores de Android hay que utilizar las librerías de compatibilidad, en este caso la versión v4 de estas librerías, por ese motive la instancio añadiendo el prefijo support.v4.

Una vez tenemos esto solo nos falta crear la interfaz de usuario de los diferentes fragmentos que se adjuntarán a la ViewPager. Puesto que he necesitado mucho código para crear los 5 fragmentos, solo voy a mostrar el código de los dos primeros.

Archivo inico.xml

Se corresponde con el primero de los fragmentos, que será el que aparezca cuando se inicia la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:ads="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#e5e5e5"
    android:focusable="true"
    android:focusableInTouchMode="true"
    android:gravity="center_horizontal"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginBottom="5dp"
        android:layout_marginTop="5dp"
        android:layout_weight="1.2"
        android:orientation="horizontal" >

        <TextView
            style="@style/LabelsAdd"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="8dp"
            android:layout_weight="1"
            android:gravity="bottom"
            android:text="@string/Tipo"
            android:textAlignment="gravity" />

        <TextView
            style="@style/LabelsAdd"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="1"
            android:gravity="bottom"
            android:text="@string/Favoritos"
            android:textAlignment="gravity" />

        <Button
            android:id="@+id/botonOpcionesFavoritos"
            style="?android:attr/buttonStyleSmall"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_gravity="bottom"
            android:layout_marginRight="5dp"
            android:layout_weight="0.5"
            android:background="@android:drawable/ic_input_add"/>

    </LinearLayout>
```

```
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginBottom="15dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="4dp"
        android:layout_weight="1.3"
        android:orientation="horizontal" >

        <Spinner
            android:id="@+id/spinnerTipo"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="2dp"
            android:layout_marginRight="8dp"
            android:layout_weight="1.2"
            android:background="@drawable/spinner_background_ab_example"
            android:paddingLeft="10dp"
            android:textAlignment="center" />

        <Spinner
            android:id="@+id/spinnerFavoritos"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="8dp"
            android:layout_marginRight="2dp"
            android:layout_weight="2"
            android:background="@drawable/spinner_background_ab_example"
            android:paddingLeft="@dimen/activity_horizontal_margin" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >

        <TextView
            style="@style/LabelsAdd"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_marginLeft="8dp"
            android:layout_weight="3"
            android:gravity="bottom"
            android:text="@string/Concepto" />

        <TextView
            style="@style/LabelsAdd"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_marginLeft="5dp"
            android:gravity="bottom"
            android:layout_weight="2"
```

```
        android:text="@string/Cantidad" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2" >

        <EditText
            android:id="@+id/etConcepto"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="15dp"
            android:layout_weight="3"
            android:ems="10"
            android:background="@android:drawable/edit_text"
            android:inputType="text" />

         <EditText
            android:id="@+id/etPrecio"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginRight="5dp"
            android:layout_weight="2"
            android:ems="10"
            android:background="@android:drawable/edit_text"
            android:inputType="numberDecimal" />


    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_marginBottom="5dp"
        android:layout_weight="1.2" >

        <TextView
            style="@style/LabelsAdd"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_marginLeft="10dp"
            android:layout_weight="0.33"
            android:gravity="bottom"
            android:text="@string/Catergoria"
            android:textAlignment="gravity" />

        <Button
            android:id="@+id/opcionesCategorias"
            style="?android:attr/buttonStyleSmall"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            android:layout_weight="0.17"
            android:background="@android:drawable/ic_input_add"
            android:gravity="bottom" />
```

```
    <TextView
          style="@style/LabelsAdd"
          android:layout_width="0dp"
          android:layout_height="fill_parent"
          android:layout_marginLeft="20dp"
          android:layout_weight="0.33"
          android:gravity="bottom"
          android:text="@string/Fecha"
          android:textAlignment="gravity" />
  </LinearLayout>

  <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_marginLeft="4dp"
      android:layout_marginRight="4dp"
      android:layout_weight="1.3"
      android:orientation="horizontal" >

      <Spinner
          android:id="@+id/spinnerCategoria"
          android:layout_width="0dp"
          android:layout_height="wrap_content"
          android:layout_marginLeft="2dp"
          android:layout_weight="3"
          android:layout_marginRight="15dp"
          android:background="@drawable/spinner_background_ab_example"
          android:paddingLeft="@dimen/activity_horizontal_margin" />

      <Button
          android:id="@+id/butFecha"
          android:layout_width="0dp"
          android:layout_height="fill_parent"
          android:layout_marginRight="2dp"
          android:layout_weight="2"
          android:background="@drawable/list_focused_mitab"
          android:text="Button" />

  </LinearLayout>

  <FrameLayout
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="4" >

      <LinearLayout
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_gravity="bottom"
          android:orientation="vertical" >

          <LinearLayout
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:layout_marginBottom="1dp"
              android:orientation="horizontal" >
```


```
            <Button
                android:id="@+id/butSalir"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:minHeight="40dp"
                android:layout_marginLeft="7dp"
                android:layout_marginRight="100dp"
                android:layout_weight="1"
                android:background="@drawable/list_focused_mitab"
                android:text="@string/salir" />

             <Button
                android:id="@+id/butGuardar"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:minHeight="40dp"
                android:layout_marginRight="6dp"
                android:layout_weight="1"
                android:background="@drawable/list_focused_mitab"
                android:text="@string/Guardar" />

        </LinearLayout>
        <com.google.android.gms.ads.AdView
            android:id="@+id/adView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            ads:adSize="SMART_BANNER"
            ads:adUnitId="ca-app-pub-9998880883111466/1912110231" />
    </LinearLayout>
</FrameLayout>
</LinearLayout>
```

Archivo entradas.xml

Se trata del segundo fragmento en el que maestro las entradas previamente introducidas por el usuario.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="272dp"
        android:background="#e5e5e5"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0.65"
        android:orientation="vertical" >
```

```
    <LinearLayout
          android:layout_width="match_parent"
          android:id="@+id/layout1"
          android:layout_height="25dp"
          android:layout_marginTop="5dp"
          android:orientation="horizontal" >

          <TextView
              style="@style/LabelsAdd"
              android:layout_width="0dp"
              android:layout_height="fill_parent"
              android:layout_marginLeft="10dp"
              android:layout_marginRight="10dp"
              android:layout_weight="1.5"
              android:text="@string/Mes" />

          <TextView
              style="@style/LabelsAdd"
              android:layout_width="0dp"
              android:layout_height="fill_parent"
               android:layout_weight="1"
              android:layout_marginLeft="10dp"
              android:text="@string/Anyo" />

      </LinearLayout>

      <LinearLayout
          android:layout_width="match_parent"
          android:id="@+id/layout2"
          android:layout_height="35dp"
          android:layout_marginBottom="10dp"
          android:orientation="horizontal" >

          <Spinner
              android:id="@+id/spinnerMes"
              android:layout_width="0dp"
              android:layout_height="wrap_content"
              android:layout_marginLeft="2dp"
              android:layout_marginRight="10dp"
              android:layout_weight="1.5"
              android:background="@drawable/spinner_background_ab_example"
              android:paddingLeft="@dimen/activity_horizontal_margin"
              android:textAlignment="center" />

          <Spinner
              android:id="@+id/spinnerAnyo"
              android:layout_height="wrap_content"
              android:layout_width="0dp"
              android:layout_weight="1"
              android:layout_marginLeft="5dp"
              android:layout_marginRight="2dp"
              android:background="@drawable/spinner_background_ab_example"
              android:paddingLeft="@dimen/activity_horizontal_margin" />

      </LinearLayout>
```

```
<LinearLayout
            android:layout_width="match_parent"
            android:id="@+id/layout3"
            android:layout_height="40dp"
            android:layout_marginBottom="5dp"
            android:orientation="horizontal" >

            <TextView
                style="@style/LabelsAdd"
                android:layout_width="0dp"
                android:layout_height="fill_parent"
                android:layout_marginLeft="10dp"
                android:layout_weight="1"
                android:text="@string/MostrarPor" />

            <Spinner
                android:id="@+id/spinnerMostrarPor"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="2"
                android:layout_marginLeft="2dp"
                android:background="@drawable/spinner_background_ab_example"
                android:paddingLeft="@dimen/activity_horizontal_margin" />

        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="5dp"
            android:background="@android:color/black"
            android:orientation="horizontal" >
            </LinearLayout>

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@android:color/background_dark" >

            <ListView
                android:id="@android:id/list"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_gravity="center_horizontal" >
            </ListView>
        </FrameLayout>
    </LinearLayout>

    <TextView
        android:id="@+id/total"
        android:layout_width="match_parent"
        android:layout_height="25dp"
        android:background="@android:color/black"
        android:gravity="right"
        android:textAlignment="gravity"
        android:textColor="@color/green"
        android:textSize="18sp"
        android:textStyle="bold"
        android:typeface="serif" />
</LinearLayout>
```

El código de los 3 fragmentos restantes es similar excepto por los gráficos que explicaré en otro apartado. Una vez tenemos implementada la interfaz de usuario debemos implementar los controladores que permiten al usuario interactuar con la interfaz.

6.1.1.2 Controladores para manejar la estructura ActionBar + Tabs + ViewPager

Aquí necesitamos implementar 6 controladores, uno por cada fragmento y otro que se le llama actividad principal y es el punto de entrada a la aplicación. Al igual que he hecho con la interfaz de usuario solo mostraré capturas de código de la actividad principal y de los controladores de los dos primeros fragmentos. En este caso solo mostraré el código que considero más importante ya que los 3 controladores completos ocupan varios miles de líneas de código.

Archivo MainActivity.java

El código que ponemos en esta clase java será lo primero que se ejecute cuando el usuario pulse el icono de la aplicación en su teléfono, para conseguir esto debemos indicar en el archivo manifiesto que esta es la actividad principal. Lo hacemos de la siguiente forma.

```
<activity
    android:name=".MainActivity"
    android:screenOrientation="portrait"
    android:windowSoftInputMode="adjustPan"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Como puede verse en el intent-filter declaro que es la acción principal y como categoría LAUNCHER que indica que será la actividad encargada de lanzar la aplicación. Al implementar esta clase Java extiendo de ActionBarActivity, se trata de una clase java disponible en el SDK de Android que nos permite utilizar la barra de acciones en la aplicación. A continuación en el método onCreate (Ver ciclo de vida de las actividades en la figura 2 de la página 19) adjunto la interfaz de usuario previamente definida.

```
public class MainActivity extends ActionBarActivity implements InterfazDeComunicacion,
ActionBar.TabListener {

@Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main_activity);
```

Más adelante explicaré los implements, lo siguiente es instanciar la barra de acciones, el ViewPager y los tabs.

```
aBar = getSupportActionBar();
        aBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

aBar.addTab(aBar.newTab().setText(R.string.NuevaEntrada).setTabListener(this));
        aBar.addTab(aBar.newTab().setText(R.string.Entradas).setTabListener(this));

aBar.addTab(aBar.newTab().setText(R.string.Estadisticas).setTabListener(this));
        aBar.addTab(aBar.newTab().setText(R.string.Objetivo).setTabListener(this));
        aBar.addTab(aBar.newTab().setText(R.string.opciones).setTabListener(this));

        // Establece el ViewPager con las secciones del Adapter
        mViewPager = (ViewPager) findViewById(R.id.pager);
        // Crea el Adapter que devuelve un Fragment por cada sección
        adapterViewPager = new MyPageAdapter(getSupportFragmentManager());
        mViewPager.setAdapter(adapterViewPager);
```

Y a continuación instanciamos el adaptador como una clase interna, que se encargará de devolver a la ViewPager el fragmento adecuado cada vez que el usuario deslice el dedo por la pantalla.

```
public class MyPageAdapter extends SmartFragmentStatePagerAdapter {

        public MyPageAdapter(FragmentManager fm) {
                super(fm);
        }

        @Override
        public Fragment getItem(int position) {
                Fragment fragment = new Fragment();
                Bundle args = new Bundle();

                switch (position) {
                case 0:
                        fragment = new Add_f();
                        break;
                case 1:
                        fragment = new Entradas_f();
                        break;
                case 2:
                        fragment = new Graficos_f();
                        break;
                case 3:
                        fragment = new Objetivos_f();
                        break;
                case 4:
                        fragment = new Opciones_f();
                        break;
                default:
                        break;
                }

                return fragment;
        }
```

Y por último adjuntamos un listener a la ViewPager de modo que se compenetre con los tabs, así cuando el usuario cambie de fragmento deslizando el dedo también observaremos que se cambia al tab correspondiente.

```
mViewPager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {

                @Override
                public void onPageSelected(int position) {
                        aBar.setSelectedNavigationItem(position);

                }

                @Override
                public void onPageScrolled(int arg0, float arg1, int arg2) {
                }

                @Override
                public void onPageScrollStateChanged(int arg0) {
                }
        });
```

A continuación veremos una parte muy reducida del código de los fragmentos, un fragmento es un trozo (o fragmento) de una actividad que tiene su propio layout y su propio ciclo de vida, pero no puede existir fuera de la actividad y el ciclo de vida de la actividad afecta al ciclo de vida de los fragmentos que contiene. En este caso los cinco fragmentos están contenidos en la MainAcitivity. Antes de ver el código mostraré en una imagen como es el ciclo de vida de un fragmento y como se relaciona con el ciclo de vida de su actividad contenedora. Así mismo explicaré muy brevemente como debemos sobre escribir lo métodos principales que definen este ciclo de vida.

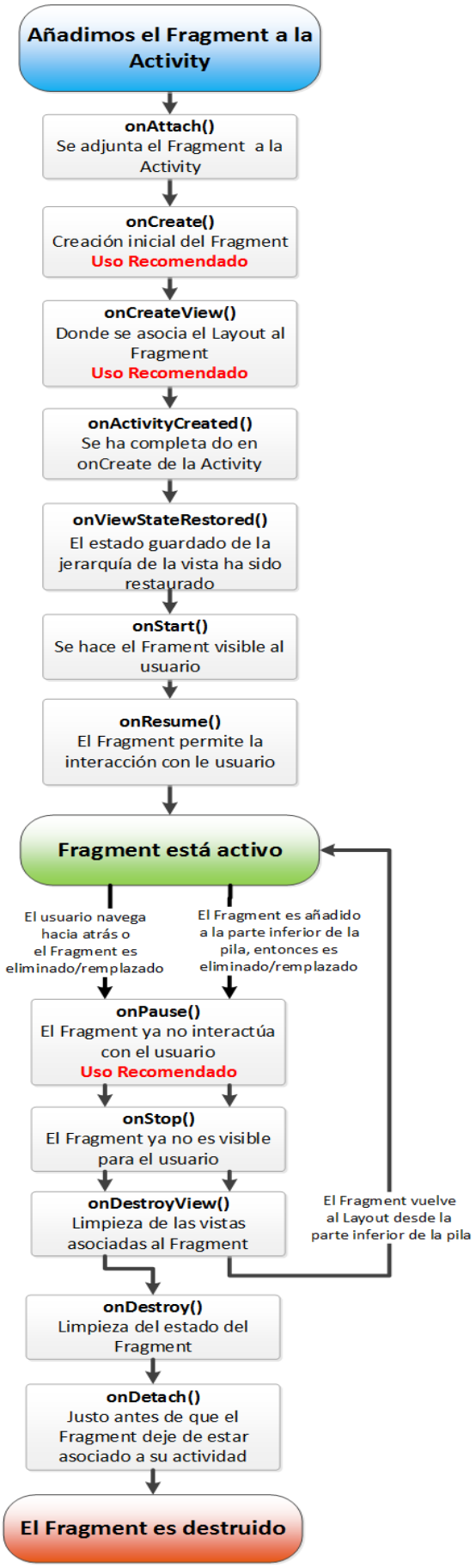


Figura 10. Ciclo de vida de un fragmento

Describo más detalladamente los 4 métodos que sobre escribo en mi aplicación:

onAttach()

Es llamado cuando la instancia del fragmento se asocial con la actividad que lo contiene.

OnCreate()

Este método es llamado por el sistema cuando se crea el fragmento. Aquí debemos inicializar los componentes esenciales que queremos retener durante toda la vida del fragmento .

onCreateView()

Es llamado por el sistema cuando llega el momento de dibujar la interfaz de usuario del fragmento. Aquí debemos adjuntar el archivo xml que define dicha interfaz. Debe devolver la vista previamente adjuntada o null si se trata de un fragmento sin interfaz de usuario.

onActivityCreated()

Es llamado cuando se ha terminado de ejecutar el método onCreate() de la actividad que contiene al fragmento. Lo podemos utilizar para comunicarnos con la actividad contenedora y para instanciar cada una de las vistas definidas en el archivo xml que previamente hemos adjuntado en onCreaterView() a las cuales les cargaremos un listener para manejar las interacciones del usuario.

Archivo Inicio.java

```
public class Inicio extends Fragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {

            super.onCreate(savedInstanceState);

            sDataFactory =
            DataFactory.getInstance(this.getActivity().getApplicationContext());

            setHasOptionsMenu(true);

    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                    Bundle savedInstanceState) {

            View rootView = inflater.inflate(R.layout.inicio, container, false);
            return rootView;
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {

            super.onActivityCreated(savedInstanceState);

            spinTipo = (Spinner) getView().findViewById(R.id.spinnerTipo);
            spinFavoritos = (Spinner) getView().findViewById(R.id.spinnerFavoritos);
            spinCategorias = (Spinner) getView().findViewById(R.id.spinnerCategoria);

            etConcepto = (EditText) getView().findViewById(R.id.etConcepto);
            etPrecio = (EditText) getView().findViewById(R.id.etPrecio);
            fecha = (Button) getView().findViewById(R.id.butFecha);
            btnOpFavoritos = (Button)
            getView().findViewById(R.id.botonOpcionesFavoritos);
            btnOpCategorias = (Button) getView().findViewById(R.id.opcionesCategorias);
            btnGuardar = (Button) getView().findViewById(R.id.butGuardar);
            btnSalir = (Button) getView().findViewById(R.id.butSalir); adapSpinTipo =
            ArrayAdapter.createFromResource(this.getActivity(),R.array.valoresSpinnerTip
            o,  android.R.layout.simple_spinner_item);

            Vector <String> favoritos = DataFactory.listFav.obternerListaFavoritos();
            adapSpinFavoritos = new
            ArrayAdapter<String>(this.getActivity(),android.R.layout.simple_spinner_item,
            favoritos);

            Vector <String> categorias = DataFactory.listCat.obternerListaCategorias();

            adapSpinCategorias = new
            ArrayAdapter<String>(this.getActivity(),android.R.layout.simple_spinner_item,
            categorias);

            spinTipo.setAdapter(adapSpinTipo);
            spinTipo.setOnItemSelectedListener(new OnItemSelectedListener() {
                    @Override
                    public void onItemSelected(AdapterView<?> parentView,
                                    View selectedItemView, int position, long id) {
                                            seleccionSpinTip = position;
                    }

                    @Override
                    public void onNothingSelected(AdapterView<?> arg0) {

                    }
            });
```

Archivo Entradas.java

```
public class Entradas_f extends Fragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        sDataFactory = DataFactory.getInstance(this.getActivity());
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View rootView = inflater.inflate(R.layout.entradas, container, false);
        return rootView;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        total = (TextView) getView().findViewById(R.id.total);
        spinMes = (Spinner) getView().findViewById(R.id.spinnerMes);
        spinAño = (Spinner) getView().findViewById(R.id.spinnerAnyo);
        spinMostrarPor = (Spinner)getView().findViewById(R.id.spinnerMostrarPor);
        lv = (ListView) getView().findViewById(android.R.id.list);

        adapPorMes = new AdaptadorListaEntradas(this,sDataFactory.getRegistros());
        lv.setAdapter(adapPorMes);
        lv.setOnItemLongClickListener(new OnItemLongClickListener() {

        @Override
        public boolean onItemLongClick(AdapterView<?> padre, View view,
                                       int position, long id) {

                try {
        EntradaPuntual reg =(EntradaPuntual)
        padre.getItemAtPosition(position);
        identificador = reg.getId();
        muestraOpciones(padre);
                    }
                catch(Exception ex) {

                    }
                return false;
            }
        });
```

El resto de los fragmentos se implementan de una forma similar, el objeto sDataFactory contiene el modelo de datos de mi aplicación por lo tanto lo primero que hago es obtener una instancia en cada fragmento. El método que contiene la mayor parte del código es el onActivityCreated(), ya que es desde donde manejo todas las interacciones que realiza el usuario.

6.1.1.3 Comunicación entre los fragmentos

Los fragmentos no pueden comunicarse directamente entre sí, deben hacerlo a través de la actividad que los contiene. A menudo es necesario pasar información de un fragmento a otro o que un fragmento sea consciente de un evento que el usuario a realizado en otro.

En mi caso debido al funcionamiento de la ViewPager necesito refrescar ciertas vistas de algunos fragmentos cuando el usuario realice ciertos eventos en otros. Siempre que la ViewPager instancia un fragmento instancia también los dos adyacentes a este, esto es necesario ya que podríamos encontrarnos con vistas pesadas de dibujar y de no hacerlo así no siempre sería posible pasar de una vista a otra de una forma suave cuando el usuario desliza el dedo sobre la pantalla. Pero esto conlleva que al estar ya creados los fragmentos adyacentes el sistema no volverá a llamar a los métodos que definen sus ciclos de vida y el código que tengamos en ellos no volverá a ejecutarse. Los problemas concretos en el caso de mi aplicación son los siguientes:

- **El usuario introduce una nueva entrada**: El fragmento adyacente es el que contiene la lista de entradas. Este fragmento ya estará instanciado y por lo tanto el sistema ya habrá llamado a todos los métodos de su ciclo de vida, esto provocará que si a continuación el usuario desea ver el listado de entradas la que acaba de introducir no será visible todavía, ya que la lista ya estaba instanciada antes de que el usuario realizara esta acción.
- **El usuario elimina una entrada**: Esto se hace en el fragmento entradas y el fragmento adyacente es el que muestra la gráfica con los gastos/ingresos anuales. De nuevo tenemos el mismo problema, si el usuario elimina o modifica el importe de un gasto/ingreso dichas actualizaciones no serán visibles en la gráfica, ya que esta ya está dibujada aunque todavía no sea visible.

Para notificar a un fragmento un evento ocurrido en otro debemos crear una interfaz con los métodos que deseemos utilizar, definir esa interfaz en el fragmento con el que nos queremos comunicar e implementarla en la actividad que lo contiene. El fragmento captura la implementación de la interfaz en su método onAttach() y entonces llama a los métodos definidos en dicha interfaz para comunicarse con la actividad. En nuestro caso las entradas se introducen en el fragmento inicio y se pueden eliminar o modificar en el fragmento entradas. Así la interfaz deberá contener un método que se ejecute cada vez que se guarda una entrada y otro que se ejecute cada vez que se elimina una entrada.

Muestro el código de la interfaz que he creado:

```
public interface InterfazDeComunicacion {

        public void onRegisterSaved();
        public void onRegisterDeleted();
}
```

Y definimos dicha interfaz en los respectivos fragmentos para que estos se comuniquen con la actividad que los contiene:

```
public class Inicio extends Fragment {

 InterfazDeComunicacion  comunicador;

 @Override
 public void onAttach (Activity activity) {
    super.onAttach(activity);
      try {
        comunicador = (InterfazDeComunicacion) activity;
      } catch (ClassCastException e) {
        throw new ClassCastException(activity.toString()+"Debe implementar la interfaz");
         }
   }


public class Entradas_f extends Fragment {

  InterfazDeComunicacion  comunicador;

  @Override
 public void onAttach(Activity activity) {
    super.onAttach(activity);
        try {
        comunicador = (InterfazDeComunicacion) activity;
        } catch (ClassCastException e) {
           throw new ClassCastException(activity.toString()+"Debe implementar la interfaz");
        }
    }
```

Una vez hecho esto ya podemos llamar a los métodos definidos en la interfaz desde los correspondientes fragmentos. En inicio llamaremos al método onRegisterSaved() cada vez que introduzcamos un nuevo registro y en entradas llamaremos a onRegisterDeleted() cada vez que eliminemos un registro.

```
public class DialogoEliminarItem extends DialogFragment {
      @Override
      public Dialog onCreateDialog(Bundle savedInstanceState) {
         AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
         builder.setIcon(android.R.drawable.ic_menu_delete);
         builder.setMessage(R.string.Eliminar);
         builder.setPositiveButton(android.R.string.yes, new DialogInterface.OnClickListener() {
            Override
            public void onClick(DialogInterface dialog, int id) {
               sDataFactory.EliminarRegistroPorIdentificador(identificador);
               adapPorMes.notifyDataSetChanged();
               actualizaTotalPorEliminarRegistro();
               comunicador.onRegisterDeleted();
                     }
                  })

           }
```

```
public void crearRegistro() throws ParseException {
    if (etConcepto.getText().toString().length() <= 0
                    || etPrecio.getText().toString().length() <= 0) {
        DialogoCamposNoValidos camposNoValidos = new     DialogoCamposNoValidos();
        camposNoValidos.show(getFragmentManager(), "Error");
    } else {
        String concepto = etConcepto.getText().toString();
        String pre = etPrecio.getText().toString();
        String categoria = seleccionSpinCat;
        int tipo = seleccionSpinTip;
        double precio = Double.parseDouble(pre);
        double preRedondeado = Math.rint(precio * 100) / 100;
        String fe = fecha.getText().toString();
        Date d = MainActivity.formatoFecha.parse(fe);
        EntradaPuntual reg = new EntradaPuntual(d, concepto, categoria,
        tipo,preRedondeado, false, -1, -1, -1);
        Calendar cal = Calendar.getInstance();
        cal.setTime(d);
        reg.setAño(cal.get(Calendar.YEAR));
        reg.setMes(cal.get(Calendar.MONTH));
        sDataFactory.guardarRegistro(reg);
        etConcepto.setText("");
        etPrecio.setText("");
        mostrarDialogoRegGuardado();
        spinFavoritos.setSelection(0);
        spinCategorias.setSelection(0);
        comunicador.onRegisterSaved();

                }
        }
```

Ya solo nos queda implementar la interfaz de comunicación en la actividad que contiene los fragmentos. Cuando los métodos de la interfaz sean llamados, obtendremos una instancia del fragmento en cuestión a través de la ViewPager, lo que nos permitirá ejecutar métodos contenidos en dichos fragmentos.

```
public class MainActivity extends ActionBarActivity implements InterfazDeComunicacion,
ActionBar.TabListener {

 @Override
 public void onRegisterSaved() {
        Entradas_f  f = (Entradas_f) this.adapterViewPager.getRegisteredFragment(1);
        f.refrescaAdaptadores();
   }
 @Override
 public void onRegisterDeleted() {
        Graficos_f g = (Graficos_f) this.adapterViewPager.getRegisteredFragment(2);
        g.actualiza();
  }
```

6.1.2 ListViews

Las listViews nos permiten mostrar listas de elementos de una forma eficiente ya que Android solo renderiza los elementos que son visibles para el usuario. Yo las utilizo para mostrar los listados de entradas, listados de objetivos, listado de opciones y listado de operaciones. Aquí voy a mostrar como he implementado el listado de entradas.

Primero definimos el widget ListView en el archivo entradas.xml y después creamos la vista que tendrá cada elemento de la lista en un archivo diferente, en este caso el archivo elemento_lista.xml.

```
            <ListView
                android:id="@android:id/list"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_gravity="center_horizontal" >
            </ListView>
```

Y en elemento_lista.xml:

```
    <RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layaut"
    android:layout_width="fill_parent"
    android:layout_height="60dp"
    android:background="@drawable/fondoelemlista"
    android:gravity="right" >

    <TextView
        android:id="@+id/tvFecha"
        style="@style/LabelsAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="15dp"
        android:layout_marginTop="10dp"
        android:text="fecha" />
```

```
<TextView
        android:id="@+id/tvCategoria"
        style="@style/LabelsAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/tvFecha"
        android:layout_alignBottom="@+id/tvFecha"
        android:layout_centerHorizontal="true"
        android:text="categoria" />

    <TextView
        android:id="@+id/tvConcepto"
        style="@style/LabelsAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/tvFecha"
        android:layout_below="@+id/tvFecha"
        android:layout_marginTop="8dp"
        android:text="concepto" />

    <TextView
        android:id="@+id/tvPrecio"
        style="@style/LabelsAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/tvConcepto"
        android:layout_alignBottom="@+id/tvConcepto"
        android:layout_alignParentRight="true"
        android:layout_marginRight="20dp"
        android:layout_toRightOf="@+id/tvCategoria"
        android:gravity="right"
        android:text="precio" />

    <Button
        android:id="@+id/button_automatico"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:focusable="false"
        android:focusableInTouchMode="false"
        android:layout_above="@+id/tvPrecio"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:visibility="gone"
        android:text="" />

</RelativeLayout>
```

Una vez definida la interfaz de usuario tenemos que crear el correspondiente controlador. Para ello lo primero que tenemos que hacer es crear un adaptador que se encargue de dibujar cada elemento de la ListView, esto lo hacemos extendiendo de la clase ArrayAdapter que nos proporcionan las API's de Google y sobre escribiendo el método getView().

```
public class AdaptadorListaEntradas extends ArrayAdapter <EntradaPuntual>{
        Activity context;
        protected ArrayList<EntradaPuntual>listRegistros;

        public AdaptadorListaEntradas(Fragment context,ArrayList<EntradaPuntual>list) {
                super(context.getActivity(),R.layout.elementolista,list);
                this.context = context.getActivity();
                listRegistros = list;
         }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
                LayoutInflater inflater = context.getLayoutInflater();
                View item = inflater.inflate(R.layout.elementolista, null);

                EntradaPuntual reg = listRegistros.get(position);

                Button automatico = (Button) item.findViewById(R.id.button_automatico);
                TextView fecha = (TextView) item.findViewById(R.id.tvFecha);
                fecha.setText(MainActivity.formatoFecha.format(reg.getFecha()));
                TextView concepto = (TextView) item.findViewById(R.id.tvConcepto);
                concepto.setText(reg.getConcepto());
                TextView categoria = (TextView) item.findViewById(R.id.tvCategoria);
                categoria.setText(reg.getCategoria());
                DecimalFormat df = new DecimalFormat("0.00");
                String preRedondeado = df.format(reg.getPrecio());

                if(reg.getTipo() == MainActivity.GASTO) {
                        preRedondeado = "-"+preRedondeado;
                  }
                  TextView precio = (TextView) item.findViewById(R.id.tvPrecio);
                  precio.setText(preRedondeado);
                  if(reg.getTipo() == MainActivity.GASTO) {
                     precio.setTextColor(Color.RED);
                  }
                  else precio.setTextColor(Color.GREEN);

                  if(reg.isAutomatico()) {
                        automatico.setVisibility(View.VISIBLE);
                        int periodicidad = reg.getPeriodicidad();
                        switch (periodicidad) {
                                case 0 :

        automatico.setBackgroundResource(android.R.drawable.ic_menu_day);
                                        break;
                                case 1 :

        automatico.setBackgroundResource(android.R.drawable.ic_menu_week);

                                        break;
                                case 2 :

        automatico.setBackgroundResource(android.R.drawable.ic_menu_my_calendar);
                                        break;
                                case 3 :

        automatico.setBackgroundResource(android.R.drawable.ic_menu_month);

                                        break;

                                }
                  }

                  return item;
        }
```

En el constructor le pasamos el archivo xml que define la vista de cada elemento, el contexto de la actividad desde donde se llamará y un ArrayList con todas las entradas que queramos incluir en la lista. Al principio se dibujarán tantas entradas como quepan en pantalla y cada vez que el usuario se desplace por la lista se llamará al método getView() que irá cogiendo las entradas del ArrayList y las irá dibujando. Como se puede observar en este método podemos ir comprobando los atributos de cada entrada y según estos dibujar cada ítem de una forma o de otra, como por ejemplo la cantidad de una entrada en rojo o verde según sea un gasto o un ingreso.

Después en el fragmento entradas que es donde se mostrará la vista, instanciamos el adaptador y se lo cargamos a la ListView. También cargamos un listener para que aparezca un menú de opciones cada vez que el usuario pulsa un elemento de la lista.

```
public class Entradas_f extends Fragment {

 lv = (ListView) getView().findViewById(android.R.id.list);
 adapPorMes = new AdaptadorListaEntradas(this,sDataFactory.getRegistros());
lv.setAdapter(adapPorMes);
lv.setOnItemLongClickListener(new OnItemLongClickListener() {

@Override
public boolean onItemLongClick(AdapterView<?> padre, View view,
                                    int position, long id) {
  try {
     EntradaPuntual reg =(EntradaPuntual)  padre.getItemAtPosition(position);
     identificador = reg.getId();
     muestraOpciones(padre);
       }
catch(Exception ex) {

      }
return false;
   }
});
```

Y este es el resultado final:

6.1.3 AsyncTask

Se trata de una clase predefinida en Android que se utiliza para descargar de trabajo al hilo principal de la aplicación, creando hilos secundarios que se ejecutarán en paralelo a los que se les puede asignar tareas que no impliquen trabajar con elementos de la UI. Cuando se trabaja en Android solo el hilo principal puede manejar la interfaz de usuario, por lo que si se realizan tareas pesadas que no impliquen trabajar con la UI es aconsejable utilizar esta clase, ya que de lo contrario la interfaz de usuario queda bloqueada hasta que dichas tareas terminen lo que afecta de forma negativa a la experiencia de usuario y en casos extremos puede provoca que el sistema lance un ANR (Application Not Responding) preguntando al usuario si desea cerrar la aplicación.

Esta clase crea y elimina hilos de manera transparente al desarrollador y se encarga de controlar los problemas de concurrencia que pudiesen surgir al paralelizar el código utilizando diferentes hilos de ejecución.

La forma de utilizarla es crear una clase nueva que la extienda y sobre escribir los métodos que se necesiten para el desarrollo. En mi caso la utilizo para cargar el modelo de datos en memoria y realizar algunas comprobaciones cuando el usuario inicia la aplicación, entre tanto el hilo principal puede ir dibujando parte de la interfaz de usuario.

```
public class InicializaRegistros extends AsyncTask<Void, Void, Void>{

  @Override
  protected Void doInBackground(Void... params) {
     registros = bd.sacarEntradas(current_year, current_month);
     Calendar calendario = Calendar.getInstance();
      Vector<EntradaRecurrente>registrosAutomaticos =
      bd.sacarRegistrosAutomaticos(calendario.getTimeInMillis());
      Date fechaActual = calendario.getTime();
      Iterator<EntradaRecurrente> it = registrosAutomaticos.iterator();
      EntradaRecurrente reg_automatico;
      while(it.hasNext()){
        reg_automatico = it.next();
        boolean estado_operacion =
        consultarEstadoOperacion(reg_automatico.getId_operacion());
        if(estado_operacion == true){
            if(reg_automatico.getFecha().before(fechaActual) ||
            reg_automatico.getFecha().equals(fechaActual)){
               Date d = reg_automatico.getFecha();
                EntradaPuntual reg = new
                EntradaPuntual(d,reg_automatico.getConcepto(),reg_automatico.getCategoria(),
                reg_automatico.getTipo(),reg_automatico.getPrecio(),true,
                reg_automatico.getPeriodicidad(), reg_automatico.getId_cuenta(), -1);
                bd.borrarRegistroAutomatico(reg_automatico.getId());
                                    }
                            }


             }
```

6.1.4 Diálogos

Los diálogos son normalmente pequeñas ventanas que no cubren toda la pantalla y suelen utilizarse para que el usuario tome decisiones y reciba o introduzca información adicional. Aquí no voy a mostrar código ya que sería extenderme demasiado, solamente voy a describir los tipos de diálogos que utilizo y mostraré algunas capturas de pantalla.

6.1.4.1 AlertDialog

Es uno de los tipos más básicos de diálogo, puede mostrar un título, hasta un máximo de 3 botones y una lista de elementos seleccionables. También es posible personalizar su apariencia creando tu propio layout. Muestro dos ejemplos de este tipo de diálogos que utilizo en mi aplicación.

Dialogo 1. Permite al usuario activar una contrseña

Dialogo 2.Muestra las entradas programadas que el usuario haya establecido previamente

6.1.4.2 DatePickerDialog

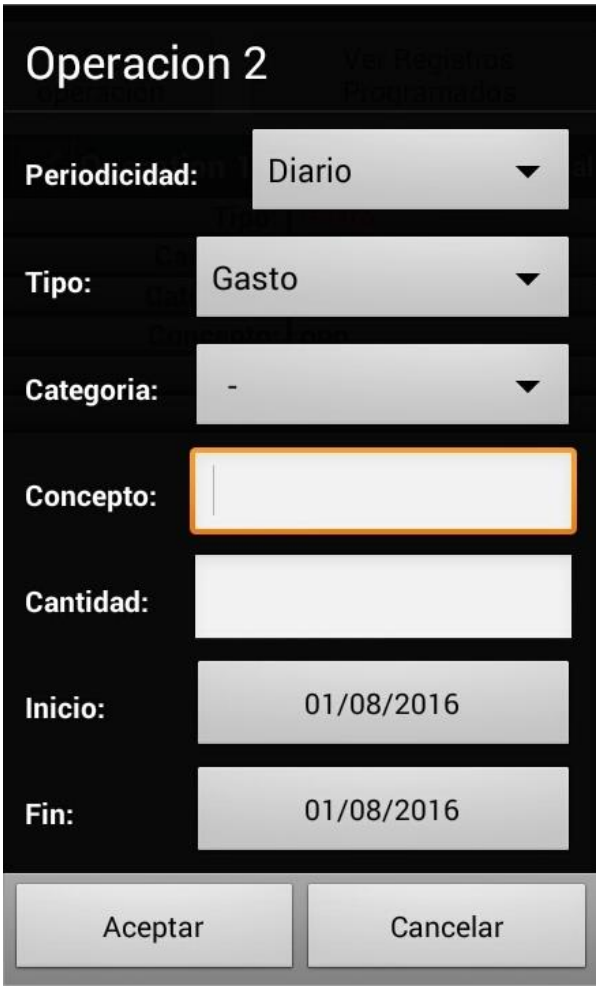
Son diálogos con una interfaz de usuario predefinida y que se utilizan para permitir al usuario seleccionar fechas. Utilizo varios de estos en mi aplicación.

Dialogo 3. Permite al usuario seleccionar una fecha del calendario

6.1.4.3 FragmentDialog

Estos diálogos se utilizan para propósitos generales, funcionan de forma similar a las actividades y fragmentos. Incluyen una vista personalizada que normalmente se define en un archivo xml. También utilizo varios en mi aplicación, muestro uno para crear y recuperar respaldos de la base de datos y otro para programar entradas recurrentes.

Dialogo 4. Permite crear y recuperar respaldos a la vez que muestra información sobre las entradas de la base de datos



Dialogo 5. Permite crear operaciones que se encargaran de crear las correspondientes entradas recurrentes

6.1.5 Spinners y botones

Se trata de controles básicos que existen en todas las plataformas. Los Spinners despliegan listas con diferentes opciones y permiten al usuario seleccionar una de ellas. Poco que decir sobre los botones, simplemente captan el evento que realiza el usuario al pulsarlos y envían dicho evento al controlador donde se programa la respuesta adecuada. En Android puedes incluir el evento en el mismo archivo xml donde defines el botón o mediante un listener en el controlador.

6.1.6 La base de datos SQLite

Se trata de una maravillosa herramienta que Android te proporciona para la capa de persistencia y resulta increíblemente sencillo trabajar con ella. Se trata un motor para bases de datos SQL que según su página oficial https://www.sqlite.org tiene las siguientes características:

- **No requiere soporte de un servidor:** SQLite no ejecuta un proceso para administrar la información si no que implementa un conjunto de librerías encargadas de la gestión.
- **No necesita configuración:** Libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc..
- **Usa un archivo para el esquema:** Crea un archivo completo de la base de datos lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones no pueden ser accedidos por contextos externos.
- **Es de código abierto:** Está disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

En el apartado 4.3 muestro el esquema de la base de datos, aquí solo añadiré unos cuantos detalles que considero más propios del apartado de implementación.

6.1.6.1 SQLiteOpenHelper

Para la creación de la base de datos y el control de la versión de ésta, se ha creado la clase BD que extiende a SQLiteOpenHelper. Esta clase se instancia como un objeto más de la aplicación y se emplea para conectar con la base de datos y en caso de que no exista crearla.

Los métodos más importantes son:

- El constructor, desde donde llamamos al constructor de su clase padre, bien para crear la base de datos si no existe o para conectarnos a ella si existe. Debemos pasar como parámetros el nombre la base de datos y la versión de la misma.
- El método onCreate(SQLiteDatabase db), donde creamos las tablas de la base de datos. Este método solo se ejecuta si la base de datos no existe
- El método onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion), se utiliza para actualizar la estructura de la base de datos. Recibe como parámetros la versión antigua, que corresponde con la versión de la base de datos instalada y la nueva versión, en caso de que hagamos una actualización de la aplicación y hayamos creado una nueva versión de la base de datos. Dentro de este método podemos poner condiciones para borrar, crear nuevas tablas o modificar las existentes dependiendo del número de versión que tenga instalado el usuario.

En esta clase también ponemos los métodos para realizar las consultas a la base de datos, así dichos métodos podrán aplicarse al objeto de esta clase que instanciemos y esto nos dará acceso a la misma en cualquier parte de la aplicación. A continuación muestro fragmentos de código de los 3 métodos mencionados arriba y de algunos métodos para realizar las consultas mencionadas.

Creación de la clase BD sobre escribiendo los métodos onCreate y onUpgrade:

```
public class BD extends SQLiteOpenHelper{

                super(context,"MiBD",null,4);

                }

   @Override
   public void onCreate(SQLiteDatabase db) {

       db.execSQL("CREATE TABLE if not exists registros (_id INTEGER PRIMARY KEY, fecha I
       NTEGER, concepto TEXT, categoria TEXT, tipo INTEGER, precio DOUBLE, automatico
       BOOLEAN, periodicidad INTEGER, id_cuenta INTEGER, id_adjunto INTEGER, year
       INTEGER, month INTEGER)");
                                                .
                                                .

       db.execSQL("CREATE TABLE if not exists objetivos (_id INTEGER PRIMARY KEY , finicio
       long, ffin long, tipo TEXT, nomdecatocon TEXT, cantlimite DOUBLE, id_cuenta
       INTEGER)");


   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

          if (oldVersion < 2) {

             db.execSQL("DROP TABLE IF EXISTS registros");
             db.execSQL("DROP TABLE IF EXISTS favoritos");
             db.execSQL("DROP TABLE IF EXISTS categorias");
             db.execSQL("DROP TABLE IF EXISTS objetivos");
                                                .
                                                .
                }

          if(oldVersion == 2){

            ArrayList<RegistroAntiguo> lr = sacarRegistrosPorMigracion(db);
            db.execSQL("DROP TABLE IF EXISTS registros");
            db.execSQL("CREATE TABLE registros (_id INTEGER PRIMARY KEY, fecha INTEGER,
            concepto TEXT, categoria TEXT, tipo INTEGER, precio DOUBLE, automatico
            BOOLEAN, periodicidad INTEGER, id_cuenta INTEGER, id_adjunto INTEGER, year
            INTEGER, month INTEGER)");
            guardarRegistroPorMigracion(db,lr);

                                                .
                                                .


                }
            }
```

Método para obtener las entradas de la base de datos:

```
public ArrayList<EntradaPuntual> sacarEntradas(int year, int month){

    ArrayList<EntradaPuntual> lr = new ArrayList<EntradaPuntual>();
    SQLiteDatabase db = getReadableDatabase();
    Cursor cur;
        if(month == -1){
            cur = db.rawQuery("SELECT * FROM registros WHERE year = "+year+" ORDER BY
            fecha", null);
          }
          else{
            cur = db.rawQuery("SELECT * FROM registros WHERE year = "+year+" and month
            ="+month+" ORDER BY fecha", null);
          }


    int FECHA, CONCEPTO, CATEGORIA, TIPO,  PRECIO, AUTOMATICO,  _ID, PERIODICIDAD,
    YEAR, MONTH,ID_CUENTA,ID_ADJUNTO;

    FECHA = cur.getColumnIndex("fecha"); CONCEPTO = cur.getColumnIndex("concepto");
    CATEGORIA =  cur.getColumnIndex("categoria");
    TIPO =  cur.getColumnIndex("tipo"); PRECIO = cur.getColumnIndex("precio"); AUTOMATICO
    = cur.getColumnIndex("automatico");
    _ID = cur.getColumnIndex("_id"); PERIODICIDAD = cur.getColumnIndex("periodicidad");
    ID_CUENTA = cur.getColumnIndex("id_cuenta");
    ID_ADJUNTO = cur.getColumnIndex("id_adjunto"); YEAR = cur.getColumnIndex("year");
    MONTH = cur.getColumnIndex("month");

    while(cur.moveToNext()) {
        EntradaPuntual reg = new EntradaPuntual(new
        Date(cur.getLong(FECHA)),cur.getString(CONCEPTO),cur.getString(CATEGORIA),
        cur.getInt(TIPO) , cur.getDouble(PRECIO),
        Boolean.parseBoolean(cur.getString(AUTOMATICO)), cur.getInt(PERIODICIDAD) ,
        cur.getLong(ID_CUENTA), cur.getLong(ID_ADJUNTO));

        reg.setId(cur.getLong(_ID));
        reg.setAño(cur.getInt(YEAR));
        reg.setMes(cur.getInt(MONTH));
        lr.add(reg);
          }

   cur.close();
   return lr;

     }
```

Método para eliminar una entrada:

```
public void borrarRegistro(long identificador) {
  SQLiteDatabase db = getWritableDatabase();
  db.execSQL("DELETE FROM registros WHERE _id = "+identificador);
    }
```

Método para actualizar una entrada:

```
public void editarRegistro(Date d, String con, String cat, double cantidad, int tipo, long id,
int y, int m) {
   SQLiteDatabase db = getWritableDatabase();
    db.execSQL("UPDATE registros SET  fecha = '"+d.getTime()+"'," +"concepto = '"+con+"'," +
    "categoria = '"+cat+"',tipo = '"+tipo+"', precio = '"+cantidad+"', year = '"+y+"', month =
    '"+m+"' WHERE _id = '"+id+"'");
        }
```

6.1.6.2 Optimización del esquema de la base de datos

Puesto que sobre la tabla EntradasPuntuales preveo hacer búsquedas utilizando los campos id_cuenta y fecha creo un índice sobre estos campos. Al hacerlo habilito al motor SQL a realizar búsquedas binarias que son mucho más eficientes. Esto tiene efectos negativos sobre las inserciones y actualizaciones, pero como estas operaciones preveo que se realicen con menos frecuencia, la creación de dichos índices queda justificada.

```
db.execSQL("CREATE INDEX indice_tabla_entradasPuntuales" +
        "EntradasPuntuales (id_cuenta, fecha)");
}
```

También se puede observar en el esquema que defino la clave primaria como INTEGER PRIMARY KEY en lugar de INTEGER PRIMARI KEY AUTOINCREMENT, esto es por motivos de eficiencia, si leemos la documentación oficial https://www.sqlite.org/autoinc.html :

> The behavior implemented by the AUTOINCREMENT keyword is subtly different from the default behavior. With AUTOINCREMENT, rows with automatically selected ROWIDs are guaranteed to have ROWIDs that have never been used before by the same table in the same database. And the automatically generated ROWIDs are guaranteed to be monotonically increasing. These are important properties in certain applications. But if your application does not need these properties, you should probably stay with the default behavior since the use of AUTOINCREMENT requires additional work to be done as each row is inserted and thus causes INSERTs to run a little slower.

Así solo deberíamos utilizar la clausula AUTOINCREMENT si queremos que las entradas tengan una misma id durante toda la vida de la aplicación. En mi caso este identificador lo uso solamente para realizar las operaciones SELECT, UPDATE y DELETE, con lo cual solo necesito que las entradas conserven un mismo identificador durante cada sesión de usuario y no durante toda la vida de la aplicación.

6.2 API's de terceros

En la aplicación he integrado una serie de API's de terceros que utilizo para diversos propósitos, en este apartado paso a enumerar y explicar para que utilizo cada una de estas API's.

6.2.1 API's de Facebook

Se trata de unas API's que Facebook ha creado para la plataforma Android, ofrecen una gran cantidad de posibilidades, en mi caso las utilizo para que el usuario pueda compartir en su Facebook que utiliza mi aplicación. Para poder hacer esto el usuario debe tener instalada la aplicación de Facebook en su teléfono.

Para este propósito las API's te proporcionan un diálogo con una interfaz predefinida que puedes personalizar. Paso a mostrar el diálogo y el código para instanciarlo.

Y este es el código para instanciarlo:

```
public void mostrarDialogoFacebook(){

 if (FacebookDialog.canPresentShareDialog(this.getActivity().getApplicationContext(),
                FacebookDialog.ShareDialogFeature.SHARE_DIALOG)) {
     // Publish the post using the Share Dialog
     Resources res = this.getActivity().getResources();
     String mensaje = res.getString(R.string.msn_comparte_facebook);
     FacebookDialog shareDialog = new FacebookDialog.ShareDialogBuilder(this.getActivity())
     .setLink("http://play.google.com/store/apps/details?id=com.logica.controlspends")
     .setName("ControlSpends")
      setCaption(mensaje)
     .setPicture("https://lh6.ggpht.com/3uIdIUzN6j4W84wuJyro6BtHCZHFBTzlHvgwRR4IkI9TX
     CKXHxbQIr2IXAsRw_kZ8g=w300")
           .build();
     MainActivity.uiHelper_facebook.trackPendingDialogCall(shareDialog.present());

      }
 else {

     Toast.makeText(MainActivity.ctxMainActivity,R.string.msn_no_app_facebook,Toast.LENGT
     H_SHORT).show();
      }
}
```

6.2.2 API's de Google Analytics

Para utilizar estas API's debes abrirte una cuenta en Google Analytics, esto te da acceso a un portal web donde puedes visualizar todos los datos que recoges sobre el uso de tu aplicación. Algunos datos interesantes que puedes recopilar son:

- El número de usuarios activos que tiene tu aplicación y con que frecuencia la utilizan.
- Cuantas veces acceden los usuarios a las diferentes funcionalidades de la aplicación, por ejemplo puedes saber las veces que pulsan un determinado botón o cuando abren alguno de los diálogos. Esto te ayuda a saber que es lo más les gusta de la aplicación.
- Versiones de Android y características de los dispositivos de los usuario, así como los países en los que se utiliza la aplicación.
- Otros datos específicos de la aplicación que pueden ser un buen indicador de la fidelidad de los usuario, como el número de entradas que cada usuario tiene introducidas.

Todo esto lo conseguimos instanciando un objeto Tracker y llamándolo en los sitios adecuados, por ejemplo dentro del listener de un botón. También le podemos pasar parámetros a este objeto para que recoja información de la base de datos como el número de gastos/ingresos que tienen almacenados los usuarios. A continuación muestro el código necesario para instanciar este objeto.

```
public void inicializaTracker(){
 GoogleAnalytics analytics;
 final String PROPERTY_ID = "UA-52563550-2";
 analytics = GoogleAnalytics.getInstance(mAppContext);
 long num_reg = sDataFactory.num_registros_en_bd();
 long num_automaticos = sDataFactory.num_registros_automaticos_en_bd();
 String reg = String.valueOf(num_reg);
 String reg_aut = String.valueOf(num_automaticos);
 Tracker tracker  = analytics.newTracker(PROPERTY_ID);
 tracker.setSessionTimeout(300);
 tracker.setScreenName("Num_reg = "+reg+" Num_automaticos =   "+reg_aut);
 tracker.send(new HitBuilders.AppViewBuilder().build());
        }
```

6.2.3 Google play services

Estas API's permiten entre otros servicios poner publicidad y ofrecer compras integradas en las aplicaciones. Yo las utilizo solamente para incluir publicidad, para ello debes abrirte una cuenta en AdMob que es la agencia de publicidad de Google. También es posible utilizar publicidad de otras agencias, incluso de varias de ellas en la misma aplicación, cada una proporciona sus propias API's. Puesto que AdMob pertenece a Google la integración de su publicidad en las aplicaciones Android es muy sencilla.

Debes darte de alta en el sitio web de AdMob, https://www.google.com/admob/ y desde ahí puedes crear los anuncios. Al crear un anuncio se te asigna un token que se utiliza para saber que ese anuncio concreto se está mostrando en tu aplicación.

Hay varios tipo de anuncios, yo utilizo banners e interstitial. Los banners son pequeños anuncios que suelen mostrarse en la parte de arriba o al pie de la pantalla, mientras que los interstitial ocupan toda la pantalla.

En el caso de los banners los definimos en el archivo xml como un elemento más y después los cargamos desde el controlador de la vista correspondiente. En caso de que no haya publicidad para mostrar el banner no ocupará espacio en pantalla. En el caso de los interstitials simplemente se llaman por código desde el correspondiente controlador y actúan como una actividad independiente con su propio layout ocupando toda la pantalla del dispositivo.

A continuación muestro un ejemplo de código para instanciar un banner:

```
<com.google.android.gms.ads.AdView
  android:id="@+id/adView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  ads:adSize="SMART_BANNER"
  ads:adUnitId="ca-app-pub-9998880883111466/1912110231" />
```

```
AdView adView = (AdView) getView().findViewById(R.id.adView);
AdRequest adRequest = new AdRequest.Builder().build();
adView.loadAd(adRequest);
```

Y para que aparezca un interstitial a pantalla completa, por ejemplo cuando pasamos del fragmento gráficos al fragmento objetivos:

```
InterstitialAd interstitial = new InterstitialAd(this);
interstitial.setAdUnitId("ca-app-pub-9998880883111466/7897907038");
AdRequest adRequest = new AdRequest.Builder().build();
interstitial.loadAd(adRequest);

@Override
public Fragment getItem(int position) {
    Fragment fragment = new Fragment();
    Bundle args = new Bundle();
    switch (position) {
          case 0:
            fragment = new Inicio();
            break;
          case 1:
            fragment = new Entradas_f();
            break;
          case 2:
            fragment = new Graficos_f();
            break;
          case 3:
             fragment = new Objetivos_f();
             displayInterstitial();
            break;
          case 4:
            fragment = new Opciones_f();
            break;
          default:
            break;
                  }
return fragment;
          }
```

6.2.4 API's Apache POI

Se utilizan para generar archivos Excel (xls). Permite crear elementos básicos de Excel como hojas, filas y celdas, así como darles formato e introducir valores y formulas en ellas.

```
Workbook libro = new HSSFWorkbook();
Sheet hoja = libro.createSheet("Hoja 1");
Row fila = hoja.createRow(0);
CellStyle style = libro.createCellStyle();
Font font = libro.createFont();

style.setFillForegroundColor(IndexedColors.GOLD.getIndex());
style.setFillPattern(CellStyle.SOLID_FOREGROUND);

font.setColor(IndexedColors.WHITE.getIndex());
font.setBoldweight((short)5);
font.setFontHeightInPoints((short)12);
style.setFont(font);

Cell c0 = fila.createCell(0);
c0.setCellValue(fecha);
c0.setCellStyle(style);
Cell c1 = fila.createCell(1);
c1.setCellValue(categoria);
c1.setCellStyle(style);
Cell c2 = fila.createCell(2);
c2.setCellValue(concepto);
c2.setCellStyle(style);
Cell c3 = fila.createCell(3);
c3.setCellValue(cantidad);
c3.setCellStyle(style);

double gastos =0;
double ingresos =0;
double total;
ArrayList<EntradaPuntual> registros = sDataFactory.sacarTodasLasEntradasParaExcel();
Collections.sort(registros);
Iterator<EntradaPuntual> it = registros.iterator();
EntradaPuntual reg;
Font fuenteGasto = libro.createFont();
Font fuenteIngreso = libro.createFont();
DataFormat format = libro.createDataFormat();
fuenteGasto.setColor(IndexedColors.RED.getIndex());
fuenteIngreso.setColor(IndexedColors.GREEN.getIndex());
CellStyle estiloGasto = libro.createCellStyle();
CellStyle estiloIngreso = libro.createCellStyle();
estiloGasto.setFont(fuenteGasto);
estiloGasto.setDataFormat(format.getFormat("#.##"));
estiloIngreso.setFont(fuenteIngreso);
estiloIngreso.setDataFormat(format.getFormat("#.##"));
.       .       .
```

6.2.5 API's AndroidPlot

Permiten dibujar varios tipos de gráficas, son una API's gratuitas y de código abierto. Yo las utilizo para dibujar las gráficas que muestran los gastos/ingresos de todo un año, agrupados por meses. A continuación muestro un fragmento de código utilizando estas API's, el resultado puede verse en la figura 6 del capítulo 5.3.

```
public void creaGraficaDeGastos(boolean actualizar_existente) {

    if(actualizar_existente)   mySimpleXYPlot.removeSeries(series1);
    //redondeamos los numeros
   double [] gM = new double[12];

   for(int i = 0; i<12;i++) {
        gM[i] = Math.round(gastosMensuales[i] * 100.0) / 100.0;
     }
    //creamos la grafica
    mySimpleXYPlot = (XYPlot) getView().findViewById(R.id.mySimpleXYPlot);
    mySimpleXYPlot.setDomainRightMin(13);
    DecimalFormat df = new DecimalFormat("#.##");
    mySimpleXYPlot.setRangeValueFormat(df);
     Number[] series1Numbers =
    {0,gM[0],gM[1],gM[2],gM[3],gM[4],gM[5],gM[6],gM[7],gM[8],gM[9],gM[10],gM[11]};
    double max = 0.0;
    for(int i=0;i<12;i++) {
       double v = gM[i];
        if(v > max){
            max = v;
         }
     }
    if(max < 9)
        mySimpleXYPlot.setRangeTopMin((int)max+ 1);
    else if(max >= 9 && max  <=50)
       mySimpleXYPlot.setRangeTopMin((int)max+(int)max*15/100);
    else
        mySimpleXYPlot.setRangeTopMin((int)max+(int)max*7/100);
    int numEtiquetas = (int) max/10;
    if(numEtiquetas == 0) {numEtiquetas = 1;}
    int modulo = numEtiquetas % 5;
    if(modulo == 1) numEtiquetas = numEtiquetas +4;
    if(modulo == 2) numEtiquetas = numEtiquetas +3;
    if(modulo == 3) numEtiquetas = numEtiquetas +2;
    if(modulo == 4) numEtiquetas = numEtiquetas +1;

    mySimpleXYPlot.setRangeLabel("");
    mySimpleXYPlot.setDomainLabel("");
     series1 = new SimpleXYSeries(Arrays.asList(series1Numbers),
     SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Meses");
    LineAndPointFormatter series1Format = new LineAndPointFormatter(
     Color.rgb(0, 0,100),
     Color.rgb(0, 100, 0),
     Color.argb(200,250,0,0),
     new PointLabelFormatter(Color.WHITE));

    mySimpleXYPlot.addSeries(series1, series1Format);
    mySimpleXYPlot.setTicksPerDomainLabel(1);
    mySimpleXYPlot.setRangeStep(XYStepMode.INCREMENT_BY_VAL,numEtiquetas);
                          .
                          .
                          .
```

6.3 El modelo de datos

El modelo de datos representa una proyección en memoria de la base de datos. Debe contener los datos que el usuario necesita visualizar en la interfaz y para ello hay que utilizar las estructuras de datos adecuadas que proporciona Java, además estas estructuras deben contenerse en un objeto al que se pueda acceder desde cualquier actividad o fragmento de la aplicación.

Para este propósito utilizo la clase DataFactory que sigue el patrón singleton (instancia única) que permite asegurar que esta clase tendrá una sola instancia, así como proporcionar un punto de acceso global a ella. Para ello se implementa un método que crea una instancia del objeto solo sí todavía no existe ninguna y si ya existe una instancia creada nos la devuelve. Esto lo conseguimos utilizando un constructor privado y creando un método que se encargue de comprobar si ya existe una instancia de esa clase.

```
public class DataFactory {

	private BD bd;
	private static DataFactory sDataFactory;
	private Context mAppContext;

	private DataFactory(Context appContext){
		mAppContext = appContext;
		bd = new BD(mAppContext);
	}

	public static DataFactory getInstance(Context ctx){

		if(sDataFactory == null){
			sDataFactory = new DataFactory(ctx.getApplicationContext());
		}

		return sDataFactory;
	}
```

Creamos este objeto pasándole como único parámetro el contexto de la aplicación, esto nos permitirá hacer cosas como instanciar un objeto que representa la base de datos y que usaremos para realizar consultas sobre la misma.

Este objeto contendrá las siguientes estructuras de datos:

```
public ArrayList<EntradaPuntual>registros;
public ArrayList<EntradaPuntual>regPorMes;
public static ListaCategorias listCat;
public static ListaFavoritos listFav;
public static ListaObjetivos listObj;
public static Vector<Operacion>operaciones;
```

Al principio el enfoque que utilice fue cargar toda la base de datos en estas estructuras. Este enfoque tiene ventajas y desventajas. Por una parte una vez iniciada

la aplicación el acceso a los datos es muy rápido ya que éstos se encuentran cargados en memoria, sin embargo, en caso de que existan muchos datos provoca que la aplicación tarde más en inicializarse y al mismo tiempo estás consumiendo mucha memoria que es un recurso valioso en teléfonos móviles.

Reflexionando sobre cual podría ser la mejor forma de tratar esto, me di cuenta de que en el momento en que los usuarios inician la aplicación solo necesitan visualizar los siguientes datos:

- Las entradas correspondientes al mes actual
- Las categorías y entradas frecuentas que hayan ido personalizando
- Los totales de gastos/ingresos mensuales del año en curso
- La lista de objetivos que pudieran haber introducido

Esto reduce mucho la cantidad de datos que necesito mantener en memoria. Imaginemos un usuario que lleve utilizando la aplicación unos 3 años con una media de unas 3 entradas al día tendríamos 365*3*3 = 3285 entradas en memoria y la mayor parte de ellas nunca serán visualizadas por el usuario, con este nuevo enfoque solo mantendremos 3*31= 93 entradas, y si el usuario desea ver entradas correspondientes a meses o años anteriores las obtendrá directamente de la base de datos. Es cierto que estas cifras no suponen un gran problema aún tratándose de un dispositivo móvil, pero como esto también es un ejercicio académico he decidido cambiar este enfoque para disminuir el uso de memoria y poder verificar si realmente representa una mejora.

A continuación comparo la cantidad de memoria utilizada en cada uno de estos dos enfoques. Para esta comparativa he utilizado Dalvik Debug Monitor Server (DDMS) que es una herramienta incluida en el Android SDK y que podemos utilizar en eclipse. Estos son los resultados obtenidos teniendo 2607 entradas puntuales y 49 entradas automáticas.

Cargando toda la base de datos en memoria

ID	Heap Size	Allocated	Free	% Used	# Objects
1	10,273 MB	8,283 MB	1,990 MB	80,63%	122.941

Cargando en memoria solo las entradas del mes en curso

ID	Heap Size	Allocated	Free	% Used	# Objects
1	8,621 MB	5,585 MB	3,036 MB	64,78%	66.443

Esta herramienta nos muestra 5 valores importantes:

- El tamaño de la pila de memoria (Heap Size): Android asigna un tamaño de pila de memoria para cada aplicación, la cual puede crecer dependiendo de las necesidades de dicha aplicación, pero solamente hasta un límite estricto, ninguna aplicación puede sobre pasar dicho límite.
- La cantidad de memoria utilizada por la aplicación (Allocated). De esta cantidad dependerá el tamaño de la pila de memoria.
- La cantidad de memoria libre en la pila (Free) y el porcentaje de pila utilizado: Esto es relativo ya que la pila puede crecer si la aplicación lo necesita, siempre que no se rebase el límite asignado.
- El número de objetos alojados en memoria por la aplicación.

Como vemos hay una diferencia importante en la cantidad de objetos que la aplicación aloja en memoria en cada uno de los dos enfoques. 122.941 en el primer caso y 66.443 en el segundo, lo que conlleva una diferencia igualmente importante en la cantidad de memoria que consume la aplicación, 8,283 MB frente a 5,585 MB en el segundo caso. Esta diferencia será cada vez mayor a medida que el usuario conserve más entradas almacenadas en la aplicación.

Además evitamos el riesgo de desbordar la memoria, ya que Android establece un límite estricto para el tamaño de pila que puede reclamar cada aplicación, este límite es un porcentaje del tamaño de la RAM que monta cada dispositivo y si una aplicación ha llegado a este límite y reclama alojar más memoria recibirá un OutOfMemoryError. Con este enfoque evitamos este problema, ya que sea cual sea la cantidad de entradas que cada usuario quiera mantener almacenadas en la aplicación solo se alojarán en memoria como máximo las del año en curso. Si el lector desea saber más sobre la gestión de memoria en dispositivos Android puede referirse a la documentación oficial en https://developer.Android.com/training/articles/memory.html.

7. Conclusiones

7.1 Objetivos conseguidos, no conseguidos y futuras extensiones

En el apartado 1.3 de esta memoria establecí 3 hitos a alcanzar durante el desarrollo del presente proyecto. Ahora con el proyecto ya finalizado voy a analizar que es lo que he conseguido y lo que no en cada uno de estos 3 hitos, así como futuras extensiones que a fecha de hoy considero interesantes.

Rediseño de la interfaz de usuario

La principal mejora ha sido introducir la ViewPager como forma de navegación entre las cinco pantallas principales. Aún así no estoy del todo satisfecho con el diseño conseguido, en concreto considero mejorable la forma de presentar la información al usuario, la apariencia de algunas pantallas y la combinación de colores. Así considero este hito parcialmente conseguido, en parte porque estos aspectos son más propios de un diseñador gráfico que de un desarrollador software.

Rediseño de la lógica de la aplicación

Este hito lo considero cumplido, después de reorganizar y hacer refactorización de código el resultado ha sido un código bastante legible y ordenado, lo que me ha permitido corregir algunos bugs. El uso correcto de las estructuras de datos y el ahorro de memoria conseguido en el modelo de datos, según explico en el punto 6.3 de esta memoria, han hecho que la aplicación se más eficiente. La reorganización del código también me ha permitido implementar más fácilmente las nuevas funcionalidades.

Añadir nuevas funcionalidades

Este objetivo también lo considero superado, lo único que me ha faltado por motivos de tiempo ha sido la opción de permitir a los usuarios crear varias cuentas y adjuntar fotografías de facturas. Sin embargo he dejado tanto la base de datos como la lógica de la aplicación preparadas para ello, así que no debería costarme mucho esfuerzo implementarlo. He implementado con éxito las siguientes funcionalidades:

- Mostrar un gráfico con los gatos/ingresos anuales
- Permitir al usuario introducir entradas recurrentes
- Posibilidad de que el usuario comente la aplicación en Facebook
- Permitir establecer objetivos de gasto
- Exportar a Excel
- Crear respaldos diarios de la base de datos automáticamente y permitir al usuario crear respaldos manualmente

En cuanto a futuras extensiones tengo en mente las siguientes ideas:

- Que la aplicación pueda ser compartida entre un grupo de personas, es decir, que se puedan crear cuentas de usuario y varios usuarios puedan introducir gastos e ingresos. Esto podría ser útil, por ejemplo, en el caso de un matrimonio para administrar gastos comunes. Esta extensión implicaría el uso de un servidor, donde se almacenarían los datos de las cuentas compartidas.
- Incluir un servicio web para cambio de moneda, en caso de que el usuario utilice la aplicación cuando este de vacaciones en un lugar donde se utilice otro tipo de moneda.
- Incluir una calculadora básica, en la pantalla donde se introducen los gastos. Esto ha sido una petición de algunos usuarios a través de los comentarios que dejan en Google Play. El objetivo es que el usuario pueda usarla para sumar diferentes cantidades que quiera introducir como un mismo gasto.
- Desarrollar la aplicación para iOS y Windows Phone, bien de forma nativa o utilizando alguno de los enfoques mencionados en este proyecto para crear aplicaciones hibridas.

7.2 Conocimientos adquiridos en el grado que he empleado en este proyecto

En este apartado voy a mencionar las asignaturas cursadas en el grado que más me han ayudado en el desarrollo de este proyecto.

- Ingeniería del software. Adquirí conocimientos de modelado y diseño conceptual así como patrones arquitectónicos para el desarrollo de un producto software.
- Programación. Adquieres las bases que te permiten comprender como funciona un lenguaje de programación, se aprende utilizando Java, pero muchos conocimientos son extensibles a cualquier otro lenguaje orientado a objetos.
- Estructuras de datos y algoritmos. Aprendes a utilizar correctamente las distintas estructuras de datos que te proporcionan los lenguajes de programación. Esto te permite desarrollar software más eficiente, algo importante y más tratándose de dispositivos móviles.
- Bases de datos. Adquieres bases sólidas sobre SQL, esto me ha permitido trabajar con SQLite sin ningún problema.
- Competición de programación. Adquieres habilidades en el diseño de algoritmos que resuelvan un problema concreto, así como a escribir código eficiente.
- Desarrollo centrado en el usuario. Me proporcionó los conocimientos necesarios para desarrollar una interfaz gráfica intuitiva y fácil de usar.

7.3 Experiencia personal en el desarrollo del proyecto

La realización de este proyecto me ha resultado gratificante desde un punto de vista personal. Podría haber elegido algunos de los proyectos propuestos por profesores, pero preferí hacerlo sobre desarrollo para teléfonos móviles porque es un tema que me gusta y el haber podido llevar a la práctica casi todas las cosas que me propuse al inicio del mismo es algo de lo que me siento orgulloso.

Para desarrollar la aplicación que presento en este proyecto he tenido que leer mucha documentación, algún libro y repasar muchas cosas de las asignaturas que he mencionado en el apartado anterior, pero esta vez desde un punto de vista práctico. Esto me ha permitido entender mejor los conceptos teóricos, ya que cuando te chocas con los problemas reales a la hora de desarrollar un producto software y tienes que solucionarlos es cuando realmente entiendes la teoría que se suele aprender una forma más abstracta. El tener que buscar y entender información sobre la plataforma Android también me ha ayudado a mejorar mi capacidad de auto aprendizaje, algo muy importante hoy en día, ya que la información disponible en internet es inmensa y tienes que aprender a reconocer rápidamente que partes pueden ayudarte a resolver tu problema y cuales no.

Por último si algún lector está interesado en probar mi aplicación puede descargarla de Google Play siguiendo este link:

https://play.google.com/store/apps/details?id=com.logica.controlspends&hl=es

Bibliografía

Asignatura Desarrollo Centrado en el Usuario. *Curso 2015-2016 (Temas 2,3,5)*. UPV, DSIC.

Asignatura Ingenieria del Software. *Curso 2012-2013 (Tema 6)*. UPV, DSIC.

Gironés, J. T. (2013). *El Gran Libro de Android* (3rd Edición ed.).

Google. *Android Developers*: https://developer.android.com

Google. *Google Ads*: https://developers.google.com/ads/

Google. *Google Analytics*: https://www.google.es/intl/es/analytics/

Phillips, B. (2013). *Android Programming: The Big Nerd Ranch Guide* (1st Edición ed.).

PhoneGap: http://phonegap.com

SQLite: https://www.sqlite.org

Stackoverflow: http://stackoverflow.com

Wikipedia: https://es.wikipedia.org

Xamarin: https://www.xamarin.com