

UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T.Telecomunicación (Sist. Electrónicos)



UNIVERSIDAD
POLITECNICA
DE VALENCIA



ESCUELA POLITECNICA
SUPERIOR DE GANDIA

**“Creación de una aplicación,
programada en Java, para smartphones
basados en el sistema operativo
Android para un portal turístico.”**

TRABAJO FINAL DE CARRERA

Autor:

Martin Puig Tenschert

Director:

Dr. Jordi Mauri Castelló

GANDIA, 2012

Índice:

	página
0. Introducción, estado de arte, motivación personal	1
1. Telecomunicaciones y Turismo	3
2. Smartphones	4
2.1 Interfaces y Protocolos	4
2.2 Sistemas Operativos	9
2.3 Aplicaciones	10
3. Móviles Android	12
4. Lenguaje de Programación Java	13
4.1 Ficheros y Códigos	13
4.2 Entornos y Herramientas	14
4.3 Clases, Objetos y Funciones	14
4.4 Relaciones y Diferencias con C++	16
5. Activity	17
6. Entorno de Programación	19
6.1 Instalación	19
6.2 Configuración del Entorno	21
6.3 Partes de una aplicación	23
6.4 El Entorno	24
6.5 El Emulador	27
7. La Aplicación de Gandía	30
8. Las Pantallas	32
9. Programación del Código Fuente	37
9.1 Ficheros	37
9.2 Objetivos	45
9.3 Otros Códigos	68
10. Prueba en Varios Dispositivos	77
11. Publicación de una Aplicación	80
12. Estadísticas	82
13. Conclusiones	85
14. Bibliografía	86
15. Índice de Imágenes	88
16. Anexos	90

“Programming is 10% science, 20% ingenuity, and
70% getting the ingenuity to work with the science.”

-Anonymous

0. Introducción

En el presente proyecto final de carrera de la carrera de Sistemas Electrónicos se describe la creación de una aplicación (programa) para teléfonos móviles basado en el sistema operativo Android. Se trata de una aplicación para uso en el sector turístico que podrá ser usada tanto por turistas del destino como guía como por habitantes de la ciudad para recibir información de diferente índole.

La aplicación se programa en lenguaje Java y se intentará aprovechar al máximo los recursos ofrecidos por los nuevos teléfonos (smartphones) como el GPS, conexión a Internet, la brújula, pantalla táctil, etc. Finalmente la aplicación, como producto final, será publicada como versión gratuita en la página de aplicaciones de Android (AndroidMarket) para que pueda ser usada por todos los usuarios.

Estado de arte:

La rápida evolución de la tecnología, sobre todo en el sector de las telecomunicaciones, ofrece muchas posibilidades para la creación de nuevos productos. Desde hace pocos años los teléfonos smartphones están viviendo un gran crecimiento. Entre sus tareas básicas se encuentran funciones con GPS, brújulas, pantallas táctiles, cámaras de fotos con varios megapíxeles, etc. Algunos de los modelos más actuales (año 2012) son capaces proyectar imágenes con un cañón integrado en el dispositivo o de mostrar imágenes en sus pantallas en tres dimensiones. Los móviles más modernos ofrecen conexiones rápidas a través de 4G. A pesar de usar procesadores más rápidos, se están consiguiendo consumos de batería más bajos. Existen algunos dispositivos con cámaras de fotos de 8 Megapíxeles. Algunos móviles o funciones, no están disponibles (o no se han extendido lo suficiente) aun en el mercado europeo. Es el caso por ejemplo del NFC que ofrece la posibilidad de realizar pagos en tiendas a través del teléfono.

Smartphones modernos como el Nexus S son capaces de sobrevivir condiciones extremas. En una competición de Google se enviaron varios de éstos móviles como sondas meteorológicas al espacio (30 km) y soportaron temperaturas de hasta -50 °C. [1]

En cuanto a las aplicaciones, se han puesto de moda aquellas relacionadas con redes sociales. Se trata de las aplicaciones más descargadas. Un campo interesante también es la realidad aumentada o augmented reality (AR). En las aplicaciones AR se mezcla información del mundo real con el mundo virtual. Para ello se hace uso sobre todo de la cámara de fotos, brújula, el GPS y una base de datos. De esta manera al enfocar con el smartphone sobre algún lugar, en la pantalla del móvil puede aparecer información de interés sobre este sitio. Este es el método que usan algunas aplicaciones (sacan información de una base de datos a partir de datos de GPS y brújula). La realidad aumentada real en cambio, identifica con la cámara un objeto y muestra información sobre éste. Para ello el smartphone necesita hacer operaciones más difíciles al interpretar una fotografía.

[1] AREAMOBILE

La evolución se realiza de forma exponencial y será interesante conocer las novedades de los próximos años. En los siguientes puntos de este proyecto de carrera se encuentra una descripción más profunda de las características de los teléfonos actuales.

Motivación personal:

Como estudiante de electrónica tengo un interés especial en las nuevas tecnologías y su funcionamiento. Debido a los altos costes de creación de dispositivos electrónicos de cualquier índole que resultan ser unas barreras de entradas grandes para una única persona, decidí optar por la programación de software ya que permiten crear productos finales sin grandes costes económicos. Los conocimientos adquiridos durante la carrera y esfuerzo personal permiten crear un producto final (software) útil que puede ser usado por muchas personas.

Como diplomado en turismo y administrador del portal turístico www.gandiaturistica.com quise combinar ambas carreras en mi proyecto final de carrera de sistemas electrónicos. Ante la ausencia de una aplicación turística de Gandía, decidí programar esta aplicación en cooperación con la empresa Aplicae en la cual realicé prácticas de empresas durante el último cuatrimestre de la carrera.

1. Telecomunicaciones y turismo.

El sector turístico pertenece al sector terciario (de servicios) y es el sector más importante en España. Según la Organización Mundial del Turismo (OMT), España es el segundo país en cuanto al número de turistas recibidos (después de Francia y por delante de Estados Unidos).

A pesar de que el ser humano viajó desde siempre (por diferentes motivos), fue a partir de la revolución industrial (siglo XIX) cuando muchas personas comenzaron a viajar por motivos turísticos (descanso, cultura, salud, etc.). El gran boom turístico se produjo entre 1950 y 1973 cuando el sector turístico internacional comenzó a crecer a un ritmo mucho mayor a todos los años anteriores durante la historia.

Desde los comienzos del turismo, este sector ha vivido muchos cambios importantes. Hoy existe una gran variedad de oferta en cuanto a destinos turísticos, tipos de turismo y medios de transportes. Lo que ha permanecido igual es la necesidad del turista de informarse sobre su destino y su viaje. Por ello es importante ofrecerle información turística sobre medios de transportes, alojamiento, lugares de interés, etc. Hasta hace unos años, esta información se ofrecía en folletos y guías turísticas impresas. Desde hace unos años, está teniendo más importancia la información turística digital. Ya es una tarea habitual buscar información del destino en páginas webs, reservar vuelos y alojamiento a través de los portales especializados en este sector y guiarse con ayuda de mapas digitales y el sistema de geoposicionamiento GPS (Global Positioning System).

Desde hace relativamente poco tiempo es posible ofrecer esta información también en dispositivos móviles como en PDAs (Personal Digital Assistant) o smartphones. Sobre todo los smartphones están viviendo un gran crecimiento. Entre las aplicaciones (Apps) que se distribuyen para estos nuevos dispositivos, en la lista de las más descargadas se encuentran en segundo lugar (después de las aplicaciones para redes sociales) las aplicaciones relacionadas con mapas y geoposicionamiento. Estas aplicaciones son usadas sobre todo en el sector turístico y como sistema de navegación en general.

Son ya varios destinos turísticos (no tantos en España a pesar de ser un destino turístico muy importante) que ofrecen aplicaciones propias que incluyen mapas, lugares de interés, etc. Aunque las nuevas tecnologías están avanzando y sustituirán finalmente a los sistemas tradicionales, aún hay muchas empresas y ayuntamientos en España que no se dan cuenta de la importancia de estas nuevas tecnologías y de las oportunidades que éstas les ofrecen. De este modo sigue habiendo Hoteles con sistemas de reservación y facturación basados en pantallas Dos (Ms-Dos) y con versiones de Office anticuadas que no son capaces de abrir ficheros nuevos (esto sucede en varios Hoteles de Gandía).

En tiempos de la web 2.0 y la rápida evolución tecnológica, las empresas y los ayuntamientos deberían estar más abiertos a estas nuevas tecnologías.

2. Smartphones

Los smartphones son teléfonos móviles que ofrecen más opciones que los teléfonos móviles tradicionales. Debido a esto están viviendo un fuerte crecimiento. Según estudios de la compañía Nielsen [2], que realiza estudios de mercado, el 49% de los móviles españoles ya son smartphones (datos del tercer cuatrimestre del 2011). Durante el primer cuatrimestre sólo el 39,2% de los móviles españoles eran smartphones [3]. Las características básicas de un smartphone son sus interfaces y la conexión de Internet con la posibilidad de instalar aplicaciones adicionales. Dependiendo del fabricante, los smartphones pueden basarse en diferentes sistemas operativos. Según un estudio de Socialcast del año 2011, en Estados Unidos, el 39% de los hombres y el 31% de mujeres poseen un smartphone. En un estudio por edades resultó que el 52% de los jóvenes entre 18 y 29 años y el 45% de los comprendidos entre los 30 y 49 años tienen un smartphone. También los ingresos juegan un papel importante ya que a mayores ingresos, mayor porcentaje de compras. El 59% de personas con ingresos superiores a 70.000 \$ tienen un smartphone mientras que de las personas con ingresos menores de 30.000 \$ sólo el 22% posee estos móviles. El mismo caso se da al comparar el nivel de estudios de los propietarios de los smartphones. A mayores estudios, mayor número de usuarios. En el estudio de Socialcast también se descubrió que el sistema operativo más usado es el Android. En los anexos 6 y 7 se encuentran los resultados de los estudios de Socialcast y Wilson Electronics (fabricante de accesorios para móviles) con datos interesantes y curiosos sobre los hábitos de los usuarios de móviles tradicionales (Wilson) y de smartphones (Socialcast).

2.1 Interfaces y protocolos

Los smartphones poseen varios interfaces que le permiten al usuario una utilización cómoda e intuitiva del teléfono. Durante la planificación de un teléfono nuevo los fabricantes tienen mucho en cuenta el uso intuitivo y fácil del dispositivo. De este modo la mayoría de smartphones disponen de una pantalla táctil que facilita mucho la navegación por los diferentes menús del dispositivo. Las pantallas táctiles detectan la pulsación por un usuario mediante cambios de corriente eléctrica en las diferentes capas de la pantalla táctil. Esto se puede lograr de diferentes formas por lo que se diferencian fundamentalmente dos tipos de pantallas táctiles. Cada una de ellas tiene ventajas e inconvenientes que se detallan a continuación.

2.1.1 Las primeras pantallas creadas para este tipo de móviles fueron las pantallas resistivas [4]. Su funcionamiento básico es el siguiente. Al presionar con el dedo o un objeto sobre la pantalla, se unen dos capas de material conductor en ese punto. El microcontrolador del dispositivo puede de esta forma calcular la posición exacta en la que se produjo la acción. Para ello normalmente las capas resistivas son organizadas en forma de matriz. Debido a que se tiene que pulsar físicamente sobre la pantalla para conseguir la unión de los materiales conductores, estas pantallas suelen ser más blandas. La ventaja de las pantallas blandas es su

[2],[3] NIELSEN

[4] LG

mayor resistencia a caídas y golpes. Además suelen ser más exactas a la hora de calcular la posición en la que se pulsó y permiten no sólo la navegación con los dedos sino también con otros objetos (“lápices”, guantes, etc.) y su coste de fabricación es relativamente bajo. La ventaja de la pantalla blanda puede ser al mismo tiempo un inconveniente al rallarse fácilmente si se usan objetos puntiagudos. Las imágenes de las pantallas no son tan claras como en otras pantallas al usarse varias capas de diferentes materiales en la pantalla lo que empeora la visibilidad. Estas capas también producen más reflexiones de luz solar.

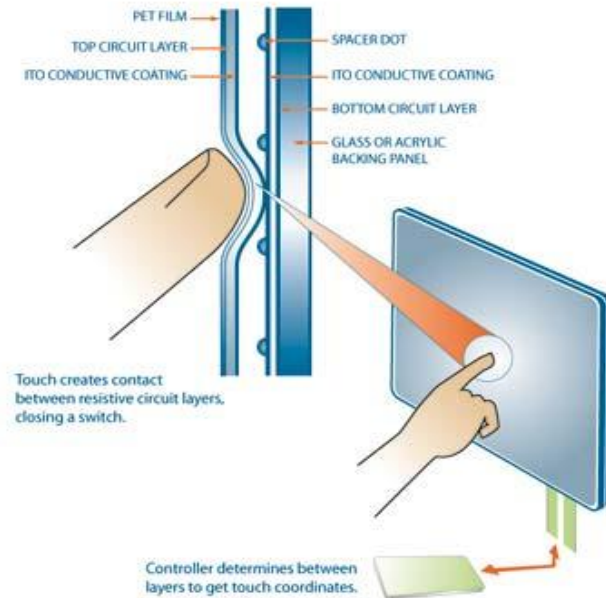


Imagen 2.1 Pantalla resistiva

En la imagen 2.1 se muestra el uso de las diferentes capas de una pantalla táctil resistiva. En su parte interna se encuentra la capa de cristal o material acrílico en la que se muestra la información del dispositivo (la pantalla). A continuación se encuentran las capas de la parte táctil

de la pantalla. La segunda capa contiene el circuito electrónico posterior de la pantalla que se conecta con la capa ITO (Indium Tin Oxide/óxido de indio) de material conductor [5]. Este material, en finas capas, es transparente y no tiene color. Su composición es del 90% de In_2O_3 y del 10% de SnO_2 . En el espectro infrarrojo el material se comporta como un espejo metálico.

Esta capa está separada, mediante aire y pequeñas gotas o puntos (spacer dot) de material no conductor, de la capa conductora ITO que a su vez está conectada a la capa del circuito electrónico superior de la pantalla (la unión de las dos capas ITO cierra el circuito electrónico). Por último se usa una capa de plástico blando PET (Polyethylene terephthalate) como protección.

2.1.2 Las pantallas capacitivas [6] son pantallas más modernas y las que actualmente se usan en los smartphones de gama más elevada. Igual como en las pantallas anteriores se detecta las acciones de pulsación mediante el cambio de corrientes (o cambio del campo electro-magnético) y los datos son calculados e interpretados en el microcontrolador del smartphone.

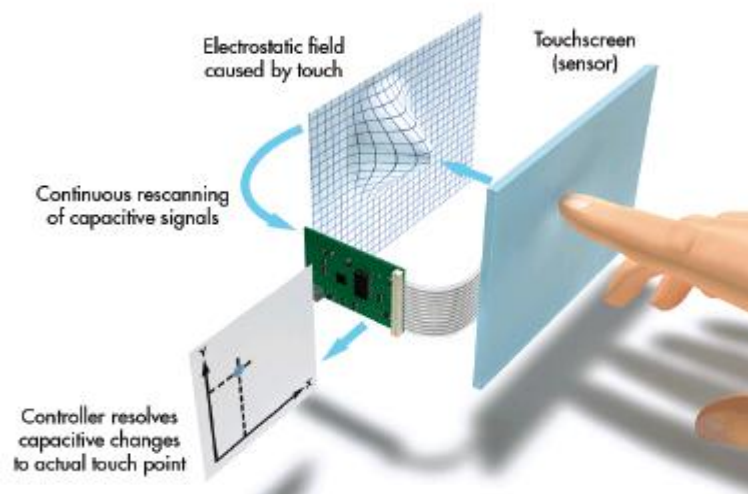


Imagen 2.2 Pantalla capacitiva

[5] WIKIPEDIA [6] HOWSTUFFWORKS

La forma de recibir esta información es bastante diferente a la anterior. Los sensores capacitivos usados en circuitos electrónicos son alimentados con tensión alterna. La estructura física del sensor es la siguiente: Dos o más capas de material conductor están separadas por medio de un dieléctrico (material no conductor). La corriente de salida del sensor depende del área de las capas conductoras (proporcional) y la separación entre ambas (inversamente proporcional). Entre ambas placas del condensador se crea un campo eléctrico E.

En el caso de las pantallas táctiles capacitivas se tiene, igual como en el caso de las pantallas resistivas, una red o matriz de pequeños sensores. En este caso sensores capacitivos (cada uno de dos capas). Una de ambas capas es alimentada con corriente alterna a una frecuencia determinada. De esta forma se crea una capacidad básica, que aquí llamaremos “capacidad natural”, entre ambas capas. Si se acerca un dedo para pulsar sobre la pantalla, se “rompe” esta “capacidad natural” (al estar el dedo/persona conectada a “tierra”). El microcontrolador detecta este cambio de capacidad en la segunda capa (parte receptora) y calcula su posición. Esta forma de calcular la pulsación en la pantalla es más complicada y consume más recursos para los cálculos. Cada pulsación (y la grasa de los dedos que se pasa a la pantalla) provoca variaciones de la “capacidad natural” del dispositivo por lo que el microcontrolador tiene que tomar ajustes constantes. Una de las ventajas de este tipo de pantallas es que al usar menos capas que las pantallas resistivas, la imagen de la pantalla es mucho más clara. La pantalla es también más sensible al no ser necesario tener que pulsar físicamente sobre la misma. El teléfono detecta la presencia del dedo a poca distancia de la pantalla táctil. El uso de pantallas capacitivas permite mejorar la navegación al permitir acciones de “multitouch”. Esto es el uso de gestos como realizar un zoom con dos dedos. Las pantallas son más duras por lo que no se rallan tan fácilmente. El inconveniente de las pantallas duras es que una caída o un golpe fuertes puede romper la pantalla.

En el Anexo 1 se encuentran más imágenes sobre el funcionamiento de estas pantallas

2.1.3 Teclados y botones

En el pasado los móviles disponían de un número muy limitado de teclas. A parte del teclado numérico y botones de navegación (menú, derecha, izquierda, Ok) había pocos botones más al no hacer falta en estos dispositivos. Los smartphones pueden tener varios teclados y botones diferentes.

Todos los smartphones que disponen de pantalla táctil (la mayoría) disponen también de un teclado virtual que aparece en la pantalla. El usuario pulsa sobre la misma pantalla para escribir.

En la mayoría de los casos el teclado virtual está oculto y se activa al pulsar sobre un campo de texto o seleccionándolo desde el menú. Adicionalmente otros teléfonos modernos llevan un teclado físico tipo “Querty” (como los teclados del ordenador cuyas primeras letras en la fila superior son q,w,e,r,t,y) incorporado, bien debajo de la pantalla (como diferentes modelos de Blackberry) o bien como teclado extraíble en el interior del móvil.



Imagen 2.3 Teclados (numérico tradicional, Querty y softkey)

A parte de los teclados existen diferentes botones para la navegación. Estos pueden ser como en el caso de los teclados virtuales en las pantallas (llamadas soft-key) o como botón físico (hard-key). Sus funciones son generalmente: abrir menús, seleccionar, control del volumen y atrasar. Las teclas hard-key pueden variar según el modelo de teléfono por lo que hay que tenerlo en cuenta a la hora de desarrollar una aplicación móvil.

Los botones ópticos son menos utilizados y su función es la misma como la de los hard-keys normales. Encontramos estos botones “trackball óptico” en smartphones como el Desire de HTC. El botón incorpora un emisor de luz y un receptor (sensor fotoeléctrico). Al pasar con el dedo varía la luz reflejada en el sensor. Este cambio es detectado por el microcontrolador y permite una navegación sin el uso de la pantalla táctil. Los trackballs ópticos facilitan la navegación por páginas webs al seleccionar automáticamente los enlaces.

2.1.4 Protocolos

Para la recepción de datos se utilizó hasta el momento la red telefónica móvil 2G. Las aplicaciones de los smartphones en cambio necesitan una red que proporcione una transferencia de datos más rápidas para actualizar aplicaciones o el sistema operativo, recibir información de páginas webs, actualizar el pronóstico del tiempo, etc. Debido a esto, los smartphones tienen que conectarse a Internet mediante la red móvil 3G y redes WIFI (Wi-Fi es un nombre y no significa “Wireless Fidelity” como se puede leer en muchas fuentes). El término WIFI define todos los dispositivos WLAN (Wireless Local Area Network) basados en los estándares 802.11 de la IEEE (Institute of Electrical and Electronics Engineers).

La conexión 2G opera en modo GMS (Global System for Mobile Communications) usando GPRS (General Packet Radio Service) con la que se alcanzan unas velocidades de hasta 220 kbps (teóricamente). La conexión 3G en cambio usa la red UMTS (Universal Mobile Telecommunication System) [6] y puede alcanzar velocidades de hasta 14,4Mbps (en condiciones óptimas). Según cada operador y contrato, las velocidades y volúmenes de tráfico varían. Pero la velocidad de transmisión 3G será siempre superior a la ofrecida por la red 2G.

[7] WIKIPEDIA

Los móviles modernos son capaces incluso de actuar como “hotspot” y ofrecer conexión de Internet a otros dispositivos. De esta forma es posible conectarse a Internet con un ordenador de sobremesa a través del teléfono móvil. Para ello es suficiente con activar dicha función en el smartphone, establecer una contraseña de red y conectarse a esta red con el ordenador usando la misma contraseña. Los inconvenientes de esta conexión son que sin cobertura 3G no funciona la conexión, el volumen de tráfico mensual limitado (según contrato del operador) y la velocidad de conexión que puede ser inferior a una conexión ADSL por cable.

Al igual que algunos móviles tradicionales, los smartphones soportan también el protocolo Bluetooth para intercambiar ficheros o para conectarse con otros dispositivos (a muy cortas distancias) como por ejemplo los “manos libres”. El protocolo más novedoso es el NFC (Near Field Communication). Se trata de un estándar para la transmisión de datos en campo muy cercano. La conexión se puede realizar mediante dos dispositivos activos o mediante uno activo y otro pasivo, usando un Tag RFID pasado en la ISO 14443 o ISO 15693. Durante los próximos años el número de smartphones con esta tecnología aumentará. De momento, según el uso actual, parece que el uso de NFC seguirá creciendo en Japón (donde ya se usa en algunos bancos para realizar transferencias) pero no vivirá un crecimiento parecido en Europa en los próximos tiempos. El NFC presenta algunas ventajas que serían interesante aprovechar. Durante la compra en tiendas, el NFC podría ser una forma alternativa de pago. Para ello sería necesario equipar las tiendas con los lectores correspondientes lo que a nivel nacional supondría grandes inversiones. Algunas grandes empresas reconocen la oportunidad y las ventajas que ofrece esa nueva tecnología y están trabajando en dispositivos que se comunican por NFC (como por ejemplo algunos puntos de información en estaciones de trenes grandes en Alemania).

2.1.5 Otras características

La mayoría de smartphones disponen de entre una y dos cámaras digitales integradas. La principal es usada para hacer fotos y grabar vídeos. Esta cámara de fotos suele hacer fotografías con una resolución de entre 3 y 5 megapíxeles. Esto equivale ya casi a los megapíxeles usados en una cámara de fotos digital corriente. La segunda cámara de los smartphones (no todos la incluyen) puede encontrarse en la parte delantera o trasera y funciona como webcam para videoconferencias.

La mayoría de smartphones llevan receptores de GPS (Global Positioning System) integrados. El dispositivo capta las señales de los diferentes satélites y calcula a partir de ellos su posición (a través de la triangulación). Gracias al GPS los móviles pueden ser usados como sistemas de navegación o para mostrar información relacionados con el entorno (en caso de aplicaciones turísticas por ejemplo). Entre todas las aplicaciones para smartphones descargadas, se encuentran las aplicaciones de mapas en segundo lugar (después de las aplicaciones de redes sociales), lo que demuestra que el GPS es una de las funciones fundamentales de los smartphones.

Otras ventajas que diferencian los smartphones de los móviles tradicionales es el funcionamiento parecido al de ordenadores y portátiles. Es posible de esta forma escuchar

música, ver películas, trabajar con archivos y documentos, jugar y mucho más. A medida que la tecnología avanza será posible realizar cada vez más tareas con los smartphones.

2.1.6 Diferencias entre smartphones

El mercado de los smartphones es un mercado relativamente nuevo en la que algunas empresas potentes de electrónica e informática intentan luchar por dominar el mercado. Debido a eso existen muchos tipos de smartphones con sistemas operativos distintos. Los diferentes dispositivos pueden contar con varios interfaces, accesorios y características enumeradas anteriormente. La calidad y el precio pueden variar considerablemente. Así es posible conseguir algunos móviles de forma gratuita con nuevos contratos en una compañía telefónica mientras que para conseguir otros móviles hace falta invertir tanto dinero como para un ordenador nuevo. En algunos casos (como en iPhones) se paga mucho dinero por el nombre y la marca. El precio de fabricación de un iPhone es de aproximadamente 140 euros [8] (comprando los componentes a gran escala). El precio de venta del iPhone 4 ronda según el modelo entre 500 y 700 euros [9].

Algunos smartphones disponen de más de un procesador lo que permite realizar grandes tareas en menos tiempo, importante para aplicaciones 3D con OpenGL (plataforma y lenguaje para gráficos informáticos).

Teléfonos como el GW820 eXpo de LG son capaces de proyectar fotos y vídeos mediante un pequeño cañón integrado. Las imágenes proyectadas equivalen a una pantalla de hasta 40 pulgadas (101,6 cm). El smartphone Optimus 3D del mismo fabricante permite grabación y visualización en tres dimensiones sin necesidad de gafas especiales. También los smartphones Evo 3D de HTC o el Sharp Lynx ofrecen este tipo de pantallas.

Motorola en cambio distribuye un smartphone llamado Defy que según el fabricante está completamente protegido contra agua.

Se observa que hay muchas clases diferentes de móviles para todos los gustos pero la mayor diferencia entre los smartphones es posiblemente el sistema operativo que se analizará en el siguiente punto.

2.2 Sistemas Operativos.

Al principio fue Nokia la empresa que lanzó al mercado móviles con funciones únicas en el sector. Hoy en día sus móviles, con el sistema operativo Symbian, ya no pueden competir con la oferta de otros smartphones como los iPhones y los móviles Android. Pese a ello, igual debido a su precio competitivo, en algunas estadísticas aparecen aún como líder de mercado. Realmente de moda está el iPhone. Su uso intuitivo y el diseño moderno lo han convertido en uno de los móviles más vendidos en todo el mundo. Apple, la empresa fabricante del iPhone y su sistema operativo iOS, se ha convertido en una marca con muchos seguidores. El coste elevado de sus productos y contratos hace que sus clientes (al estar acostumbrado a pagar bastante dinero) estén dispuestos a pagar también dinero por aplicaciones adicionales.

[8] FR_ONLINE

[9] APPLE

Android es el sistema operativo de Google. Al ser un sistema abierto, gente con conocimientos de programación pueden crear sus propias aplicaciones y distribuirlas (como se verá en el punto siguiente). El inconveniente de esta gran oferta de aplicaciones y que los usuarios no estén acostumbrados a pagar dinero por software o servicios, hace que los ingresos por aplicaciones son mucho inferiores comparado con las aplicaciones para iPhones. Android ha alcanzando ya al iPhone en cuanto a ventas de teléfonos y seguramente también lo superará en el futuro en cuanto a ingresos ya que diferentes empresas como HTC, LG, Sony Ericsson, Samsung, etc. usan este sistema operativo tanto para smartphones como para tablets (tabletas digitales). El sistema operativo de teléfonos Blackberry (Blackberry OS) creado por RIM (Research in Motion) tiene un porcentaje parecido al de Android, pero decrecerá en el futuro. Los smartphones Microsoft (Windows Phones) representan un número mucho menor que los citados anteriormente.

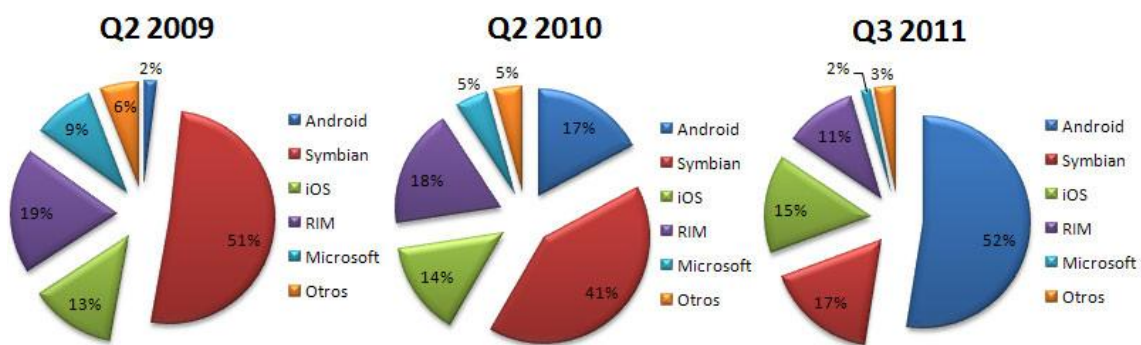


Imagen 2.4 Cuotas de mercado. Estudio de Gartner [10 y 11]

En la gráfica de la empresa Gartner (estudios de mercados tecnológicos) se observa la gran evolución que tuvo el sistema operativo Android durante los últimos años. Pasó de tener una cuota de mercado del 1,8% (2009) al 52% (2011) adelantando a los sistemas operativos de Apple y de Microsoft. Los claros perdedores de esa evolución son Symbian (sistema operativo de Nokia) y Windows Mobile.

2.3 Aplicaciones:

Mientras que la mayoría de móviles antiguos son sistemas cerrados que no permiten ser actualizados o incluir adicionalmente programas nuevos, los smartphones, como todos los ordenadores permiten justamente esto. Las diferentes aplicaciones (Apps) son programas informáticos diseñados para móviles. Para cada sistema operativo hace falta una versión diferente de una aplicación. Una aplicación para iPhone no es compatible con Android o Symbian. Para cada sistema operativo hacen falta lenguajes de programación

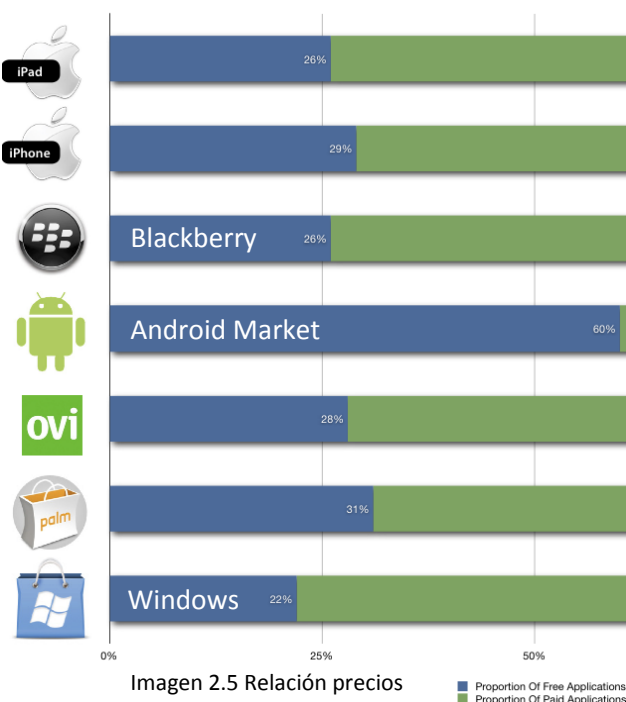


Imagen 2.5 Relación precios

■ Proportion Of Free Applications
■ Proportion Of Paid Applications

diferentes con sus respectivas librerías. También los entornos de programación varían. Los móviles para los que más personas programan aplicaciones son el iPhone y los Android. Las aplicaciones para iPhones son programadas en lenguaje Objective C (una variante del C muy orientada a objetos). Las aplicaciones para el sistema operativo Android en cambio se programan en Java. Las aplicaciones se pueden distribuir en las diferentes plataformas. En el “Appstore” de Apple se pueden distribuir las aplicaciones de iPhones y iPads. Para ello el programador tiene que pagar una tasa anual de aproximadamente 80 euros. Todas las aplicaciones son analizadas por Apple antes de ser publicadas. Para publicar aplicaciones en el “Android Market” el programador tiene que pagar una tasa única de 25 euros. Las aplicaciones no son analizadas como en el Appstore por lo que se publican bastantes programas con “bugs” (fallos). Dependiendo del mercado en el que se venden las aplicaciones, éstas suelen ser mayoritariamente de distribución gratuita o costar muy poco dinero. Sólo un tercio de las aplicaciones ofrecidas en el Android Market son de pago, de las cuales la mitad se venden por menos de un euro. Algunas pocas aplicaciones de este market cuestan 25 dólares. Siempre hay excepciones. Una aplicación para iPhones fue borrada del Appstore por costar 1000 dólares y sólo sacar en pantalla una imagen de un diamante con un texto “I’m rich” (soy rico). La aplicación fue descargada ocho veces. Varios programadores del Android Market están intentando vender aplicaciones parecidas por 100 euros (sin éxito).

En la gráfica se observa de relación entre las aplicaciones de pago (color verde) y las aplicaciones de distribución gratuita (en azul) en los diferentes mercados. Se observa que el 60% de las aplicaciones del mercado de Android son gratuitas mientras que sólo el 29% de las aplicaciones del iPhone lo son. Estos datos fueron publicados por la empresa Distimo que ofrece varias estadísticas sobre la evolución de los appstores.

El gran número de aplicaciones gratuitas en Android puede atraer a muchos clientes que se decidan por la compra de un teléfono Android. Por otro lado no parece un mercado demasiado interesante para programadores que quieran ganar dinero. Son muy pocos los que consiguen desarrollar una aplicación que se ponga de moda y con la que se puedan ganar miles o incluso millones de euros como por ejemplo con la aplicación “Car Locator” (simplemente memoriza la posición GPS del coche aparcado y ayuda a volver a encontrarlo) o el juego Angry Birds. Con este juego (uno de los más rentables) los desarrolladores ganaron más de 50 millones de euros pero tuvieron que hacer una inversión de 100.000 euros para su desarrollo y actualización.

3. Móviles Android:

Tal y como se ha citado en el punto 2.2 Sistemas Operativos, son ya varias empresas las que distribuyen sus propios smartphones con el sistema operativo Android (incluso con diferentes entornos y versiones). Esto hace que el programador tiene que tener en cuenta que algunas funciones de sus aplicaciones pueden no ser soportadas por algunos teléfonos. Los problemas más habituales son los diferentes tamaños de pantallas (una misma aplicación puede no verse igual en dos móviles diferentes), las versiones (algunas funciones no estarán disponibles en algunos móviles) y las teclas tipo “hard-key” ya que algunos móviles pueden tener más teclas y otros pueden usar sólo la pantalla táctil. Debido a esos problemas es esencial planear una nueva aplicación bien y testarla en diferentes dispositivos.

La ventaja de los móviles Android es que existen menos limitaciones para la creación de aplicaciones pues se trata de un sistema abierto (código abierto) y permite reemplazar incluso las aplicaciones que llevan los móviles de fábrica. Esto permite personalizar un smartphone al máximo con los gustos individuales del usuario.

El hecho de que varias empresas construyan teléfonos para este sistema operativo y muchas personas y empresas programen aplicaciones nuevas, hace que la evolución de esta tecnología sea cada vez más rápida.

Debido a la variedad de empresas que investigan para desarrollar nuevos móviles para el sistema operativo Android, será seguramente éste el que en el futuro podrá convertirse en el líder de mercado. Además ya hay varios modelos de tablets que utilizan este sistema. En el próximo futuro saldrán a la venta también portátiles con este mismo sistema operativo.

Los móviles de Android tienen un núcleo de Linux como interfaz para el hardware lo que lo convierte en un sistema muy estable y seguro. Cada aplicación necesita unos permisos para ejecutarse. Esto impide (normalmente) que se ejecute contenido maligno como virus y troyanos o que una aplicación capture información personal del usuario. Esto se verá más detalladamente en el punto 9.1.3 (Android Manifest). Para poder manipular aspectos críticos del sistema, tanto como programador o usuario, es necesario tener derechos de “root”, un tipo de “super-administrador” con todos los derechos. Para ello se tiene que “rootear” el dispositivo. Cosa que no se recomienda porque puede generar huecos de seguridad, provocar fallos en el sistema y perder la garantía del dispositivo.

En los siguientes apartados se explicará el entorno de programación para móviles Android y el lenguaje de programación utilizado. Se explicará la forma de programar mediante un ejemplo práctico. Se tratará de una guía turística que hace uso de varios recursos que ofrece el sistema operativo Android y sus smartphones.

4. El lenguaje de programación Java

Las aplicaciones para el sistema operativo Android se programan en el lenguaje de programación Java. Java es un lenguaje orientado a objetos creado en el año 1995 por la empresa Sun Microsystems que desde el año 2010 es propiedad de Oracle Corporation. El nombre que le dieron sus creadores al nuevo lenguaje es el de un fuerte grano de café (el preferido por estos programadores). De allí viene también el logotipo de la taza de café. Como



Imagen 4.1
Logotipo Java

curiosidad: Los cuatro primeros bytes (32 bits) de cada clase de java tienen siempre el valor “mágico” 3405691582 que en hexadecimal es 0xCAFEBABE e incluye las palabras “cafe” y “babe”.

Es importante no confundir Java con Javascript. Los programas y aplicaciones son programadas en Java, mientras que Javascript es utilizado sobre todo en entornos webs (códigos en páginas HTML para incrustar aplicaciones interactivas). La versión más actual de Java es la JSE 7.0 (Dolphin) del 28 de Julio de 2011 y la próxima versión saldrá a finales del año 2012.

4.1 Ficheros y códigos

Al programar se crean ficheros *.java que contienen el código fuente. A diferencia con otros lenguajes de programación, para la ejecución del programa, no se crea directamente código en lenguaje máquina sino Bytecode que es interpretado en un entorno específico de Java llamado Java Runtime Environment (JRE) que cuenta con la Java Virtual Machine (Máquina Virtual de Java) encargada de ejecutar los programas. Primero el código fuente es convertido a Bytecode para ser interpretado. Cuando haga falta se compila al instante mediante un compilador llamado “just in time compiler”. En la programación en Java se puede hablar pues de la existencia de dos compiladores, el Bytecode-Compiler y el Native Compiler (crea código máquina). Los ficheros de Bytecode son de tipo *.class y no son directamente ejecutables. La ventaja del Bytecode y la máquina virtual es que los códigos programados pueden ser usados independientemente de la arquitectura y sistema operativo. Para cada sistema operativo habrá un entorno determinado que interpreta el mismo código. Sun/Oracle Corporation crearon entornos para Windows, Linux y Solaris. Distribuidores de otros sistemas operativos ofrecen entornos propios.

En los inicios la ejecución de programas creados en Java fue más lenta comparada con otros lenguajes de programación orientada a objetos como el C++. Con la optimización dinámica de la máquina virtual de Java, este lenguaje se convirtió en uno de los más eficientes y se acerca a las velocidades de los programas programados en C++ y C#.

4.2 Entornos y herramientas

El JDK (Java Development Kit) es un software con herramientas de desarrollo para programar programas en Java. Este JDK es distribuido bajo la licencia GNU (General Public Licence/licencia para software libre).

Para Java existen diferentes entornos de trabajo. Algunos de ellos son de libre distribución (Open Source) mientras que otros son de licencia. Entre los entornos libres destacan Eclipse de Eclipse Foundation y NetBeans de Sun. Para la programación de éste proyecto se utilizó el entorno de programación de Eclipse.

Para la programación de aplicaciones para móviles Android hace falta además la instalación de un plugin llamado Android Development Tools (ADT). Este plugin permite una configuración rápida de proyectos, la creación fácil del User Interfaces (interfaces de usuarios), depuración mediante herramientas del Android SDK (Android Software Development Kit), exportación de ficheros de instalación, etc. En el siguiente punto (5. Entorno de programación) se describirá detalladamente cómo instalar y usar el programa Eclipse para crear aplicaciones Android.

Usuarios de ordenadores con sistema operativo Mac OS de Apple pueden crear programas usando el entorno de programación Xcode que soporta varios lenguajes de programación. Xcode está optimizado para Java pero permite también programar aplicaciones para teléfonos iPhone en lenguaje Objective C.

Lenguaje de programación	Código fuente (.java)
JDK	Herramientas de desarrollo, compiladores de Java, etc.
	Java Bytecode (.class, .jar)
JRE	API
	Máquina Virtual Java (JVM) con Just in time Compiler
Sistemas Operativos	Windows, Linux, Solaris, Mac OS X, ...

Imagen 4.2 Tabla resumen de programación en Java

4.3 Clases, objetos y funciones

Al ser Java, igual que C++, un lenguaje orientado a objetos, se caracteriza por el uso de clases y objetos. En el código fuente de este proyecto se hace uso de ellos por lo que en este apartado se introducen ambos conceptos. Una clase es simplemente un concepto. Un ejemplo típico en la programación para describir una clase es el de "Persona". Una persona se caracteriza por sus datos internos que podrían ser su nombre, edad, sexo, etc. y además tiene unos métodos (acciones que puede realizar) como hablar, andar, escribir, etc.

Para usar una clase se tiene que instanciar. Esto equivale a crear un objeto. De la clase Persona, algunos objetos podrían ser José, María, etc. (que son personas). La clase Persona se podría escribir en código Java de la siguiente forma:

```
public class Persona{
    private String nombre;
    private int edad;
```

```
        private int sexo;
    }
```

La palabra *class* es una palabra reservada (no se puede usar para otros fines que para los que se ha diseñado). La palabra *class* indica que a continuación sigue una declaración de una clase. La palabra *public* justo delante de “class Persona” indica que esta clase puede ser vista por todas las demás clases del paquete. “Persona” es el nombre de la clase y del fichero Java (Persona.java). En el interior de la clase persona, se encuentran en este caso tres variables privadas de tipo *String* e *Int*. Por defecto y sin especificar nada, todas las variables son de tipo privado con lo que no pueden ser accedidas por otras clases. De lo contrario se podría declarar las variables como *public* (publicas). Normalmente se suelen declarar las variables como privadas y las funciones como públicas. Nombre, edad y sexo son los nombres de las variables. Al igual que en funciones matemáticas se usan las variables “x”, “y”, “z”, etc. se puede dar a las variables el nombre que el programador ve conveniente. El nombre de la variable debe indicar directamente a que se refiere. En este ejemplo se ve claramente que la variable “edad” contendrá la edad de una persona mientras que la variable “nombre” contendrá su nombre. Para guardar un nombre (que es una cadena de caracteres) dentro de una variable, ésta tiene que ser de tipo *String*. Números enteros pueden ser guardados en variables tipo *Int* y números con coma flotante en variables de tipo *double* o *float*.

La descripción de “andar” estaría determinada en la función:

```
public void andar(){
    //acción de andar
}
```

A esta función se accedería con el código: andar();

Un ejemplo fácil de para describir una función, es el caso de una función matemática. Como ejemplo se crea una función que suma dos números recibidos.

Al comienzo disponemos de dos valores $x=10$ e $y=20$ se quiere realizar su suma dentro de la función “calcula” y que ésta devuelva el valor de dicha suma en la variable “resultado”. Todas las variables fueron declaradas anteriormente como variables tipo *Int* al tratarse de números enteros.

El código `resultado=calcula(x,y);` llamará a la función “calcula” y le transmitirá los valores de x e y (10 y 20 respectivamente). Una vez que la función haya realizado los cálculos necesarios (la suma) devolverá el resultado con lo que se guardará en la variable “resultado”. Mediante `calcula(x,y);` se llama pues a la siguiente función:

```
public int calcula(int a , int b){
    return a+b;
}
```

Como indicado anteriormente las funciones suelen declararse como públicas. Después de la palabra reservada *public* sigue el tipo de la variable que devuelve la función. En este caso es de tipo *Int* ya que el valor devuelto será la suma de dos enteros. Este valor será el que toma la variable “resultado” (de `resultado=calcula(x,y);`) que fue declarado como *Int*. A

continuación sigue el nombre de la función (calcula) y entre paréntesis los dos valores recibidos por "calcula(x,y)". En la función "a" tomara el valor de "x" y "b" el de "y". Dentro de la función se realiza el cálculo deseado que en este caso es la suma de a y b. Finalmente su resultado es devuelto (return) con lo que "resultado" será igual a "a+b" o "x+y".

El mismo resultado lo tendrían las siguientes líneas de código dentro de la función:

```
int c;  
c=a+b;  
return c;
```

De esta forma queda más intuitivo. Se declara una nueva variable "c" que obtendrá el valor de la suma c=a+b. En los lenguajes de programación la variable que toma un valor se escribe a la izquierda. Después de que "c" obtenga el valor de la suma, se devuelve el resultado guardado en "c" escribiendo "return c".

4.4 Relaciones y diferencias con C++

El lenguaje de programación Java es bastante parecido al lenguaje C++ por lo que programadores del segundo lenguaje no deben encontrarse con demasiados problemas durante la programación. Ambos lenguajes están orientados a objetos pero a diferencia de C++, Java funciona sin punteros (que pueden dar a error fácilmente) y no tiene arquitecturas complejas que permiten la reiterada herencia. Existe pues sólo una herencia simple. Clases sólo pueden tener una superclase. Java se ocupa internamente de los procesos relacionados con memoria y con la liberación automática de memoria ya no necesitada. Esto hace que algunos aspectos de programación sean más fáciles.

En Java tampoco es posible darle a operadores (operadores aritméticos como "+" y "-" u operadores lógicos como "&&" o "||") nuevos significados o funciones lo que ayuda a que los códigos sean más legibles.

En cuanto al uso de ambos lenguajes, tras la tendencia de los últimos años se puede decir que Java es el lenguaje de programación más usado seguido de C y C++. Estos datos se reflejan también en la demanda de trabajadores con estos conocimientos.

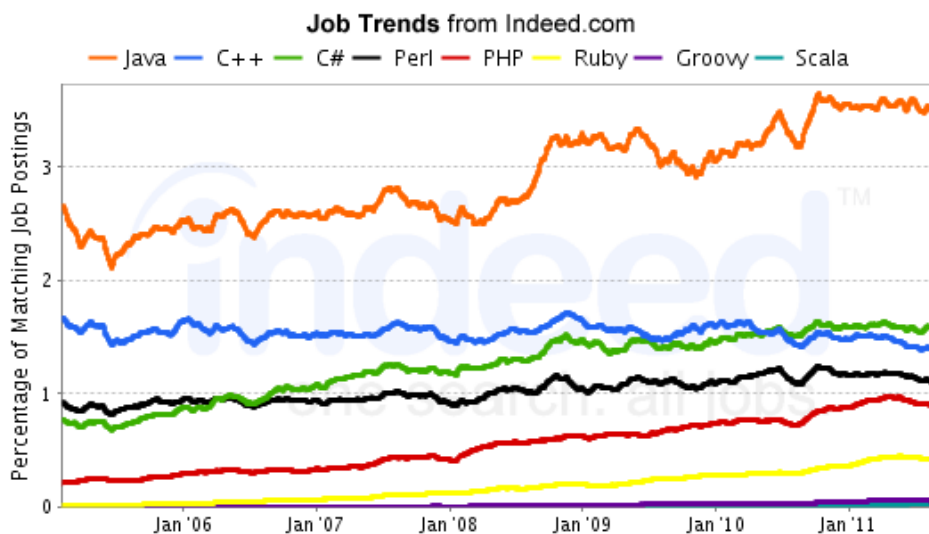


Imagen 4.3 Demanda de lenguajes de programación

5. Activity

Antes de comenzar a explicar códigos fuentes y el funcionamiento del entorno de programación es necesario explicar el significado y funcionamiento de una *activity* ya que es un concepto que aparecerá muchas veces a lo largo de este proyecto.

Cada pantalla de una aplicación de Android es una *activity*. Se trata de uno de los componentes más usados en la programación para este sistema. Una *activity* representa pues una pantalla o interface que puede recibir datos del usuario. Al realizar un cambio de una pantalla a otra se cierra o pausa la *activity* actual y se arranca la *activity* nueva. Durante este cambio es posible pasar datos de la primera a la segunda *activity*. Se puede programar por ejemplo una pantalla con un formulario en el que el usuario introduce unos datos y posteriormente cambiar a otra *activity* que se encarga de guardar esos datos en algún lugar determinado o realizar cualquier otra operación con estos datos.

Cada clase de *activity* se guarda en un fichero independiente. Al tratarse de código Java, se guarda en ficheros tipo *.Java.

Al abrirse *activities* nuevas, las viejas se guardan temporalmente en una pila llamada *History-Stack* o *Activity-Stack*. Éstas permiten que el usuario pueda volver a *activities* usadas anteriormente. Se puede decir que las diferentes pantallas son apiladas. Este proceso se repite hasta que la memoria del dispositivo dedicada a esta operación está llena. Una vez llegado a este punto, el sistema comienza automáticamente a cerrar las *activities* viejas. Para no perder los datos (por ejemplo de formularios o correos escritos), se guardan éstos en el disco duro del teléfono. Si el usuario vuelve a acceder a una *activity* abierta anteriormente que aún se encuentra en el *Activity-Stack*, ésta se abrirá más rápido y sin gastar demasiados recursos. En el caso de que la *activity* fue cerrada por el sistema, la aplicación arrancará desde el principio y si se guardaron datos antes de cerrarla, los puede recuperar con lo que el usuario no se da cuenta de los procesos escondidos que realiza el teléfono.

Cuando el usuario abandona una aplicación, realmente no la está cerrando (como en el caso de programas de ordenadores). La aplicación queda activa. Desde cualquier *activity* se puede cambiar a una *activity* anterior, manteniendo pulsada la tecla (hard-key) "home" (muchas veces tiene como símbolo una casa) durante más de dos segundos. Al realizar esta acción se abre una ventana nueva con los iconos de todas las *activities* activas. Pulsando sobre uno de estos iconos se puede cambiar a la *activity* deseada.

Las *activities* en Android se caracterizan también por tener un determinado ciclo de vida (lifecycle), relacionado con el *Activity-Stack*, que se resume en la imagen 5.1. Antes hace falta comentar que una *activity* se puede encontrar en cuatro estados diferentes:

1. Una *activity* está activa (*running*) cuando es mostrada en la pantalla del smartphone. En este caso la *activity* se encuentra en la capa superior del Stack.
2. Una *activity* está pausada (*paused*) cuando es visible sólo en parte (por ejemplo en el caso

de abrirse una ventana emergente semi-transparente). Esta activity aún está activa, pero podría ser cerrada por el sistema en caso de necesidad de memoria. 3. Cuando una *activity* ya no es visible porque la “tapa” otra activity en la pantalla, se dice que está *stopped*. La activity sigue conteniendo la información actual, pero puede ser cerrada cuando le haga falta memoria al sistema.

4. Cuando una *activity* se encuentra en uno de los dos casos anteriores y el sistema necesita más memoria se cierra o mata (*kill*) el proceso.

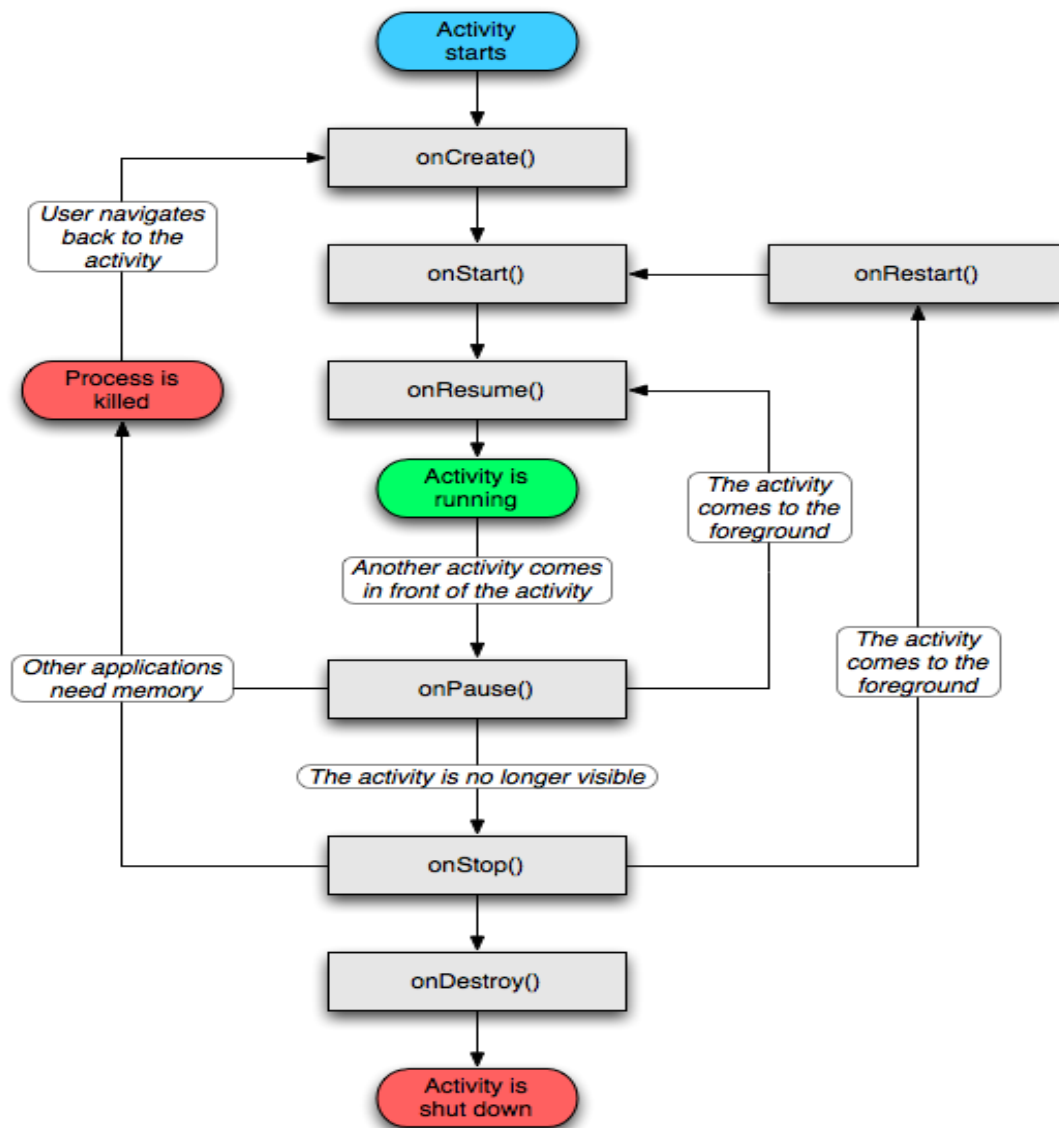


Imagen 5.1 ciclo de vida de una activity

Para cambiar dentro de una aplicación de una *activity* a otra como comentado anteriormente, se usa un *intent*. Un *intent* describe lo que una activity tiene que hacer. Un intent es además capaz de transmitir datos. Para cambiar a otra *activity* se usa el método *startActivity()*.

6. Entorno de programación

Para realizar este proyecto se usó el entorno de programación de libre distribución Eclipse en la versión 3.6.2 (Helios) de Eclipse Foundation. Se trata del entorno más usado para la programación de aplicaciones de Android. Antes de comenzar a programar es necesario configurar el ordenador y descargar e instalar diferentes paquetes y plugins como se explica en el apartado de instalación. Todas las descargas usadas en este proyecto son gratuitas y se encuentran en las páginas webs oficiales de Oracle, Eclipse y Android.

6.1 Instalación

El proceso de instalación es relativamente largo. El primer paso consiste en la descarga de diferentes programas y plugins. Es recomendable comenzar con la descarga e instalación de la última versión del JDK (java development kit) de Oracle.

A continuación es necesaria la descarga del entorno de programación Eclipse. Para ello se recomienda la versión Helios (o una versión superior). Una vez instalado ya es posible desarrollar programas en Java. Para poder desarrollar aplicaciones de Android es necesario añadir varios paquetes adicionales.

El Android SDK (software development kit) está disponible para Windows, Mac OS X y Linux. Tras la instalación del SDK se instala el ADT Plugin (Android development tool) para Eclipse. El ADT contiene las herramientas para el desarrollo de aplicaciones. Estas herramientas pueden entre otras, ayudar a simplificar el diseño o la depuración de las aplicaciones creadas.

Como último paso queda descargar los componentes esenciales del SDK. Para ello se abre el programa Eclipse por primera vez y en el menú "Window" se selecciona la pestaña "Android SDK and AVD Manager" dónde se encuentra una lista de paquetes instalados, paquetes disponibles, ajustes y dispositivos virtuales. Para un correcto funcionamiento es necesario descargar o actualizar la lista completa. La descarga de todos los componentes dura bastante tiempo al descargar e instalarse muchos componentes. Tras la instalación, la ventana del SDK and AVD Manager tendrá el siguiente aspecto.

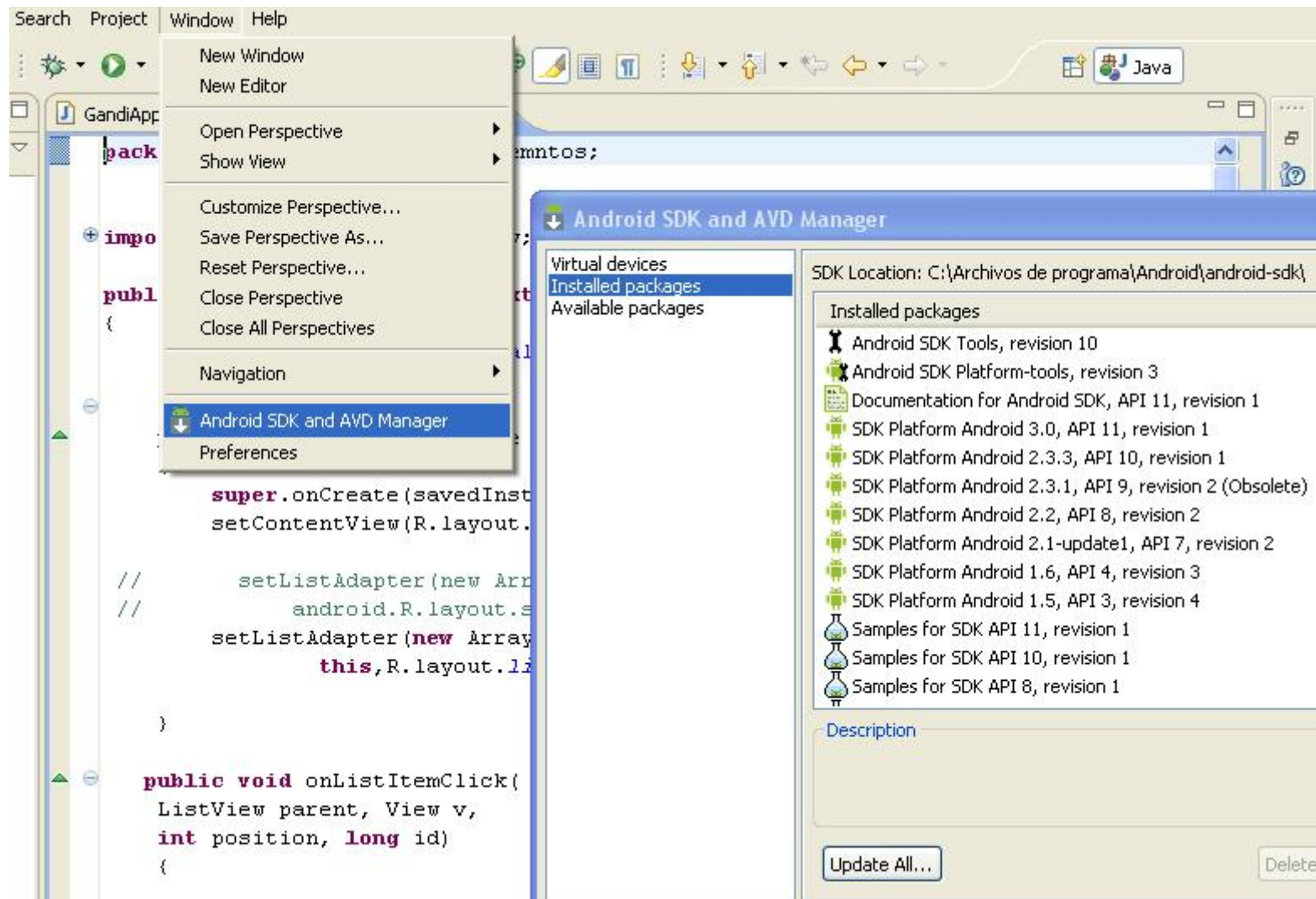


Imagen 6.1 Android SDK and AVD Manager

6.2 Configuración del entorno

Para crear una aplicación de Android es necesario crear un nuevo proyecto. Para ello se pincha en el menú sobre File, New, Project (no sobre Java Project). Seguido se abre una nueva ventana con la orden “select a wizard” dónde se busca la carpeta Android y se pincha sobre Android Project antes de pulsar en el botón Next. En la siguiente ventana, Eclipse le pide al usuario varios datos sobre la nueva aplicación. En primer lugar se escribe el nombre de la aplicación nueva. El siguiente dato a introducir es el Built Target. Esto es decir para que versión de Android se quiere programar. Esto depende mucho del tipo de aplicación que se quiere crear. La tecnología avanza muy rápido por lo que los nuevos móviles tendrán versiones de Android superiores a los móviles de hace uno o dos años. Las versiones nuevas por tanto ofrecen más posibilidades a la hora de crear aplicaciones. Un smartphone con Android 3 podrá ejecutar aplicaciones que se programaron con versiones (plataformas) inferiores sin problemas. Al contrario los móviles basados en Android 2 no podrán ejecutar una aplicación programada para una versión más actual. De allí es importante conocer el porcentaje de uso actual de cada versión de Android y ser consciente de que según la versión utilizada habrá un porcentaje de usuarios que no podrán usar la aplicación.

Google publica cada determinado tiempo estadísticas sobre las versiones de los teléfonos móviles que acceden al Android Market para ayudar al programador a decidirse por una versión. Estos porcentajes se ajustan bastante a la realidad pero no tiene en cuenta aquellos teléfonos que no se conectaron al market durante ese período de tiempo. Los datos de la segunda quincena de Agosto del año 2011 se pueden analizar en la siguiente gráfica circular y en su respectiva tabla.

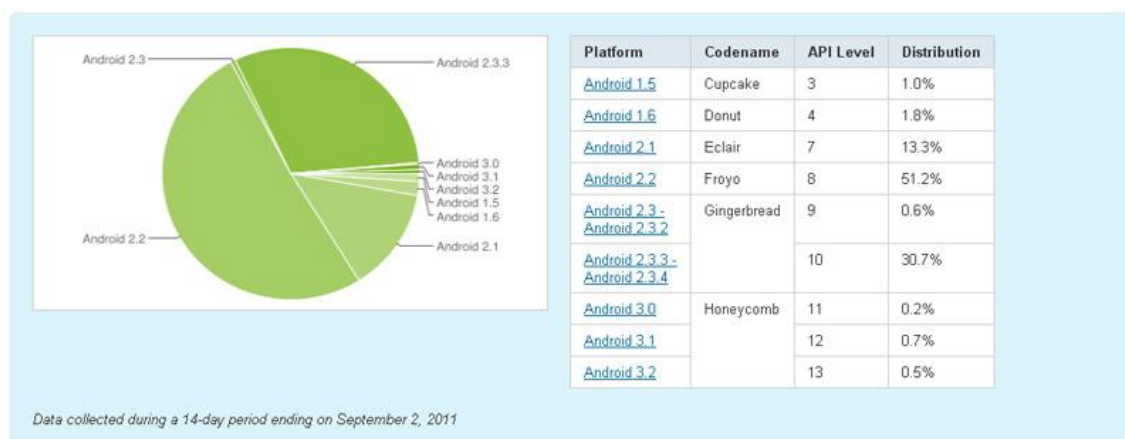


Imagen 6.2 Relación del uso de plataformas Android

Según estos datos la mayoría de teléfonos utilizan la versión 2.2 (55,9%) seguido de la versión 2.3 (24,3%) y la 2.1 (15,2%). Las versiones Anteriores (1.5 y 1.6) se pueden despreciar al representar tan sólo un 3,3%. Aplicaciones nuevas deberían estar programadas para la plataforma 2.1 o 2.2 (API* Level 7 u 8 respectivamente). A medida que se actualizan los smartphones viejos o el número de venta de teléfonos nuevos aumenta, habrá que cambiar de plataforma. Para ello es interesante vigilar estas estadísticas [12]. [12] ANDROID DEVELOPER

Debido a estadísticas anteriores con porcentajes parecidos, se programó la aplicación de este proyecto inicialmente para la plataforma 2.1 – update1 (de Android Open Source Project). Tras incluir en la aplicación la API de Google Maps y modificar el código fue necesario cambiar el “Build Target” (visto en el punto anterior) a “Google APIs” para garantizar el correcto funcionamiento de la aplicación. La plataforma sigue siendo la 2.1 –update1 con API Level 7. Pueden existir pues “paquetes” diferentes dentro de una misma versión. Esto demuestra que se tienen que tener en cuenta diferentes variables a la hora de decidirse por una plataforma o versión.

Después de elegir la versión o plataforma se rellenan los siguientes campos de texto. El “nombre de aplicación” y el campo “create activity” pide simplemente un nombre (pueden ser iguales). El campo importante en este formulario es “package name” y debe contener al mínimo dos palabras separadas por un punto. Como norma general se escribe una dirección de dominio y luego el nombre de la aplicación. Serían nombres válidos por tanto com.aplicae.gandia o com.gandiaturistica.gandia dónde en el primer caso, “com.aplicae” (de www.aplicae.com) sería el dominio y “gandia” sería el nombre de la aplicación. Esta forma de nombrar los paquetes se hace para que cada aplicación provenga de un paquete único y no se produzcan conflictos de nombres.

El último campo de este formulario es el número de versión del SDK. En este proyecto se usó la versión 7.

Una vez finalizada esta configuración rápida se abre el entorno de trabajo completo y está listo para empezar a programar.

*API Level:

El API Level (Application Program Interface) está estrechamente relacionado con la versión de Android. El nivel de API es un número entero que identifica el marco de la API (framework API). Este framework API puede ser usado por las aplicaciones para interactuar con el sistema Android. Una API fija cómo hay que invocar determinadas funciones con el fin de estandarizar la programación.

El API framework engloba diferentes aspectos. El significado concreto y el funcionamiento se explica en los capítulos dónde son usados. Ésta es sólo una muy breve explicación del contenido general del framework API. En primer lugar contiene varios paquetes y clases básicas. Un juego de elementos y atributos para declarar un archivo de tipo XML llamado “manifest” y otro para declarar y acceder a diferentes recursos. Como cuarto “paquete” contiene una colección de Intents. Por último incluye diferentes permisos que las aplicaciones o el sistema operativo pueden pedir.

El número entero del API Level es usado por el Sistema Operativo para indicar cuál es el framework máximo que soporta mientras que las aplicaciones lo usan para indicar los requisitos mínimos que necesita el sistema para que la aplicación pueda funcionar. Versiones incompatibles no son instaladas por el sistema.

6.3 Partes de una aplicación

En la parte derecha del entorno de programación se encuentra el explorador de paquetes (package explorer) con algunas carpetas y ficheros básicos necesarios para la creación de una aplicación Android en Java. A medida que se programa se tienen que añadir más carpetas y ficheros adicionales. En este apartado se describe brevemente el contenido de éstas carpetas para comprender su funcionamiento y uso y poder localizar los diferentes elementos dentro del entorno de programación. Los diferentes ficheros y sus códigos fuentes serán explicados con más detalles en capítulos posteriores con ayuda de ejemplos reales (de la aplicación de Gandía).

La primera carpeta se llama “src” y contiene todos los ficheros con los códigos fuentes programados (ficheros tipo *.java). Al comenzar a programar sólo contiene un fichero Java vacío al que se le añade el código fuente. A medida que se programan más “activities” (pantallas o ventanas de la aplicación) se añaden más ficheros. Al final cada activity tendrá como mínimo un fichero *.java.

La segunda carpeta se llama “gen” y contiene un fichero llamado R.java. Hay que tener mucho cuidado con este fichero. Durante la programación este fichero guarda información sobre el programa. Así por ejemplo guarda información sobre variables declaradas y sus valores en hexadecimal. El entorno de programación se encarga de actualizar el fichero automáticamente.

El programador no debe manipularlo ya que un dato erróneo en este fichero llevará a que la aplicación no se pueda ejecutar. Encontrar un error en un fichero “R” y corregirlo es muy difícil.

La siguiente carpeta importante en este nivel es la carpeta llamada “res” que a su vez contiene varias carpetas y ficheros diferentes. Entre estas carpetas encontramos tres con el nombre “drawable”. En ellas se guardan todas las imágenes que aparecerán en la aplicación y el icono del “ejecutable”. En la carpeta layout se encontrarán todos los ficheros tipo *.xml donde se define el aspecto de las diferentes pantallas de la aplicación. Se puede o bien escribir el código directamente en este fichero o utilizar el “Graphical Layout” que permite seleccionar elementos como botones, listas, imágenes. etc. de una lista. La segunda opción puede ser más fácil, pero se pierde algo de control. Además hay que cambiar y fijar posteriormente algunos ajustes. En la última carpeta (“values”) se encuentra inicialmente un fichero de tipo *.xml llamado “string.xml” en la que se guardan como indica su nombre todos los strings (cadenas de texto) usados en la aplicación.

Como último fichero importante en esta carpeta se encuentra el fichero AndroidManifest.xml que es manipulado también por el programador. En este fichero se tienen que listar todas las activities de la aplicación. Si una activity no se encuentra en este fichero y se intenta abrirla desde la aplicación, dará un error y la aplicación se cerrará. Además se tienen que incluir todos los permisos necesarios para que la aplicación funcione. Sin los permisos necesarios, la aplicación no puede recibir datos del receptor GPS, ni establecer conexiones a Internet o realizar llamadas telefónicas.

6.4 El entorno

En la imagen 6.3 se muestra el entorno de programación Eclipse con sus pestañas y ventanas (las más importantes son las enmarcadas y numeradas).

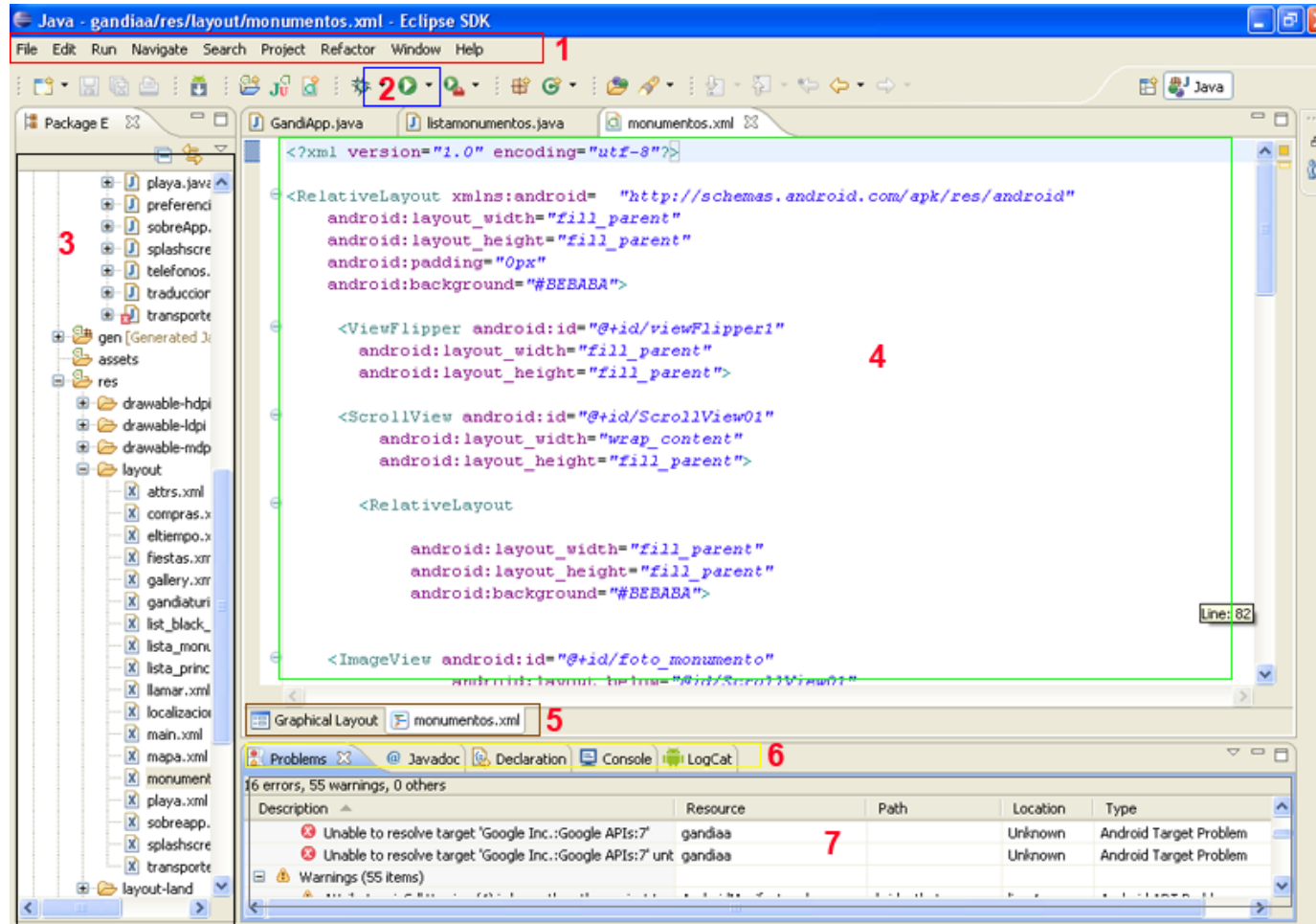


Imagen 6.3 Entorno de programación

En la parte superior del programa Eclipse (enmarcado de color rojo y con el número uno) se encuentra la barra de menú dónde se encuentran las pestañas para configurar la aplicación (crear proyectos nuevos, actualizar paquetes del SDK y AVD y crear dispositivos virtuales). En la barra de menú se encuentran todos los accesos a las diferentes ventanas del programa. La mayoría de ellas ofrecen opciones muy específicas que habitualmente no son usadas.

En la parte izquierda (enmarcado de color negro y con el número 3) se encuentra la ventana “Package Explorer”. Este explorador de paquetes muestra todas las aplicaciones creadas hasta el momento. Cada paquete consta de diferentes carpetas y archivos que incluyen el código fuente de las diferentes pantallas de la aplicación, los códigos de diseño de las actividades, las listas con las cadenas de textos (*strings*) utilizados, las imagenes usadas, el fichero R, el fichero con los permisos, etc.

Los ficheros *.java y *.xml con los diferentes códigos se manipulan en la ventana central del entorno de programación (enmarcado en color verde y con el número 4). En la parte superior se encuentran las pestañas de todos los ficheros abiertos en ese momento. En la imagen el fichero sobre el que se está trabajando (pestaña activa) es el fichero de diseño de la *activity* “monumentos”, de allí su nombre “monumentos.xml”. En la pestaña anterior se encuentra el código java (monumentos.java) de esa misma *activity*.

En el caso de los archivos de diseño (*.xml), el entorno Eclipse ofrece dos opciones diferentes de diseño. En este caso se seleccionó el diseño por código, pero como se observa en el punto 5 (enmarcado en marrón) se puede escoger también la opción “Graphical Layout” que permite escoger diferentes botones, listados, textos, barras, etc. de una lista. Esta forma de realizar los diseños es más intuitivo y se parece a la forma de diseñar aplicaciones para el sistema operativo de Apple. El Graphical Layout ayuda al programador en los casos de utilizar componentes que no se utilizan a menudo (y cuyo código fuente no se conoce de memoria). De todos modos, después de realizar el diseño de esta forma es necesario volver a la vista de diseño por código para tomar todos los ajustes necesarios.



Imagen 6.4 Graphical Layout

Una vez esté terminada la parte de programación de la aplicación (o de una parte determinada como una *activity*) se puede pasar a la simulación y / o búsqueda y corrección de posibles errores. Para ello se puede pinchar sobre el icono redondo de color verde con un triángulo blanco en el centro (enmarcado de color azul y con el número 2). El programa Eclipse comenzará a analizar el código fuente en busca de errores y alertas o avisos. En el caso de no encontrarse errores se ejecutará el código y la aplicación arrancará en el

emulador (un dispositivo virtual que se explica en el punto 5.5). En el caso de que se encuentren errores, éstos se mostrarán en una pequeña ventana en la parte baja (enmarcada en gris y con el número 7) de Eclipse siempre que en el punto 6 (amarillo) esté seleccionada la pestaña “Problems”.

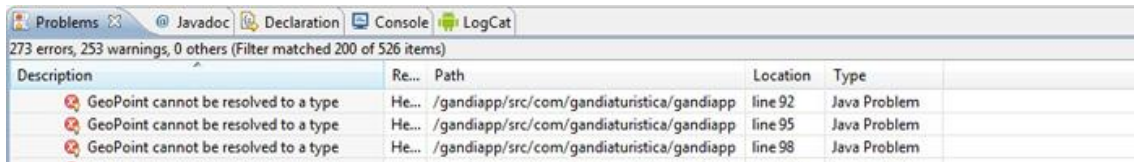


Imagen 6.5 Problemas en el código

La información allí mostrada se diferencia entre “errors” que tienen que ser solucionados obligatoriamente para que la aplicación funcione y “warnings” que se deben corregir recomendadamente ya que pueden dar a fallos (en el caso normal una aplicación también funciona aunque contenga warnings). En ambos casos (errors y warnings) se indica, entre otros, la línea de código en la que se encuentra el fallo y una breve descripción del fallo como se muestra en la imagen anterior. En este caso se observa que en las líneas 92, 95 y 98 hay un problema en el código Java de un fichero de la aplicación de Gandía. Como descripción se indica: “GeoPoint cannot be resolved to a type”. Los *GeoPoints* contienen las coordenadas GPS de determinados lugares. En el caso de esta aplicación guardan las coordenadas de los lugares de interés, hoteles, etc. Como en muchos casos, el verdadero error no se encuentra en la misma línea cómo la indicada por el entorno de programación sino unas líneas antes. La información de que el *GeoPoint* no pudo ser resuelto a un tipo indica que posiblemente no se haya declarado “GeoPoint” de la forma correcta. Al cambiar la declaración al principio del programa, todos los errores de este tipo desaparecerán.

Una vez el código fuente ya no contenga errores, se prepara éste para ejecutarlo en el emulador (un dispositivo/smartphone virtual).



Imagen 6.6 Ejecución de una aplicación

Este proceso tarda varios segundos y mientras tanto Eclipse muestra su estado en la misma ventana dónde indicaba los errores. Para ver la información hace falta seleccionar la pestaña “Console”. En cuanto aparezca en esta ventana el mensaje “Success!” y “Startig activity [...]” se puede ver la ejecución de la aplicación en el dispositivo virtual.

6.5 El emulador

El emulador, que muestra las aplicaciones en un dispositivo virtual, se abre en una nueva ventana. Se trata de un programa que simula el funcionamiento de un smartphone y soporta casi todas las funciones de un smartphone.

El emulador arranca cuando Eclipse reciba la orden de “run” (arrancar la aplicación). La carga del emulador es lenta y dependiendo del ordenador en el que se ejecuta puede tardar varios minutos. La interfaz de este dispositivo virtual consta de una pantalla “táctil” en la parte izquierda y de un teclado “qwerty” y botones de menús a la derecha. Inicialmente la pantalla estará negra y sólo aparecerá en ella el nombre “Android”. Una vez arrancado el emulador completamente se enchufará la pantalla y tras desbloquearla (como en una pantalla táctil real) se verá la ventana principal del sistema operativo Android. A partir de ese momento se puede simular casi el comportamiento completo de un smartphone. Pocos segundos después de arrancar, se ejecutará automáticamente la aplicación que se estuvo programando.



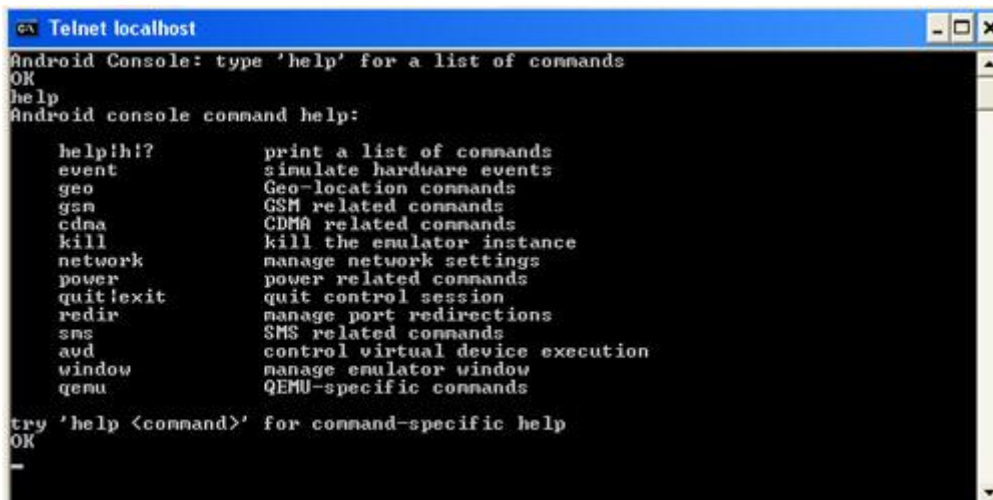
Imagen 6.7 Emulador Android del entorno Eclipse

6.5.1 Posibilidades del emulador

El emulador permite tomar un sinnúmero de ajustes sobre el mismo dispositivo virtual como si se tratase de un dispositivo real. Ejecutando el menú principal desde la ventana inicial (mostrada en la imagen) se puede cambiar la imagen de fondo del dispositivo, realizar una búsqueda en Internet (siempre que el ordenador en el que se ejecuta el emulador está conectado a Internet), leer las notificaciones del sistema operativo o de aplicaciones y tomar ajustes (settings). Entre los ajustes se encuentran ajustes de redes, llamadas, sonidos, pantalla, seguridad, posición, etc. El emulador permite también ver las aplicaciones instaladas y las que se encuentran en ejecución.

El emulador permite además la captura de imágenes. En este caso se pincha sobre el botón de la cámara de fotos de los botones de menú. En la mayoría de los casos se abre una ventana con un fondo de cuadrados blancos y negros y otro que se mueve constantemente por la pantalla. Existen códigos especiales para configurar el emulador de tal forma que en esta cámara virtual se capturen las imágenes de la webcam del ordenador. Al tratarse de una función bastante específica, no está configurada por defecto.

Un problema parecido se produce al simular posiciones de GPS con el simulador. En un dispositivo real, al conectar el receptor GPS, el teléfono puede calcular su posición geográfica. En el ordenador en cambio esto no es posible. Para simular posiciones GPS en el emulador hace falta conectarse al emulador a través del programa “Telnet” mediante el comando “telnet localhost 5554”. Para ello en Windows se abre “Símbolos del sistema” en “todos los programas/accesorios”. Una vez añadida la línea de código anterior, se conecta a la consola. La imagen siguiente muestra una captura de pantalla de la consola tras escribir el comando “help” para conocer todos los comandos posibles y sus descripciones.



```
GA Telnet localhost
Android Console: type 'help' for a list of commands
OK
help
Android console command help:

  help!h?      print a list of commands
  event        simulate hardware events
  geo          Geo-location commands
  gsm          GSM related commands
  cdma         CDMa related commands
  kill         kill the emulator instance
  network      manage network settings
  power        power related commands
  quit!exit    quit control session
  redir        manage port redirections
  sms          SMS related commands
  avd          control virtual device execution
  window      manage emulator window
  genu        QEMU-specific commands

try 'help <command>' for command-specific help
OK
-
```

Imagen 6.8 La consola

Mediante el comando “geo fix” seguido del valor de longitud y latitud se podrá establecer la posición “actual” del dispositivo.

Entre las otras opciones que ofrece el emulador se encuentran también las siguientes:

- El emulador permite navegar por páginas webs (igual que en el caso de las búsquedas comentadas antes, el ordenador necesita conexión de Internet para ello).
- Se puede guardar información en ficheros y en bases de datos y recuperarlas posteriormente.
- Mediante la combinación de teclas “Control” y “F12” (del teclado del ordenador) es posible girar el dispositivo virtual en 90 grados para realizar pruebas básicas del sensor de aceleración (inclinación) o para cambiar entre la vista normal o la vista “landscape” (pantalla en posición horizontal).

El desarrollador de aplicaciones puede usar y probar un gran número de funciones diferentes sin necesidad de disponer de un teléfono real (para realizar las primeras versiones de una aplicación). Antes de lanzar una aplicación a un market sí que es necesario testear la aplicación en diferentes dispositivos reales para asegurarse de su buen funcionamiento.

6.5.2 Limitaciones del emulador

Pese a la gran variedad que ofrece el emulador de Android de Eclipse, no todas las funciones son soportadas. Esto sucede en funciones específicas de un teléfono que por su naturaleza no se pueden realizar con un ordenador normal.

En el apartado 6.5.1 se vio que el emulador es capaz de “registrar” una inclinación de 90 grados (cuando el usuario se lo indica mediante el teclado). La simulación del comportamiento del sensor de aceleración sobre sus tres ejes no es posible con el emulador normal de Eclipse. Debido a eso no se pueden medir fuerzas G (gravedad y aceleración) o medir la inclinación exacta en grados. El mismo problema se plantea al simular el comportamiento de la brújula.

Para solucionar este problema hace falta recurrir a otros programas o códigos Java especiales como el llamado “SensorSimulator” [13] ofrecido por “Openintents” que ayudan a solucionar esa deficiencia de Eclipse.

Al simular, el emulador no ofrece valores reales sobre su estado de memoria, batería o consumo de cuota 3G. En el caso de la información referente al uso del disco duro interno se muestra información del disco duro del mismo ordenador. El estado de carga de la batería en cambio se encuentra siempre en un 50% constante sin variación.

Al no poseer el ordenador de un tarjeta telefónica, no es posible usar el emulador para realizar llamadas a un dispositivo real. La única opción es la de simular una llamada con dos instancias de un emulador. Hace falta configurar ambos de manera correcta para poder realizar una conexión. Es un proceso que puede dar fallos. Dependiendo del uso que se le quiera dar, se puede volver a usar la consola de “Símbolos del sistema” de Windows para realizar una conexión mediante Telnet usando en este caso la línea de comandos “gsm calls” seguido de un “número de teléfono”. Como estas funciones no vienen configuradas por defecto en el entorno Eclipse, es complicado llevarlo a funcionar correctamente.

El emulador ayuda mucho a la hora de realizar pruebas rápidas del código. Si el comportamiento no es el deseado se puede cambiar el código fuente y ejecutarlo de nuevo para ver los cambios. A pesar de ser una herramienta potente durante el desarrollo, es absolutamente indispensable testear las aplicaciones en dispositivos reales. En éstos pueden surgir nuevos problemas que además pueden variar de un teléfono a otro. Uno de los problemas para los programadores de Android no es sólo la variedad de versiones del sistema operativo, sino los diferentes hardwares. Esto puede comenzar en los diferentes tamaños y resoluciones de las pantallas, pasando por la ausencia de algunos botones (hardkeys) en algunos dispositivos y acabar en la ausencia de determinados sensores electrónicos (algunos teléfonos, no todos, son capaces de medir por ejemplo la presión atmosférica).

[13] GOOGLE CODE

7. La aplicación de Gandía

La aplicación de Gandía programada en este proyecto final de carrera es una aplicación dinámica e informativa. Es dinámica porque accede a Internet para actualizar algunos datos determinados. Así por ejemplo muestra siempre la situación atmosférica actual y el pronóstico de tiempo para los siguientes días. Al acceder a una API de noticias, informa a los usuarios siempre de las noticias locales relevantes. Con esto la aplicación se diferencia de varias aplicaciones ofrecidas por otras ciudades que se limitan a información estática (la información mostrada en éstas aplicaciones no es actualizada).

Esta aplicación es informativa porque ofrece por un lado información actual (comentada en el párrafo anterior) a sus usuarios y por otro lado información turística a los visitantes y turistas que visitan el destino turístico de Gandía.

La aplicación de Gandía (creada por la web de gandiaturisticacom.com y la empresa Aplicae a través de un convenio de prácticas en empresas con la Universidad Politécnica de Valencia) es la primera aplicación turística sobre Gandía para móviles.

Gandía es un municipio de la Comunidad Valenciana con aproximadamente 110.000 habitantes. Durante los meses de verano triplica su población debido a los aproximadamente 200.000 turistas. El fin de esta aplicación es el de ofrecer información a un determinado porcentaje de estas 300.000 personas y además a potenciales visitantes que no hayan venido a la ciudad. Para ello se usó, entre otras, la información turística ofrecida en la web www.gandiaturisticacom.com. Finalmente se publicó una versión gratuita de la aplicación.

Este proyecto final de carrera intenta relacionar el turismo con las nuevas tecnologías electrónicas del sector de las telecomunicaciones creando una aplicación nueva para smartphones. En las próximas páginas de este proyecto se detallan los sensores, APIs y códigos utilizados.

La aplicación consta de aproximadamente 30 actividades con más de 45 pantallas. Esta es una explicación breve del contenido de la aplicación de Gandía. En el siguiente capítulo se explica detalladamente cómo se cumplieron los requisitos establecidos para este proyecto mediante una explicación de los sensores y códigos usados en cada actividad. Tras ejecutar la aplicación se muestra una imagen de la playa de Gandía durante unos instantes antes de abrirse la actividad principal con la lista completa de las diferentes categorías de la aplicación. Al seleccionar una de estas categorías pulsando sobre la pantalla táctil, la aplicación muestra la actividad correspondiente. Las categorías en este nivel son: mapa, el tiempo, noticias, transportes, hoteles, playa, lugares de interés, fiestas, compras, localización, galería de fotos, teléfonos, página web e información sobre la App. En el siguiente punto se encuentra una descripción de cada una de las pantallas y en el punto 9 se explica el código fuente de algunas de ellas.

El objetivo principal del proyecto fue la programación y el diseño de una aplicación para móviles Android (programada en lenguaje de programación Java) para el sector turístico. Los objetivos secundarios fueron: acceder y adquirir información automatizada de determinadas páginas webs o widgets de Android, integración de una API (application programming

interface) de Google (comunicación entre dos softwares distintos), la utilización de los componentes hardware disponibles en el dispositivo móvil como el GPS, el acelerómetro, la brújula, la pantalla táctil, botones (hardkey), etc., creación de funciones para facilitar la posterior ampliación de contenido, comprobación del funcionamiento de la aplicación en dispositivos de diferentes fabricantes y versiones del sistema operativo y finalmente la publicación de una versión gratuita en el Android Market.

Los objetivos secundarios anteriores (menos los dos últimos) se detallan en el punto 9 (programación del código fuente).

8. Las ventanas de la aplicación

Como comentado en el punto 7, la aplicación de Gandía cuenta con más de 45 pantallas distribuidas en aproximadamente 30 actividades. En este punto se da una explicación de la información que recibe el usuario (tanto turista como habitante de Gandía) a través de la aplicación. La explicación del código fuente se encuentra en el punto 9. La jerarquía completa con los niveles y actividades de la aplicación se encuentra en el anexo. Esta descripción sigue el orden tal y como se le presenta la información al usuario en el smartphone.

Mapa:

El mapa usado en esta *activity* es el mapa de GoogleMaps. La ventana del mapa de Gandía dispone de un menú para acceder a una página de ajustes. En ese lugar el usuario puede activar el GPS de su dispositivo para que se muestre su posición actual sobre el mapa, cambiar entre la vista de mapa normal o la vista satélite y escoger la información que desea que se muestre sobre el mismo mapa. La información del mapa está pensada sobre todo como ayuda para turistas que no conocen el destino bien. Sobre el mapa se muestran los hoteles, lugares de interés y muchos más lugares importantes de Gandía. La *activity* fue programada de tal forma que al abandonarla, se desactive el GPS para ahorrar batería.



Imagen 8.1 Mapa

El Tiempo:

Para que la aplicación no sea sólo interesante para turistas, sino también para habitantes de la ciudad con teléfonos basados en el sistema operativo Android, se incluyó la *activity* del tiempo. La aplicación se conecta a través de Internet a un servidor que devuelve datos atmosféricos como la temperatura, humedad, dirección del viento, etc. La *activity* del tiempo muestra el estado actual y un pronóstico para los siguientes días. Al mostrar siempre el tiempo de forma actual de la misma ciudad, es mucho más preciso y cómodo el uso de la aplicación que ver el pronóstico en la televisión (sólo a determinadas horas y para zonas geográficas poco precisas) o en un periódico (información vieja comparada con la información actual en Internet).



Imagen 8.2 El Tiempo

Noticias:

Igual como en el caso anterior, se incluyó la *activity* con noticias regionales para que la aplicación no sea puramente turística y sea utilizada también por habitantes de la ciudad. La *activity* de noticias muestra una página web adaptada (en tamaño) a las pantallas de los teléfonos móviles. La página web a la que se conecta la aplicación de Gandía es una página de noticias de la empresa Google a la que se añadió un filtro para que sólo aparezcan noticias relacionadas con Gandía. Las noticias que se ofrecen son las ofrecidas por Google que las adquiere de diferentes portales informativos tanto locales (por ejemplo Safor Guia) como nacionales (por ejemplo El País).



Imagen 8.3 Noticias

Transportes:

En la *activity* de transportes se encuentra información básica (precios y horarios) de los diferentes medios de transportes de la ciudad (autobuses, trenes y taxis). De nuevo esta *activity* puede ser de interés para turistas y habitantes. Resulta especialmente interesante el horario de Renfe al que se accede por medio de un botón. Tras escoger el origen y el destino en un pequeño formulario, se le ofrece al usuario un horario personalizado con los siguientes trenes. De esta forma el usuario evita tener que buscar la información en mapas y folletos (que pueden contener información anticuada).



Imagen 8.4 Transportes

Hoteles:

Para obtener información de hoteles de Gandía la aplicación se conecta al apartado de hoteles de la web turística de Gandía www.gandiaturistica.com. El usuario puede navegar por esta web a través del smartphone e informarse sobre los hoteles de la ciudad. En la web se encuentra una explicación con las características de los hoteles, fotos, videos, precios aproximados, etc.



Imagen 8.5 Hoteles

Playa:

El modelo turístico de Gandía sigue siendo hoy en día un turismo de sol y playa. La playa atrae a miles de turistas, sobre todo del centro del país. Como la playa es por lo que más turistas deciden visitar Gandía, se añadió una muy breve información con una imagen a esta aplicación. En el texto informativo se hace referencia a las dimensiones de la playa arenosa, a los servicios ofrecidos y a la bandera azul recibida año tras año como símbolo o garantía de calidad.



Imagen 8.6 Playa

Lugares de Interés:

A pesar de ser un destino de sol y playa, Gandía cuenta con muchos lugares de interés. En el centro histórico de la ciudad se encuentran varios monumentos históricos como la Colegiata, el Palacio Ducal, los últimos restos de la antigua muralla de la ciudad (un torreón), etc. La aplicación muestra parte de la información de la web gandiaturisticacom y permite localizar cada punto de interés en un mapa. Ofreciendo esta información en la aplicación se espera que más turistas de sol y playa de Gandía se interesen por el patrimonio cultural de la ciudad. Inicialmente se presentan unos dieciseis lugares de interés repartidos por todo el municipio de Gandía. La lista podría ser ampliada posteriormente.



Imagen 8.7 Lugares

Fiestas:

A lo largo de todo el año se celebran diferentes fiestas y eventos en la ciudad que se da a conocer en esta activity. El usuario encuentra no sólo información sobre las dos fiestas más importantes de la ciudad ("Fallas" y "Feria y Fiesta") sino también sobre la Semana Santa, noche de San Juan y otras fiestas no muy promocionadas por el Ayuntamiento de Gandía.

Las fiestas se celebran durante diferentes épocas del año por lo que se podría aprovechar ésto para desestacionalizar un poco el turismo de Gandía.



Imagen 8.8 Fiestas

Compras:

En el apartado de compras se indica a los visitantes de la ciudad las calles y los centros comerciales más importantes de la ciudad. Gandía cuenta con varios de ellos. Cada centro comercial o calle comercial cuenta con muchas tiendas de moda, cafeterías y restaurantes. Los centros comerciales y calles comerciales se encuentran en diferentes puntos de Gandía permitiendo al turista escoger entre una gran oferta.

En el centro de Gandía se encuentra la calle Mayor que es la principal calle comercial y se encuentra cerca de algunos de los lugares de interés de la ciudad (Escuelas Pías, la Colegiata, Palacio Ducal, etc.). Los centros comerciales Plaza Mayor y la Vital se encuentran en las afueras de la ciudad. Finalmente en la playa se encuentra el centro comercial Palace situado en el complejo del Hotel Gandía Palace.



Imagen 8.9 Compras

Localización:

Aquí el turista recibe información básica sobre la situación geográfica del municipio de Gandía y las distancias hasta los próximos aeropuertos de Valencia (Manises, a 65 km) y Alicante (el Altet a 116 km). Finalmente se describe muy brevemente el clima de la región.

Esta información puede ser relevante para turistas que no conocen Gandía aún y planean su primer viaje.



Imagen 8.10 Localización

Galería de Fotos:

La galería de fotos muestra fotos de los principales lugares de interés de la ciudad. Se trata de una galería por la que el usuario puede moverse, gracias a la pantalla táctil, “empujando” las fotos con un dedo a los lados para ver la siguiente fotografía. En la galería se encuentran las fotografías de todos los lugares de interés y de la playa de Gandía.



Imagen 8.11 Galería de fotos

Teléfonos:

En el listado de teléfonos el usuario encuentra de forma rápida los teléfonos más importantes de la ciudad. La aplicación permite realizar las llamadas directamente desde la aplicación, pinchando sobre el teléfono al que se quiere llamar. La lista de teléfonos contiene los números de teléfonos de las oficinas de turismo de Gandía, de información de RENFE, del Ayuntamiento y varios teléfonos en caso de emergencias (Urgencias del Hospital San Francisco de Borja y de los diferentes centros de salud, así como de bomberos y policía. La realización de llamadas desde una aplicación evita que el usuario tenga que buscar el número de teléfono al que quiere llamar y teclear los números. Es suficiente con pinchar sobre el elemento correspondiente de la lista y automáticamente se realiza la llamada.



Imagen 8.12 Teléfonos

Página Web:

En esta *activity* se muestra la página web turística de Gandía donde el usuario encuentra mucha información adicional sobre la ciudad. GandiaTurística cuenta con información de todos los lugares de interés de la ciudad, medios de transportes, gastronomía, hoteles y mucho más.



Imagen 8.13 Web Gandia

Sobre la App:

En este lugar se encuentra información sobre la creación de la aplicación. La aplicación fue desarrollada durante prácticas de empresas con la empresa APLICAE de Gandía. Aplicae es una empresa de I+D+i que se dedica a fomentar la innovación y el uso de las nuevas tecnologías en empresas. Su página web es www.aplicaeinnovacion.com. La presente aplicación, como primera aplicación turística de la ciudad, es un buen ejemplo del uso de nuevas tecnologías en el sector turístico.

Toda la información (tanto textos como imágenes) fueron sacados de la web turística de Gandía www.gandiaturista.com que tiene todos los derechos. Este portal turístico es un portal privado (no pertenece al ayuntamiento) y se convirtió en los últimos años en el portal turístico más importante de la ciudad.



Imagen 8.14 Web Aplicae

9. Programación del código fuente

En este apartado se explica cómo se consiguieron los objetivos propuestos para este proyecto. Cada objetivo es explicado, tomando una determinada *activity* como ejemplo, mediante código fuente y capturas de pantallas del emulador del entorno de programación Eclipse. Antes de comenzar con la explicación de los códigos fuente de las actividades relacionadas con los objetivos, hace falta explicar el funcionamiento básico de algunos ficheros.

9.1 Ficheros

En el apartado 6.3 (partes de una aplicación) se dio una breve introducción a los diferentes ficheros. En este apartado se explica detalladamente el uso de varios de ellos. Se diferencia principalmente entre los ficheros de diseño (layout) de tipo xml, los ficheros en los que se encuentra el código fuente de las diferentes actividades de tipo *.java y el fichero llamado "Android Manifest" (tipo xml) donde se encuentran los permisos requeridos por la aplicación.

9.1.1 fichero de layout *.xml

Al crear una nueva *activity*, el programador siempre comienza por la parte de diseño. Esto es fijar el número de botones, campos de texto, imágenes, etc. de la *activity*. Esto se hace escribiendo el código en un fichero tipo *.xml. Es interesante saber que un mismo fichero de diseño *.xml puede ser usado para varias actividades. Éstas, en consecuencia, tendrán el mismo aspecto. Como ejemplo del funcionamiento se explica el código xml de la *activity* "playa". Se trata de una *activity* relativamente simple. En la parte superior (como se muestra en la imagen 9.1) se encuentra una imagen de la playa, seguida de un título y una breve descripción de las características de la playa. Al haber demasiada información como para que ésta apareciese en la pantalla antes del pliege, hace falta indicarle al dispositivo que la ventana permite un desplazamiento vertical tal como se conoce de páginas webs o ficheros de texto para acceder a la continuación del texto.



Imagen 9.1 Activity Playa

El código completo se encuentra en el anexo 3. Aquí se explican los diferentes apartados.

La primera línea del fichero *.xml (en este caso playa.xml) debe comenzar con la siguiente línea en la que se indica que el contenido es código xml en versión 1 y codificado en utf-8. Utf-8 significa que se trata del formato Unicode Transformation Format de 8 bits y describe la forma de codificar los caracteres de texto.

```
<?xml version="1.0" encoding="utf-8"?>
```

A continuación comienza el verdadero diseño de la *activity*. Al principio se establece la forma de diseñar la *activity*. Se puede elegir entre diferentes *views* (vistas). Cada tipo de *view* permite ordenar los componentes que forman la *activity* de una forma determinada. Así se utiliza por ejemplo un *listview* para crear una lista de componentes, un *gridview* para ordenar

imagenes en una matriz, un *webview* para mostrar una página web, un *mapview* para mapas, etc.

En éste caso se usó el *Relative Layout*. Se trata de la forma habitual de organizar varios componentes (como botones, campos de texto, etc.) en una *activity*.

```
<RelativeLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="0px"
    android:background="#BEBABA">
```

En color morado, seguido de "android:" se encuentran las características del elemento. En este caso se indica con `layout_width="fill_parent"` que el ancho del elemento de tipo *RelativeLayout* tiene que ocupar todo el ancho de la pantalla. El mismo comportamiento tendrá `layout_height` pero en vertical.

`android:padding="0px"` indica que el borde es de tamaño 0 pixeles (no hay borde). Finalmente con `android:background="#BEBABA"` se indica el color de fondo de la *activity*. El valor en azul es el código de color en hexadecimal y equivale a un gris claro. Finalmente se cierra el elemento mediante el símbolo `>`.

El siguiente elemento es un *ScrollView*. El *scrollview* es una vista que permite al usuario deslizarse por la pantalla para acceder a la información que se encuentra por debajo del pliego de la página o pantalla. En la primera línea encontramos el código `android:id="@+id/ScrollView01"`. *ScrollView01* es el nombre de la identidad (*id*) de este *ScrollView*. Esta *id* es la identificación necesaria para referenciarse a este *scrollview* en concreto desde otro fichero (como por ejemplo desde *playa.java*). El ancho de este elemento se fijó como `wrap_content` que enmarca su contenido. `Wrap_content` se ajusta siempre al tamaño al que se referencia mientras que `fill_parent` intenta englobar un campo más grande.

```
<ScrollView android:id="@+id/ScrollView01"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent">
```

En algunos casos (sobre todo al hacer uso de varios *views* diferentes seguidos) hace falta añadir otro *RelativeLayout* para obtener el resultado deseado. En este caso el código indica que la forma de realizar el diseño dentro del campo del *scrollview* (que abarca el contenido completo de esta *activity*) es la de un *RelativeLayout*.

```
<RelativeLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#BEBABA">
```

A continuación comienza el código de aquellos elementos que el usuario ve en la *activity*. Debido a que en la parte superior de la *activity* se encuentra la imagen de la playa de Gandía, se introduce a continuación un *ImageView*. Esto es un campo capaz de mostrar imagenes.

En este caso la *id* de la imagen se llama "foto_monumento". En principio (según el código que se encuentra en la siguiente página) debería aparecer en ese campo la imagen "colegiata" que se encuentra en la carpeta "drawable". Esto no pasará al poderse realizar el diseño también directamente desde el código Java dónde se le indica a la aplicación que muestre la imagen "playa" en el *ImageView* llamado "foto_monumento". La ventaja de esta forma de programar consiste en la reutilización de códigos. Esto permite que otra *activity* (que tenga el mismo diseño como la *activity* "playa") acceda a este fichero de diseño (*playa.xml*) y que en el código

fuentes de su fichero *.java introduzca una información destinada a la de playa.java. Es posible programar una aplicación con varias actividades (*.java), cada una con otra información, pero con un mismo diseño al acceder todas las actividades a un mismo fichero de diseño *.xml.

Finalmente en las últimas líneas se indica que la imagen no debe dejar márgenes en su parte superior ni a ambos lados (en código: que deje un margen de cero píxeles).

```
<ImageView android:id="@+id/foto_monumento"
    android:layout_below="@id/ScrollView01"
    android:src="@drawable/colegiata"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:layout_marginTop="0px"
    android:layout_marginRight="0px"
    android:layout_marginLeft="0px"
</ImageView>
```

Debajo de la foto de la playa (del *ImageView*) aparece el título de esta *activity*. Al estar en una vista relativa (*Relative Layout*) se indica la posición del *Textview* en relación con otro elemento (en este caso el *ImageView*). En la tercera línea de este trozo de código se establece que el *TextView* tiene que ser mostrado debajo de la id "foto_monumento" mediante el código `android:layout_below="@id/foto_monumento"`. Como se observa en la imagen, tiene letra blanca sobre un fondo rojo. Esto se indica en el fichero xml de la siguiente forma:

Color de texto negro: `android:textColor="#ffffff"`
Color de fondo rojo: `android:background="#A70505"`

El tamaño de texto será de 19 dip (density independent pixels). El dip es una unidad abstracta y se refiere a los píxeles físicos de una pantalla. La línea `android:padding="15dip"` indica que el texto comienza después de 15dip. Se puede decir que es un tipo de sangría o espacio.

```
<TextView
    android:id="@+id/titulo"
    android:layout_below="@id/foto_monumento"
    android:background="#A70505"
    android:layout_width="fill_parent"
    android:textColor="#ffffff"
    android:textSize="19dip"
    android:padding="15dip"
    android:layout_height="wrap_content"
/>
```

Como último elemento de esta *activity* se añade un segundo *Textview* que incluirá la información turística sobre la playa de Gandía. Al tratarse de un elemento igual al visto anteriormente, su función es la misma. Sólo su diseño cambia ligeramente. Este elemento, en vez de tener un color de fondo, cuenta con una imagen de fondo. La imagen se llama "text_back400" y el código relacionado es `android:background="@drawable/text_back400"`.

Este *TextView* se encuentra por debajo del elemento con id "titulo", tiene color de fondo negro (#000000) y una separación del margen de 10dip.

```
<TextView
    android:id="@+id/informacion"
    android:background="@drawable/text_back400"
    android:layout_below="@id/titulo"
    android:layout_width="fill_parent"
    android:textColor="#000000"
```

```

        android:layout_height="wrap_content"
        android:padding="10dip"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
    />

```

Después de cerrar este elemento (*TextView*) con los símbolos `/>`, es necesario cerrar todos los demás elementos anteriores. Siempre se cierran desde dentro a fuera. Los elementos están organizados en una jerarquía determinada. El primer elemento en ser abierto (en este caso el *RelativeLayout* de la segunda línea) es el último elemento en ser cerrado. El *ScrollView* fue el segundo elemento abierto y se cierra en penúltimo lugar.

Para cerrar un elemento se comienza por escribir `</tipo del elemento>`

En este caso:

```

</RelativeLayout>
</ScrollView>
</RelativeLayout>

```

Habitualmente es suficiente con escribir los símbolos `</` y el entorno Eclipse escribe automáticamente el nombre del siguiente elemento que tiene que ser cerrado.

9.1.2 fichero *.java

Los ficheros *.java contienen el código fuente que determina el comportamiento de la aplicación. En este lugar se especifican las acciones que se realizan cuando el usuario interviene en la aplicación (por ejemplo pulsando un botón). Un fichero Java de este tipo siempre está relacionado con otro fichero xml que determina su diseño. En este caso el fichero `playa.java` está vinculado al fichero `layout/playa.xml` visto en el apartado anterior. El código fuente del fichero `playa.java` es muy simple. Su única función es la de mostrar un contenido determinado. Se trata de una pantalla estática (siempre muestra la misma información). Los textos informativos se encuentran guardados como strings en un fichero al que la aplicación tiene que acceder y la imagen de la playa se encuentra en la carpeta "drawables". A continuación se explica su código fuente. Como en el caso anterior, el código completo se encuentra en el anexo 4.

La primera línea de código tiene que llevar necesariamente el nombre del paquete. En este caso puede ser `com.gandiaturistica.gandia` o `com.aplicaeinnovacion.gandia`. Como visto en el punto 6.2, se trata de un nombre único. No pueden haber dos aplicaciones con un mismo nombre de paquete.

```

package com.gandiaturistica.gandia;

```

En segundo lugar se incluyen (importan) todas las librería necesarias para ejecutar las diferentes funciones usadas. Es posible añadirlas directamente pero resulta más cómodo no escribir nada inicialmente y a medida que se utilizan funciones que necesitan librerías especiales para funcionar, pulsar la combinación de teclas "Control O". Con esto se añaden automáticamente y el programador, en la mayoría de los casos, no se tiene que preocupar de los nombres exactos de las librerías usadas.

```

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;

```

```

import android.view.MotionEvent;
import android.view.View;
import android.widget.ImageView;
import android.widget.ScrollView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ViewFlipper;

```

A continuación comienza el código fuente en el que se declaran variables y se establece el comportamiento de la activity.

Se crea una clase pública (en el punto 4.3 se encuentra una introducción sobre la declaración de clases y variables) con el nombre del fichero actual (playa) y se indica que se trata de una activity. Las variables en su interior serán de tipo *TextView* (para el título y la información) e *ImageView* (para la imagen de la playa). Se accede a ellas a través de sus nombres “CampoTitulo”, “CampoInformacion” y “CampoPlaya”.

```

public class playa extends Activity {
    private TextView CampoTitulo;
    private TextView CampoInformacion;
    private ImageView CampoPlaya;

```

A continuación viene la función *onCreate* que es llamada cuando la *activity* es creada por primera vez tal como lo indica el comentario en letra azul. Los comentarios no son interpretados por los compiladores. Sirven sólo como ayuda para el programador. Los comentarios pueden encontrarse en el centro de los símbolos */** y **/* para abarcar varias líneas, o ir detrás de dos barras *//* para comentar una única línea.

```

/* Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

```

A continuación se realiza la conexión con el fichero de diseño (*layout*) de la *activity*. En este caso al relacionarlo con *playa.xml* se escribe la siguiente línea de código:

```

setContentView(R.layout.playa);

```

Una vez establecida la relación con el fichero *layout*, se relacionan los elementos de éste (los dos *TextViews* y el *ImageView*) con sus correspondientes campos en la aplicación. Para ello se usan las variables tipo *TextView* e *ImageView* declaradas en la clase “playa” al principio.

A la función *findViewById* se le da una *id* (“titulo” por ejemplo) y la función devuelve un *TextView*. La variable (tipo *TextView*) *CampoTitulo* obtendrá éste valor.

```

CampoTitulo = (TextView) findViewById(R.id.titulo);
CampoInformacion = (TextView) findViewById(R.id.informacion);
CampoPlaya = (ImageView) findViewById(R.id.foto_monumento);

```

Para estructurar el código mejor y dejarlo más legible, es preferible hacer uso de funciones. Esto es realizar las tareas y operaciones en funciones fuera de la función principal. Desde ésta se puede llamar al resto de funciones. Como ejemplo se crea la función “mostrarMonumentos”. Esta, con pequeños cambios se usa tanto en *playa.java* como en *monumentos.java* (de allí el nombre de “monumentos”). Algunos códigos se pueden reutilizar (con leves cambios) lo que hace más eficiente el proceso de programación. En el caso de tener

que realizar una misma operación en varios lugares del código de la misma aplicación, conviene mucho hacer uso de funciones al poder ahorrar así tiempo de programación y líneas de código. El uso de funciones, clases y objetos está explicado en el punto 4.3.

En la siguiente línea se llama la función `mostrarMonumento` y se cierra la función `onCreate` con el símbolo `}`.

```
        mostrarMonumento();
    }
```

A continuación se encuentra la función `mostarMonumento()`, llamada en las líneas anteriores. Esta función es pública (como la mayoría de funciones) y no devuelve ningún valor. Esto se indica con la palabra reservada `void`. En el interior de la aplicación se indica que textos tienen que aparecer en los `TextViews` (campos de texto) y qué imagen en el `ImageView`. Los textos se encuentran en el fichero `strings.xml` dentro de la carpeta “values” dentro de la carpeta “res”, mientras que las imágenes se guardan en la carpeta “drawable”.

```
public void mostrarMonumento() {
    CampoTitulo.setText(R.string.titulo_playa);
    CampoInformacion.setText(R.string.informacion_playa);
    CampoPlaya.setImageResource(R.drawable.playa);
}
```

En este caso la imagen se llama `playa.jpg` (no hace falta escribir la extensión del fichero) y los `strings` se llaman “`titulo_playa`” e “`informacion_playa`”. El contenido correspondiente a estos dos `strings` del fichero `strings.xml` es:

```
<!-- Playa -->
    <string name="titulo_playa">La Playa</string>
    <string name="informacion_playa">Gandia cuenta con una playa de
7,5 Km de longitud. El ancho de la playa de arena blanca es de 150
metros y dispone de todos los servicios. [...]</string>
```

A diferencia de los ficheros Java, en los ficheros xml los comentarios se escriben dentro de los símbolos `<!--` y `-->`. El comentario ayuda a encontrar dentro del fichero de `strings` aquellos textos que contienen la información de la playa de Gandía. A continuación se encuentran los dos `strings` (`titulo_playa` e `informacion_playa`) con sus respectivos contenidos.

Con “`CampoTitulo.setText(R.string.titulo_playa);`” se inserta el texto con nombre “`titulo_playa`” de `strings.xml` en el `Textview` con nombre `CampoTitulo`.

Una vez realizadas todas las operaciones (mostrar la información en el lugar deseado) se cierran los corchetes para cerrar la función “`mostrarMonumento`”.

La última función en esta `activity` se llama `onBackPressed` y se ejecuta cuando el usuario de la aplicación pulsa el botón (hardkey) de “atrasar” o “back” en el smartphone. Si no se indica cual es el uso del botón “back” en una aplicación, al pulsarlo, puede que o bien cierre la aplicación entera o bien no realice ninguna acción.

En la aplicación de Gandía, la función del botón “back” es la de atrasar al nivel o a la pantalla anterior (éste es el uso habitual para éste botón). Como se observa en el anexo 2 (estructura de la aplicación de Gandía) se accede a la pantalla (`activity` o `view`) de `playa` a través de la lista principal. Cuando el usuario se encuentre en la `activity` `playa` y pulse el botón de atrasar (back), tendrá que ser enviado de nuevo a la lista principal. Para ello se usan los “`intents`”.

```

@Override
public void onBackPressed() {
    Intent intent = new Intent(playa.this, lista_principal.class);
    startActivity(intent);
    finish();
    return;
}
}

```

Los *intents* son usados para abrir actividades nuevas. En este caso se crea un *intent* nuevo llamado “intent” (como ejemplo hubiera sido mejor darle otro nombre, pero a la hora de programar es más fácil porque el nombre indica lo que realmente es). “Intent intent” significa que se creó un *Intent* con nombre “intent”. Este *intent* recibe el valor del nuevo *intent* (new Intent) que cambia desde ésta *activity* (playa.this) a la actividad llamada “lista_principal”. En las siguientes líneas se ejecuta la nueva *activity* con *startActivity(intent)*; y se finaliza la actual. También es posible enviar información de una *activity* a la siguiente mediante intents como se hace por ejemplo en la *activity* de los números de teléfonos. En este caso se envía el número de teléfono guardado en esta *activity* a la *activity* “llamar” usando el comando “setData”. Esto se ve con más detalles en la explicación de dicha *activity*.

9.1.3 Android Manifest

Para que la *activity* playa.java anteriormente creada y explicada se ejecute como aplicación en el emulador o en un teléfono móvil con sistema operativo Android, hace falta indicarle al fichero AndroidManifest.xml los permisos necesarios. El uso de permisos es una práctica habitual que caracteriza los sistemas que funcionan bajo Linux. Como Android se basa en el sistema Linux hace falta repartir permisos para cada acción que se realiza. Esto es una medida de seguridad para evitar que se puedan ejecutar códigos malignos u otros códigos no deseados.

El AndroidManifest contiene diferentes informaciones respecto a los permisos. El apartado más grande y necesario en todas las aplicaciones es el siguiente:

```

<application android:icon="@drawable/icon"
android:label="@string/app_name"> </application>

```

La primera línea indica cual es el dibujo del icono de la aplicación (el aspecto del acceso directo en la pantalla del móvil). En este caso se usa el icono llamado “icon” (guardado en drawable) con el logotipo de la web de gandiaturisticacom que muestra una “G de Gandia” dentro de una “@”. El logotipo es moderno y muestra la relación entre el turismo y los nuevos medios de telecomunicación en gandiaturisticacom. Seguido al código del icono se encuentra el nombre que aparece siempre en la parte superior de la aplicación. En este caso la aplicación accede al nombre guardado en el *string* “app_name” guardado en strings.xml. El *string* guardado en ese lugar contiene el nombre “Gandia” por lo que en la parte superior de cada *activity* aparecerá una barra gris en la que pondrá “Gandia”.

El contenido de `<application [...]>` es una lista de todas las actividades con permisos para ejecutarse y sus respectivas configuraciones. Si una *activity* no se encuentra en esta lista y se intenta acceder a ella, la aplicación devuelve un error y se cierra. En el caso de la *activity* de la playa se escribe el siguiente código. Detrás de `android:name` siempre va el nombre del fichero Java correspondiente. El resto del código suele ser igual para todas las actividades pero puede haber cambios.

```
<activity android:name=".playa" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>
```

Algunos de los posibles cambios que se pueden realizar en el comportamiento de una *activity* desde el *AndroidManifest* se muestran en la pantalla inicial de la aplicación (la que se ejecuta al pinchar sobre su icono). En el caso de la aplicación de Gandía se abre una imagen de la playa de Gandía con la montaña Mondúber al fondo. Esta imagen se abre a pantalla completa, sin el nombre de la aplicación en la parte superior (como explicado en la configuración de la *activity* de la playa). Tampoco aparece en esta *activity* la barra de menú del sistema operativo Android desde el cual se accede a los mensajes, el estado de batería, etc. Para lograr esto se usa el siguiente código:

```
<activity android:name=".splashscreen"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen">
    <intent-filter>
        <category android:name="android.intent.category.LAUNCHER"/>
        <action android:name="android.intent.action.MAIN" />
    </intent-filter>
</activity>
```

La *activity* se llama "splashscreen" (el nombre inglés para las páginas de inicio que muestran una imagen y desaparecen después de unos segundos). En la tercera línea se indica el estilo (theme) de la actividad. El estilo es *NoTitleBar* y *Fullscreen* (sin barra de título y pantalla completa). En la *activity* "mapa_gandia" se usa por ejemplo sólo el estilo *NoTitleBar* sin indicar que sea *fullscreen*. Con esto no aparece la barra superior con el nombre de la aplicación pero sí la barra de menú de Android. Para indicar a Android que la *activity* "splashscreen" es la principal (la primera en ejecutarse al abrir la aplicación) se añade la categoría "LAUNCHER" (la primera en disparar o saltar).

En el fichero *manifest*, habitualmente antes de la lista de actividades, se añaden los permisos especiales para los usos de determinado hardware. En la aplicación de Gandía se usa el acceso a Internet para visitar las diferentes webs o para rastrear una web y adquirir el pronóstico de tiempo, se usa el teléfono para realizar llamadas telefónicas y finalmente para la localización exacta del usuario y el uso del mapa de Gandía se usa la brújula electrónica y el receptor GPS. Para que una aplicación pueda acceder a éstos y más recursos hacen falta los permisos. En el caso de la aplicación de Gandía hacen falta los siguientes permisos:

```
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

```

<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.INTERNET"></uses-
permission>

```

Cuando el usuario se instala una aplicación en su smartphone y ésta aplicación requiera permisos especiales, el usuario tiene que aceptarlos para poder continuar con la instalación. Con esto el usuario es advertido que una aplicación puede posiblemente determinar su posición geográfica o que la aplicación puede generarle gastos al usuario. Esto puede pasar en el caso de conexiones a Internet y en llamadas telefónicas.

En una de las últimas líneas del AndroidManifest de la aplicación actual se encuentra la línea `<uses-library android:name="com.google.android.maps" />` dónde se indica que se accede a una librería externa. Esta librería hace falta para poder mostrar los mapas de Google (googlemaps) en la aplicación. En este caso se trata de una API y requiere una librería que no se encuentra por defecto en el entorno de programación Eclipse.

9.2 Objetivos

El objetivo principal de este proyecto fue el de la programación y diseño de una aplicación para móviles Android (programada en lenguaje de programación Java) para ser usada en el sector turístico. Para ello se fijaron varios objetivos secundarios. Entre los objetivos secundarios relacionados a los recursos de los smartphone y la programación en Java se encuentran los siguientes:

- Acceder y adquirir información automáticamente de determinadas páginas webs o widgets de Android.
- Añadir una API (application programming interface) de Google en la aplicación (comunicación entre dos softwares distintos).
- Utilización de los componentes hardware disponibles en el dispositivo móvil como el GPS, el acelerómetro, la brújula, la pantalla táctil, botones (hardkey), etc.
- Crear funciones para facilitar la posterior ampliación de contenido.

Estos puntos se explican haciendo uso de su código fuente e imágenes en los puntos 9.2.1 hasta 9.2.4.

9.2.1 Adquisición automática de datos

Para facilitar el uso de la aplicación por parte del usuario se decidió ofrecerle datos actuales y un pronóstico de tiempo para varios días del estado atmosférico incluyendo la temperatura actual en grados centígrados así como las temperaturas máximas y mínimas para los próximos días, la humedad del aire en porcentajes, la dirección del viento y el estado de la atmósfera (soleado, nublado, lluvioso, etc.). Los datos son leídos de Internet (por lo que hacen falta los permisos de acceso a Internet) de una página de Google con código xml. Este código es leído por la aplicación de Gandía. Realmente se trata de una API, un programa que crea un código xml que es interpretado por la aplicación de Gandía. La tarea fundamental de la aplicación será la de sacar la información relevante del código fuente y mostrarla en la aplicación. Como la información



Imagen 9.2 Activity del tiempo

ofrecida por Google es en inglés, la aplicación tiene que comparar los *strings* ingleses del fichero xml con *strings* guardados en la aplicación para posteriormente mostrar su traducción en castellano (otros *strings* guardados en un fichero de la aplicación de Gandía). Los diferentes puntos para lograr el objetivo son bastante claros. En cambio la programación para conseguir los resultados deseados resultó ser complicada. La *activity* del pronóstico del tiempo consta de una única pantalla, pero para adquirir y procesar la información se tuvieron que programar cuatro ficheros tipo *.java y uno tipo *.xml (adicionalmente se encuentra el fichero xml de Google que la aplicación tiene que interpretar). En total hay seis ficheros de códigos involucrados en la tarea de mostrar un pronóstico de tiempo fiable.

El código completo ofrecido por la API de Google para el día 19.9.2011 se encuentra en el anexo 5. En este apartado se muestra sólo el pronóstico de tiempo para el siguiente día.

En el momento de redactar este proyecto y acceder a la información de la API de Google desde el ordenador, se observó que Google ya ofrece la descripción del pronóstico de tiempo en castellano también (al detectar el idioma preferido o la dirección ip española). En la sexta línea el dato "condition" tiene el valor "Mayormente soleado". En la aplicación de Gandía se usa el mismo fichero pero en inglés por lo que hace falta traducir el contenido de "condition" al castellano.

```
- <forecast_conditions>
  <day_of_week data="lun"/>
  <low data="15"/>
  <high data="27"/>
  <icon data="fig/images/weather/mostly_sunny.gif"/>
  <condition data="Mayormente soleado"/>
</forecast_conditions>
```

Los valores mostrados en "low" y "high" son las temperaturas mínimas y máximas respectivamente. Debajo de estas siete líneas de código se repiten las mismas líneas de código con información diferente (para cada día). Al interpretar el contenido de ese fichero xml proporcionado por la API de Google hace falta controlar la posición exacta de la línea de código que se interpreta. En caso contrario se podrían sobrescribir los datos en la aplicación y mostrar pronósticos falsos. Para interpretar el código anterior y relacionar por ejemplo el valor 15 con la temperatura mínima hace falta usar un analizador (*parser*) que rastrea o lee el código completo. En este caso se usó un *parser* tipo SAX (SAX Parser). El acrónimo SAX significa "Simple API for XML" y fue desarrollado inicialmente sólo para el lenguaje Java. Hoy existen también versiones para otros lenguajes de programación.

Un SAX Parser lee el contenido de un fichero secuencialmente desde el principio hasta el final y detecta cuando empieza y acaba un elemento. Los elementos comienzan con el símbolo < y acaban con > o /> como se observa en el ejemplo del pronóstico de tiempo. Una de las ventajas de un *parser* tipo SAX es que inicialmente no carga todo el código en memoria al tratar cada elemento por separado. Esto hace que la lectura e interpretación del fichero *.xml sea más rápida que la de un *parser* tipo DOM (Document Object Model) que carga el código completo en memoria.

Cada vez que el *parser* llega a un nuevo elemento, éste tiene que ser leído, interpretado y guardado en una variable.

El funcionamiento, de los diferentes ficheros para realizar estas tareas en esta aplicación, es el siguiente:

El fichero parseweb.java es el primero en ser usado. En él se realizan algunas configuraciones del SAX Parser, se indica la URL del fichero *.xml a analizar y se fija el diseño de la *activity* (por un lado se indica que el fichero de *layout* es eltiempo.xml, por otro lado se toman ajustes de diseño en el mismo código java). Este código de la API sólo necesita algunos cambios. La lectura del fichero deseado comienza con el comando `xr.parse(new InputSource(url.openStream()))`;

A continuación, para hacer funcionar el *parser* de la forma deseada, el programador tiene que programar el resto de ficheros. El fichero ExampleHandler.java recibe los elementos leídos por el *parser* y los compara con nombres de determinadas variables para poder a continuación guardar los valores leídos en las variables correspondientes. En el código xml del pronóstico de tiempo de Google se diferencia primero entre el tiempo actual (current conditions) y el pronóstico (forecast conditions) de tiempo. Dependiendo del valor del elemento correspondiente se fijan los valores de dos variables booleanas con los valores “true” y “false”. Esto es importante a la hora de interpretar otros elementos. Esto se resolvió con el siguiente código:

```
if (localName.equals("current_conditions")) {
    this.in_current_conditions = true;
    this.in_forecast_conditions = false;
}
else if (localName.equals("forecast_conditions")) {
    this.in_forecast_conditions = true;
    this.in_current_conditions = false;
}
```

Como ejemplo de interpretar un valor se puede ver la parte del código referente a la humedad.

```
else if (localName.equals("humidity") && (in_current_conditions==true)) {
    String attrValue = attrs.getValue("data");
    String humedad = attrValue;
    myParsedExampleDataSet.setHumedadHoy(humedad);
    this.in_mytag = true;
}
```

En este caso se compara si el nombre del elemento en el que se encuentra el *parser* equivale a “humidity” y si se encuentra al mismo tiempo dentro del elemento “current conditions”. Si es así, se recibe el valor guardado en “data”. Este valor es un *string* con información parecida a la siguiente: “Humidity: 29%”. A continuación se envía este *string* a la función setHumedadHoy() del fichero ParsedExampleDataSet.java donde se “filtra” el dato del porcentaje. Sólo interesa el 29 o 29% y no la palabra Humidity ni los dos puntos. Para ello se crea un substring y se llama una función nueva.

```
String substringHumedad(String extractedString) {
    String subString = extractedString.substring(10,13);
    return subString;
}
```

La tarea de este substring es la de extraer los símbolos y caracteres entre las posiciones 10 y 13 del *string* original. Como la primera posición del string comienza en 0 (la H de Humidity se encuentra en la posición 0), la posición 10 viene después del espacio en blanco después de los dos puntos. El resto de datos es el "19%".

Este valor es devuelto al fichero inicial (parseweb.java) por el que es mostrado en pantalla. El mismo proceso se realiza en los casos de la temperatura actual y la dirección del viento.

La interpretación del pronóstico de tiempo para los próximos días es más complicado. Esto es debido a que los códigos correspondientes al pronóstico del siguiente día, en dos días o en tres días no se diferencian (no hay ningún elemento que los identifique claramente). El trabajo sería más fácil si hubiesen elementos llamados "forecast 1", "forecast 2", etc. Para solucionar el problema se inicializa al principio una variable con valor cero (int dia = 0); Cada vez que el *parser* encuentra un elemento <forecast_condition> interpreta su contenido y luego incrementa la variable "dia" en uno para indicar que el siguiente elemento pertenece a información de otro día. Como código ejemplo se toma el estado de la atmósfera (lluvia, sol, etc.).

```
// el tiempo cada día
else if (localName.equals("condition") && (in_forecast_conditions==true))
{
    if (dia==0){
        String attrValue = atts.getValue("data");
        String condicion = attrValue;
        myParsedExampleDataSet.setCondicionCero(condicion);
        this.in_mytag = true;
    }
    else if(dia==1){
        String attrValue = atts.getValue("data");
        String condicion = attrValue;
        myParsedExampleDataSet.setCondicionUno(condicion);
        this.in_mytag = true;
    }
    [...]
    dia++;
}
```

Este código es parecido al aplicado en la lectura de la humedad. La diferencia es que ésta vez, dependiendo del día, se llama a funciones diferentes (setCondicionCero, setCondicionUno, setCondicionDos, etc.). El *string* attrValue o condicion, contiene una cadena de texto que describe la situación del tiempo (en inglés). Para que los usuarios de la aplicación de Gandía puedan leer esta información en castellano, hace falta una traducción.

Se llama a la función de traducción mediante la función traducir().

```
public void setCondicionUno(String eltiempo) {
    //Traducir al castellano
    eltiempo = traduccionTiempo.traducir(eltiempo);
    this.condicionUno = eltiempo;
}
```

La función traducir se encuentra en el fichero traduccionTiempo.java. La función se podría haber incluido al final del código fuente de otro fichero ya existente, pero de ésta forma se

muestra cómo llamar funciones desde otro fichero. Para ello se escribe el nombre del fichero y a continuación el nombre de la función (separados por un punto) como se muestra en la tercera línea del código anterior.

La función “traducir” del nuevo fichero consiste en una comparación de *strings* ingleses. Si ambos *strings* comparados coinciden, se devuelve el nombre en castellano de ese string.

A continuación se muestra un extracto del código. El código completo tiene más de treinta condiciones *if* (una para cada estado atmosférico).

```
public class traduccionTiempo {
    public static String traducir(String eltiempo) {
        if (eltiempo.equals("Partly Cloudy")) {
            eltiempo="Parcialmente nublado";
        }
        else if (eltiempo.equals("Sunny")) {
            eltiempo="Soleado";
        }
        else if (eltiempo.equals("Partly Sunny")) {
            eltiempo="Parcialmente soleado";
        }
        [...]
        return eltiempo;
    }
}
```

En cada condición *if* y *else if* se comprueba si el *string* “eltiempo” equivale al *string* entre comillas. Si el *string* coincide con “Sunny”, el *string* “el tiempo” obtendrá el valor “Soleado” y la función devolverá ese valor (*return eltiempo*);. En estas líneas se encuentra otra particularidad del lenguaje de programación Java. Mientras que en C++ se pueden comparar dos strings usando dos símbolos de igualdad seguidos (*eltiempo=="Sunny"*), en Java hace falta usar la palabra reservada *equals* para la comparación de dos *strings*. Se trata de un leve cambio respecto a C++ que los programadores de éste lenguaje deben tener en cuenta a la hora de programar. Si se realiza la comparación igual como en C++ el compilador avisa de un error y no compila el código.

Una vez el *parser* llega a la línea de código `</forecast_conditions>` detecta que el elemento termina en este lugar. El *parser* sigue leyendo las siguientes líneas del fichero xml y vuelve a realizar las tareas anteriores para el pronóstico de tiempo del resto de días.

La programación de los diferentes ficheros para interpretar el fichero xml ofrecido por Google para el pronóstico de tiempo resultó ser una de las partes más complicadas del proyecto. Los parsers de Java para aplicaciones de Android no están lo suficientemente documentadas. Los ejemplos básicos no permiten las tareas necesarias para interpretar el fichero que se usa en este proyecto. Es necesario programar el *parser* para cada fichero xml porque cada fichero tiene una arquitectura y unos elementos diferentes. En este caso además de la difícil programación de los parsers, era necesario solucionar el problema de la identificación de los

elementos repetidos (datos relacionados con diferentes días), la necesidad de sacar información determinada de varios *strings* (como en el caso de la humedad) y la comparación y traducción de otros *strings*.

9.2.2 Utilización de una API

El segundo objetivo de este proyecto fue el de usar una API para conseguir que dos programas informáticos (la aplicación de Gandía y la API) interactúen. En el caso del *parser* descrito en el punto 9.2.1 ya se ha explicado el ejemplo de la API tipo SAX. En la aplicación de Gandía se hace uso de una segunda API. Esta API es la de Google Maps (mapas de Google). Los mapas son una función indispensable para aplicaciones turísticas que permiten al usuario moverse por un lugar desconocido. Las aplicaciones de mapas y localización se encuentran en el segundo lugar de las aplicaciones descargadas para smartphones.

La integración de la API de Google Maps es un poco más fácil que la del SAX Parser. La API permite integrar un simple mapa en la *activity* de la aplicación. Para que el uso de este mapa sea más interesante, se añadieron varias funciones adicionales que se describen más adelante.

Para poder usar los mapas de Google hace falta usar una clase *MapView* para la que hace falta una librería externa. Mediante el uso de diferentes métodos se puede manipular o personalizar la información mostrada en los mapas. Antes de comenzar a trabajar con datos de los mapas de Google, es necesario aceptar las condiciones de uso, poseer una cuenta de Google y registrarse para recibir un código de identificación que se incluye luego en el código fuente. Este código se llama "Maps API Key" [14] y está encriptado en MD5 (Message-Digest Algorithm 5). Este es un algoritmo de reducción de 128 bits que forma un "hash" (función o método para generar claves) de 32 dígitos decimales. Las palabras "proyecto final de carrera" en MD5 darían como resultado el código: e1454e5f7218fb2d3cd46d41a4d33718.

Una vez obtenido el código de Google, comienza la parte de programación. El fichero *layout* contiene pocas líneas. En éstas se indica la *id* del *mapview*, el tamaño de la pantalla (pantalla completa), que la pantalla permite clicar sobre el mapa y como último la clave *apiKey* (aquí no se publica la *apiKey* original por motivos de seguridad).

```
<?xml version="1.0" encoding="utf-8"?>

<com.google.android.maps.MapView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/mapview"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:clickable="true"
  android:apiKey=" e1454e5f7218fb2d3cd46d41a4d33718 "
/>
```

En el fichero Java se hace referencia al *layout* anterior y se indica que el mapa a mostrar en pantalla es el de la *id* "mapview".

```
MapView mapView = (MapView) findViewById(R.id.mapview);
```

[14] GOOGLE CODE

Estas pocas líneas de código hacen aparecer un mapa de Google Maps en la pantalla. Este mapa, con las configuraciones básicas, no es muy cómodo para encontrar determinados lugares en Gandía ya que no está centrado sobre la ciudad (aparecerá posiblemente Estados Unidos), el mapa no contiene información turística y no se puede cambiar la vista (mapa o satélite).

Para que la aplicación sea útil para turistas, se añaden las funciones explicadas anteriormente. Para ello es necesario habilitar un menú a través del cual se accede a una pantalla dónde escoger las preferencias. En esta pantalla el usuario podrá personalizar su mapa. En esta aplicación es posible cambiar entre la vista normal (mapa clásico) y la vista de satélite. Además el usuario puede elegir la información que aparece sobre el mapa (sólo mapa, hoteles, lugares de interés, emergencias, etc.). Por último la aplicación permite al usuario mostrar su posición actual mediante el dispositivo GPS. Esta información aparecerá como punto azul parpadeando sobre el mapa. La aplicación guarda automáticamente la configuración del usuario. Si se cierra y vuelve a arrancar la aplicación (o la *activity* del mapa), aparece el mapa con la configuración establecida por el usuario.

Para conseguir este funcionamiento hace falta: habilitar el botón de menú y crear el menú, crear un fichero tipo "preference_screen" para tomar ajustes del mapa y cambiar las preferencias, darle funciones de zoom y centrar el mapa sobre Gandía y por último geoposicionar los lugares de interés. Como valor añadido al geoposicionamiento de los lugares de interés, se configuran éstos puntos como botones. Al pulsar sobre un lugar, se abre una ventana emergente con información adicional y dos botones. El primero permite obtener más información sobre este lugar y el segundo permite cerrar la ventana emergente.

La primera tarea es la habilitación del menú. El menú tiene tres opciones: Atrás (para volver a la lista principal), Salir (para cerrar la aplicación) y Opciones (para tomar los ajustes preferidos).



Imagen 9.3 Mapa con menú

Mediante las tres líneas siguientes, al principio del código fuente, se declaran las tres variables del menú.

```
private static final int MENU1 = Menu.FIRST;
private static final int MENU2 = Menu.FIRST + 1;
private static final int MENU3 = Menu.FIRST + 2;
```

Después de la función principal de la *activity* se crea una función llamada *onCreateOptionsMenu* en la que se indica el texto que aparece en los tres botones del menú. En este caso los textos (Atrás, Opciones y Salir) se encuentran en el fichero strings.xml dentro de las variables "menu_atrasar", "menu_filtro" y "menu_salir".

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
```



```

    menu.add(Menu.NONE, MENU1, 0, R.string.menu_atrasar);
    menu.add(Menu.NONE, MENU2, 1, R.string.menu_filtro);
    menu.add(Menu.NONE, MENU3, 2, R.string.menu_salir);
    return true;
}

```

Para que ocurra un evento al pulsar sobre uno de los botones de éste menú se crea otra función adicional. La función *onMenuItemSelected* detecta cual de los tres botones es presionada por el usuario. Para ello hace uso de un *switch case*. Al presionarse el botón “Atrasar” (Menu1) se activa un intent que llama a la *activity* lista_monumentos.java. En el caso de pulsar sobre salir, se da por finalizado la aplicación. La opción más interesante desde el punto de vista del código y del funcionamiento es la de “Opciones”. En este caso, el intent2 llama a la *activity* preferencias_mapa.java. Su funcionamiento se describe más abajo.

```

@Override
public boolean onOptionsItemSelected(int featureId, MenuItem item) {
    super.onOptionsItemSelected(featureId, item);
    switch (item.getItemId()) {

        case MENU1://atrasar
        // f_atrasar();
        Intent intent = new
        Intent(mapa_monumento.this, lista_monumentos.class);
        startActivity(intent);
        finish();
        break;
        case MENU2: //filtro
        Intent intent2 = new
        Intent(mapa_monumento.this, preferencias_mapa.class);
        startActivity(intent2);
        finish();
        break;
        case MENU3: //salir
        setResult(RESULT_OK);
        finish();
        break;
    }
    return true;
}

```

9.2.2.a Preferences

La *activity* a la que se accede al pulsar sobre el botón de opciones es un poco peculiar y diferente a las vistas hasta ahora. Esta *activity* usa controles como el *CheckBox* (permite habilitar y deshabilitar una opción) y un *ListPreference* (permite seleccionar un elemento de una lista). Estos controles sirven en la aplicación de Gandía para cambiar entre la vista de mapa y de satélite, para habilitar el GPS y para seleccionar la información que se quiera ver sobre el mapa.

La primera parte de este código consiste en el fichero java que hace referencia a un fichero *.xml para el cual se tiene que crear una carpeta adicional dentro de la carpeta “res”. La nueva

carpeta se llama también "xml". La referencia dentro de la *PreferenceActivity* se hace con: `addPreferencesFromResource (R.xml.preferencias_mapa) ;`

El fichero xml es parecido a los ficheros de *layout* normales. En ellos se indica qué elementos se usan y en qué lugar de la pantalla se posicionan. El siguiente ejemplo muestra la configuración para un *CheckBox* (para cambiar entre la vista de mapa y la vista satélite). El *key* equivale a una *id*. *Summary* y *title* son campos opcionales en los que se puede añadir una descripción para que el usuario sepa los cambios que puede realizar. Por último en *defaultValue* se fija el valor por defecto del *CheckBox*.

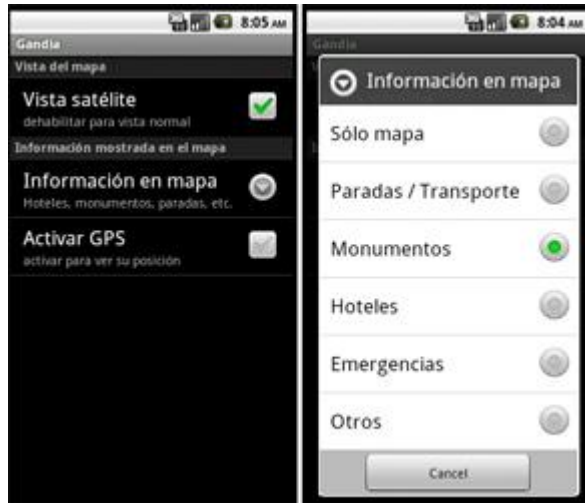


Imagen 9.4 Preferences y ListPreference

```
<CheckBoxPreference
    android:key="vistamapa"
    android:summary="dehabilitar para vista normal"
    android:title="Vista satélite"
    android:defaultValue="false"
/>
```

La configuración del *ListPreference* es parecida. En este caso, al pulsar sobre el símbolo correspondiente, se abre una ventana emergente en la pantalla. Ésta permite seleccionar un elemento de una lista. Pulsando sobre el elemento se activa un *RadioButton* de color verde como se observa en la parte derecha de la imagen 9.4. A parte del código *layout* de configuración siguiente, es necesario crear fichero xml nuevo.

```
<ListPreference
    android:key="infomapa"
    android:title="Información en mapa"
    android:summary="Hoteles, monumentos, paradas, etc."
    android:defaultValue="1000"
    android:entries="@array/mostraropciones"
    android:entryValues="@array/valoresopciones"
/>
```

El nuevo fichero se llama *arrays.xml*. Un *array* es una matriz o vector que permite guardar elementos del mismo tipo [15]. Un *array* es una zona de almacenamiento continuo y comienza siempre en la posición 0. En un *array* de tamaño 10, el primer elemento se encuentra en la posición 0 y el último en la posición 9. Un *string* (visto anteriormente en este proyecto), es pues un *Array* de elementos de tipo *String*.

El fichero *arrays.xml* contiene los valores de la lista de la información a mostrar en el mapa.

[15] WIKIPEDIA

```

<string-array name="mostraropciones">
  <item name="a">Sólo mapa</item>
  <item name="b">Paradas / Transporte</item>
  <item name="c">Monumentos</item>
  <item name="d">Hoteles</item>
  <item name="e">Emergencias</item>
  <item name="f">Otros</item>
</string-array>

```

Cuando el usuario realiza ajustes en la pantalla de preferencias, estos ajustes quedan guardados en el smartphone. En la programación en Java para Android el programador no se tiene que preocupar del estado de la memoria. Android internamente gestiona la memoria y guarda la información necesaria en el lugar correspondiente.

Una vez aclarado el uso y funcionamiento de la *activity* de preferencias se puede explicar el uso de los datos allí guardados en la activity del mapa (mapa_gandia.java).

9.2.2.b Recuperar datos de SharedPreferences

La primera tarea a realizar en la *activity* del mapa es la de leer los datos referentes a la configuración guardados por el usuario para mostrar la información correcta en la pantalla.

En primer lugar se leen las variables correspondientes a la vista del mapa y al dispositivo de GPS. A continuación la aplicación comprueba si el usuario prefiere la vista de satélite o vista normal (streetView). Dependiendo de la preferencia se activa (*true*) o desactiva (*false*) la opción correspondiente. A continuación la aplicación lee el valor guardado en "infomapa" para averiguar la información que se tiene que mostrar en la pantalla (hoteles, lugares de interés, etc.).



Imagen 9.5 Vista satélite

```

//cambiar entre vista satélite y mapa
  SharedPreferences app_preferences =
  PreferenceManager.getDefaultSharedPreferences(this); //recupera valor
  guardado anteriormente
  vista_mapa = app_preferences.getBoolean("vistamapa",
  false);
  gps_activo = app_preferences.getBoolean("posicion", false);

  if(vista_mapa==false) {
    mapView.setSatellite(false);
    mapView.setStreetView(true);
  }
  else{
    mapView.setStreetView(false);
    mapView.setSatellite(true);
  }

  SharedPreferences sharedPrefs =

```

```

PreferenceManager.getDefaultSharedPreferences(this);
StringBuilder builder = new StringBuilder();
builder.append(sharedPrefs.getString("infomapa", "1"));
String abc= builder.toString();
int in = Integer.valueOf(abc.toString());

```

En el último entero (int in) se guarda un número que identifica la información a mostrar en el mapa.

Por defecto el mapa mostrado por Google Maps puede estar centrado en algún lugar de Estados Unidos. En esta aplicación de Gandía es preferible que la aplicación muestre el municipio de Gandía para que el usuario no tenga que buscar la ciudad en un mapa mundial. En primer lugar Android tiene que centrar el mapa sobre Gandía (para ello se indican las coordenadas) y a continuación se realiza un zoom (se aumenta el mapa hasta que se vea bien la ciudad de Gandía).

```

//centrar posición inicial del mapa aquí y hacer zoom
mapView.setBuiltInZoomControls(true);
mc = mapView.getController();
GeoPoint pointInicial = geoAdapter(38.967918,-0.184793);
mc.animateTo(pointInicial);
mc.setZoom(16);

```

Con el código explicado hasta este momento, la aplicación mostrará el mapa de Gandía con vista normal o satélite y sabrá la información que tiene que ser mostrada. Esta información se encuentra debajo del código fuente anterior. La información respecto a los hoteles, lugares de interés, etc. realmente no se muestra en el mismo mapa, sino que se “pinta” en una nueva capa (*overlay*, que se encuentra encima del mapa). Seguido a la configuración de esta capa *Overlay* y el *drawable* (que hace referencia también a un nuevo fichero tipo *ItemizedOverlay*) sigue una lista con toda la información de los puntos de interés. Esta información contiene las coordenadas de GPS para el geoposicionamiento, el título del lugar y una segunda línea de descripción. Las coordenadas GPS se guardan en una variable tipo *GeoPoint* llamadas *point*, *point2*, *point3*, etc. y los correspondientes elementos que contienen la información son de tipo *OverlayItem*.

```

List<Overlay> mapOverlays = mapView.getOverlays();
Drawable drawable =
this.getResources().getDrawable(R.drawable.pin);
GandiaItemizedOverlay itemizedoverlay = new
GandiaItemizedOverlay(drawable, this);

GeoPoint point = geoAdapter(38.968548,-0.181081);
OverlayItem overlayitem = new OverlayItem(point, "Antigua
Universidad", "Escuelas Pías");

GeoPoint point2 = geoAdapter(38.965932,-0.180094);
OverlayItem overlayitem2 = new OverlayItem(point2,
"Palacio Ducal", "Palacio de los Borja");
[...]

```

En la lista anterior se encuentran todos los lugares de interés, hoteles, etc. Para mostrar estos lugares sobre el mapa se usa la función `addOverlay` de la siguiente forma:
`itemizedoverlay.addOverlay(overlayitem); //Antigua Universidad`

Al principio del fichero Java, la aplicación lee los datos de configuración guardados en el dispositivo. En la variable `int in`, se guarda un número determinado dependiendo si el usuario prefiere ver los monumentos u otra cosa sobre el mapa. Para que la aplicación muestre sólo el contenido preferido por el usuario, se compara el valor de “in” con varios números y se muestra sólo ese contenido.

Si el usuario prefiere ver la información de los monumentos, “in” tomará el valor 3. Cuando se ejecuta la aplicación y se llegue a la siguiente línea, el resultado `in==3` será verdadero con lo que se mostrarán sólo los elementos que se encuentren entre estos corchetes (sólo los monumentos).

```
else if(in == 3){  
    //monumentos  
    itemizedoverlay.addOverlay(overlayitem);  
    itemizedoverlay.addOverlay(overlayitem2);  
    itemizedoverlay.addOverlay(overlayitem3);  
    [...]
```

Los códigos hasta aquí hacen que se muestre un icono (en este caso una chincheta) en cada posición de GPS pero no especifican lo que se encuentra en éste lugar (la aplicación lo sabe, pero el usuario no lo ve). Para ofrecer al usuario información sobre los lugares representados sobre el mapa hay que convertirlos en “botones”. Al pulsar sobre uno de los iconos se tiene que abrir una ventana emergente con información y enlaces a otras actividades. Esto se hace mediante el `ItemizedOverlay`.

9.2.2.c ItemizedOverlay

El `ItemizedOverlay` controla la ventana emergente (`AlertDialog`).

La información que aparece en esta ventana es el título y la descripción de un lugar de interés como visto en el punto anterior. Adicionalmente hay dos botones, uno para cerrar la ventana emergente y otro para acceder a más información del lugar de interés seleccionado. Para que se establezca el enlace correspondiente, el código fuente de la ventana emergente tiene que leer el título del lugar o monumento y compararlo con una lista guardada en la misma aplicación. Seguido a esto se le asigna un valor determinado a una variable para identificarla más adelante.

```
else if(item.getTitle().equals("El Torreón")) {  
    b=8;  
}
```



Imagen 9.6 Más Información

En el código anterior se observa que se compara (`.equals`) el título (`item.getTitle`) con las palabras “El Torreón”. Si se cumple ese requisito, la variable “b” obtendrá un valor determinado.

El usuario tiene dos opciones, la de cerrar la ventana o la de abrir la *activity* con la información adicional. Pulsando el botón de cerrar (botón negativo) se ejecuta el siguiente código:

```
dialog.setNegativeButton("Cerrar", new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        dialog.dismiss();
    }
});
```

En el caso del botón “Más información” (“botón positivo”) se ejecuta el código:

```
dialog.setPositiveButton("Más Información", new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {

        if(b<=20){
            Intent intent = new Intent(mContext, monumentos.class);
            intent.putExtra("monumento", (int)b);
            mContext.startActivity(intent);
        }
        [...]
    }
});
```

Dentro de ésta función se lee la variable “b” guardada anteriormente (con el valor que identifica un lugar determinado) y se crea el enlace correspondiente. De esta forma el botón “Más Información” de los hoteles, lleva a la *activity* informativa de los hoteles, mientras que el botón de la ventana del Torreón de Gandía ejecutará la *activity* de los lugares de interés.

9.2.2.d Brújula

Como último punto de la *activity* del mapa de Gandía, fue necesario añadir una brújula para permitir al usuario girar el mapa hacia el norte y poder orientarse de ésta forma más fácil. La brújula aparece en la parte superior derecha del mapa de GoogleMaps. Para enfocar el mapa hacia norte hace falta girar el smartphone hasta que la aguja de la brújula muestre hacia la parte superior del mapa (el mapa es estático, se gira sólo la brújula). Para la brújula hace falta, igual como en los casos de los lugares de interés, una capa (*overlay*) sobre la que se “dibuja” la información. La brújula se activa mediante la función `enableCompass()`;

Actualmente todos los smartphones llevan una brújula digital incorporada. Para el uso de la brújula y del sensor de GPS es necesario otorgarle a la aplicación los permisos necesarios dentro del fichero `AndroidManifest.xml`.



Imagen 9.7 Brújula dentro del fichero

Como explicado en el punto 9.1.3 son necesarios para la localización del dispositivo los siguientes permisos:

```
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Con el código explicado a lo largo del punto 9.2.2 y varias líneas adicionales se solucionaron algunos de los objetivos propuestos para este proyecto que fueron el uso de una API, gravar y recuperar datos de la memoria del smartphone, el control del sensor GPS y la lectura de datos de la brújula digital.

9.2.3 Componentes hardware

Los smartphones disponen de varios sensores y otros componentes electrónicos que ofrecen un uso muy amplio del teléfono. En este proyecto ya se habló del uso de la pantalla táctil y su funcionamiento, de los botones (hardkeys) para atrasar o activar el menú, del dispositivo GPS y de la brújula electrónica. Otro sensor utilizado en la aplicación de Gandía es el sensor de aceleración. Mediante este sensor es posible calcular la inclinación del móvil sobre sus tres ejes (x,y,z). Para esta aplicación es suficiente saber si el móvil se encuentra en posición horizontal (la habitual) o en vertical. Dependiendo de su posición se puede ajustar el diseño del layout. Esto se hace por ejemplo en la pantalla inicial (splashscreen) con la imagen de la playa de Gandía, en la *activity* de la galería de fotos y la *activity* de los lugares de interés.

Si se usa el sensor de aceleración sólo para ajustar el diseño de la pantalla es necesario crear una nueva carpeta llamada "layout-land" en la que se guardan todas los ficheros layout tipo xml con el diseño para las pantallas horizontales (con el mismo nombre que los ficheros para las pantallas verticales). En el caso de la *activity* gallery.java (código para mostrar fotos de Gandía), el paquete de la aplicación tiene una carpeta "layout" y otra carpeta "layout-land" que contienen ficheros gallery.xml distintos.

La idea de utilizar estos dos ficheros xml es la de poder ajustar el diseño de la pantalla para ambos casos. Android permite cambiar la estructura del diseño, los tamaños de los elementos, colores, etc. El programador tiene gran libertad para variar ambos diseños. La única restricción existente es que obligatoriamente hace falta usar todos los elementos que se usaron en el fichero Java. Es decir, si en el código Java se hace uso de un determinado *TextView* por ejemplo, éste tiene que ser usado tanto en la vista horizontal como en la vista vertical. En el caso de no hacerlo, la aplicación devuelve un error y se cierra.



Imagen 9.8 Vista horizontal y vertical del splashscreen

Dependiendo de la posición del móvil (vertical u horizontal), se muestra una imagen u otra al aplicarse el diseño de diferentes layouts.

La diferencia entre ambos ficheros xml en éste caso es que en el primero (vertical) el *imageView* con *id* "foto_inicio" accede a la imagen *gandiaportada.jpg* (mediante este código : `android:src="@drawable/gandiaportada"`) mientras que en el otro caso (horizontal) accede a la imagen *gandiaportadaland.jpg* (mediante `android:src="@drawable/gandiaportadaland"`).

En el caso de la *activity* de los monumentos o lugares de interés en vez de cambiar una imagen, lo que se cambia es el orden de mostrar los elementos. En la vista vertical se muestra primero la fotografía, luego el título mientras que en la vista horizontal es al revés. También se cambia el orden de los botones de esta *activity*. Estos cambios se realizan todos en los dos ficheros *layout*.



Imagen 9.9 Vista horizontal y vertical de la activity "Monumentos"

Ambas imágenes muestran información de dos monumentos distintos, pero lo importante en este caso es el orden de los tres últimos elementos (un *textView* y dos botones). En el caso de

la vista horizontal el orden es: *textView*, botón, botón como se ve en el trozo de código siguiente.

```
<TextView
    android:id="@+id/contacto"
    android:layout_below="@id/informacion"
    [...]
/>
<Button
    android:id="@+id/boton_localiza"
    android:layout_below="@id/contacto"
    [...]
/>
<Button
    android:id="@+id/boton_masinfo"
    android:layout_below="@id/boton_localiza"
    [...]
/>
```

En el caso de la vista vertical el orden es: botón, *textView*, botón y se uso el siguiente código.

```
<Button
    android:id="@+id/boton_masinfo"
    android:layout_below="@id/informacion"
    [...]
/>
<TextView
    android:id="@+id/contacto"
    android:layout_below="@id/boton_masinfo"
    [...]
/>
<Button
    android:id="@+id/boton_localiza"
    android:layout_below="@id/contacto"
    [...]
/>
```

En el caso de ésta aplicación o de ésta *activity* no hubiera hecho falta realizar dos diseños diferentes. Pero en esta parte del proyecto se quiso demostrar las posibilidades que tiene el programador a la hora de crear una nueva aplicación. Una vez esté programada un diseño, no es difícil modificarlo para conseguir un segundo diseño, pero es importante conocer que existe esa posibilidad y saber cómo realizarlo.

El último hardware al que accede la aplicación de Gandía, es el circuito electrónico del mismo teléfono. Es decir, se aprovecha la electrónica que permite al smartphone a realizar llamadas telefónicas. Aunque la tarea principal de los teléfonos debería ser el de realizar y recibir llamadas, los smartphones son usados más tiempo para usar aplicaciones y navegar por Internet que para llamar. La aplicación de Gandía le permite al usuario a realizar llamadas desde dentro de la misma aplicación. De esta manera no hace falta salir de la aplicación, buscar un teléfono de interés de Gandía y marcarlo. Al pinchar sobre el elemento "teléfonos" dentro de la lista principal, se ejecuta la *activity* "teléfonos" con la lista de teléfonos de interés entre los que se encuentran Información Turística de Gandía, Ayuntamiento, Información RENFE, Policía, Hospital, etc. Automáticamente aparece en la barra superior del sistema

Android un icono verde con un símbolo de un teléfono como muestra la imagen 9.10. En la parte derecha de esta imagen se ve además la pantalla que aparece al llamar a un teléfono. Al sacar la imagen para este proyecto se “llamó” con el emulador a un número de teléfono imaginario, de allí el número 67563.

En esta activity se encuentra una lista (*array*) con los nombres (lugares) a los que el usuario puede llamar.

Al pulsar sobre un nombre (por ejemplo “Información Turística Playa”), la función *onListItemClick* averigua la posición en la lista del elemento sobre el que el usuario pulsó y a continuación mediante un switch se le asigna un valor a un *string*. Este string contendrá tanto caracteres como números. En el caso de Información Turística de la Playa de Gandía, el valor del *string* será: “tel:962842407”. Esta es la forma correcta para indicarle a Android que se trata de un número de teléfono.



Imagen 9.10 Lista telefónica y pantalla para llamar

Una vez la *activity* haya guardado el valor correcto del *string* correspondiente al elemento pulsado en la lista, se llama a la activity *llamar.java*.

La llamada se realiza mediante un *intent*. En este caso se añade al intent el *string* guardado anteriormente para que sea enviado desde la *activity* “telefonos” a la activity “llamar”. Esto se realiza mediante el siguiente código:

```
Intent intent = new Intent(telefonos.this, llamar.class);
    intent.putExtra("telefono", numeroLlamada);
    startActivity(intent);
    finish();
```

Después de declarar el *Intent* y antes de enviarlo mediante *startActivity*, se añade el *String* “numeroLlamada”. Esto se realiza mediante la función “putExtra” a la que se le envía dos valores. El primero es el nombre bajo el que se “guarda” o envía el dato (en este caso “telefono”) y el segundo es el nombre de la variable (numeroLlamada) que contiene el dato de interés. La siguiente activity arranca, la actual se cierra mediante la función *finish()* y el dato es enviado.

Tras arrancar la *activity* “llamar”, la primera tarea es la de recibir el dato (*string*) enviado previamente. Para ello hacen falta las siguientes líneas de código:

```
Bundle extras = getIntent().getExtras();
    if(extras!=null){
        cadenaRecibida = extras.getString("telefono");
        llamar(cadenaRecibida);
    }
```

Para enviar el dato hizo falta la función *putExtra* y para recibir los datos se usa la función *getExtras*. Con esto se reciben todos los datos enviados (en el caso de que se hubiesen enviado varios). La segunda línea de código verifica que el contenido recibido no está vacío. En el caso de ser vacío (no contener ningún valor), la condición no se cumple y no se realiza ninguna operación. Si en cambio se ha recibido un dato (el valor de los datos es diferente a “nada” o “cero”) se cumple la condición y se ejecutan las dos líneas de código siguientes. Dentro de la condición *if*, se encuentran dos cosas importantes. Los símbolos != indican una desigualdad (lo contrario a .equals en Java o == en C++). La palabra reservada *NULL* (nulo) * indica la ausencia de un dato o valor.

Al cumplirse la condición anterior se guarda el *string* recibido en el *string* “cadenaRecibida”. Éste es enviado a la función “llamar” mediante: `llamar(cadenaRecibida)`;

Dentro de ésta función se interpreta el *string* recibido y se realiza otro *Intent* que realiza la llamada telefónica (`callIntent`).

```
Intent callIntent = new Intent (Intent.ACTION_CALL);  
    callIntent.setData(Uri.parse(llamarNumero));  
    startActivity(callIntent);
```

Desde el punto de vista del sistema operativo Andorid, basado en Linux, una llamada telefónica puede ser un “peligro”. En primer lugar una llamada telefónica puede generar unos gastos económicos (el caso normal). Un programador con malas intenciones podría realizar una llamada sin que el usuario del teléfono se diese cuenta y escuchar de esta forma las conversaciones que se realizan cerca del teléfono afectado (como un micrófono espía).

Para evitar efectos indeseados, hace falta otorgarle a la aplicación que haga uso de llamadas telefónicas el permiso correspondiente. Al instalar la aplicación el usuario es avisado de los permisos que acepta. El código para éste permiso del *AndroidManifest* es el siguiente:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

* *NULL*: Se trata de un valor especial, sobre todo en punteros, que indica que el lugar apuntado no existe o no es correcto. Un funcionamiento parecido lo tiene la palabra *NIL*. Este concepto aparece cuando se hace referencia a algo que no existe. Un chiste de informáticos dice que la abreviación significa “No Intentes Lograrlo” (al apuntar a algo que no existe).

9.2.4 Funciones

Durante el proceso de programación de esta aplicación Android se usaron muchas funciones que ofrece el lenguaje Java. Para algunas tareas puntuales de la aplicación fue necesario crear funciones propias. Esto facilita el proceso de programación, la lectura del código fuente, tomar ajustes y cambios del código de forma rápida y una ejecución más eficiente. Tareas frecuentes se pueden llamar mediante funciones y ahorrar de esta forma líneas de código y reducir el tamaño total (en cuanto a líneas y Bytes).

9.2.4.a Funciones simples

La función más simple que se puede usar es, como explicado en el punto 4.3, aquella que no envía ni recibe nada. Esto es el caso de la función “mostrarMonumento” cuya estructura se usa en varias ocasiones durante este proyecto.

La función es llamada mediante: `mostrarMonumento()`;

Este código llama a la función que tiene el siguiente código en el que se le asigna unos valores a un `TextView` y un `ImageView`. La asignación de los valores se realiza también mediante dos funciones. En las llamadas a éstas dos funciones (`setText` y `setImageResource`) se envían además unos datos (las funciones reciben un string o imagen respectivamente).

```
public void mostrarMonumento ()
{
    CampoTitulo.setText (R.string.titulo_compras);
    CampoInformacion.setText (R.string.informacion_compras);
    CampoFiestas.setImageResource (R.drawable.compras);
}
```

9.2.4.b Envío y recepción de datos a /desde funciones

El envío de datos a una función (como visto en el caso de las funciones `setText` y `setImageResources`) es algo muy útil. En el caso de las dos funciones anteriores, existen ya librerías de Java que permiten su uso. En el caso de la *activity* del mapa de Gandía se hizo uso de una función para adaptar las coordenadas GPS a las necesidades de Java. Para ello se usó una función que no existe por defecto.

En este caso se convierten las coordenadas de GPS a coordenadas tipo *GeoPoint* con las que trabaja Android. La función que se encarga de ésta tarea se llama *geoAdapter*. Al llamar la función, se añaden los valores de latitud y longitud de un determinado lugar. Ésta información se escribe entre paréntesis detrás del nombre de la función. La estructura es la siguiente: `geoAdapter(valor de latitud, valor de longitud)`; Los valores de latitud y longitud son valores positivos y/o negativos con seis posiciones detrás de la coma. Desde el punto de vista informático se trata de un *double*.

Las coordenadas o valores con los que trabaja Android se llaman *GeoPoint* y son datos en micro-grados que equivale a “grado * 10⁻⁶”. Los valores de las coordenadas del *GeoPoint* son números enteros (tipo `int`). Para pasar las coordenadas con coma flotante (*double*) a valores enteros (*int*) en micro-grados, es necesario multiplicar los valores anteriores por 10⁶ y convertir el resultado a entero.

Como ejemplo se toma el número 0.05 guardado en un double llamado “decimal”. Para convertirlo en un entero con valor 5, se cambia el valor original de escala multiplicándolo por 100).

```
double numeroAntiguo=0.05;
int nuevoNumero = (int)(numeroAntiguo*100);
```

En el caso de las coordenadas se hace lo mismo pero multiplicando el valor por 10 elevado a 6.

El código completo es el siguiente:

```
GeoPoint pointInicial5 = geoAdapter(38.966033,-0.181596);

GeoPoint geoAdapter(double lat , double lon){
    GeoPoint temp= new GeoPoint((int)(lat * 1e6),(int)(lon * 1e6));
    return temp;
}
```

Cómo en éste caso a la función principal le hace falta el nuevo valor de las coordenadas (en micro-grados), la función *geoAdapter* se lo tiene que devolver. Para ellos se añade al final de la función la palabra reservada *return* y el nombre de la variable a devolver.

El resultado de la función (valor de retorno) es asignada a la variable *pointInicial5* de tipo *GeoPoint* de la siguiente forma:

```
GeoPoint pointInicial5 = geoAdapter(38.966033,-0.181596);
```

Mediante este ejemplo se ve cómo enviar uno o varios valores a una función y cómo recibir y asignar un valor de retorno.

De una forma similar es posible llamar una función (y enviarle unos datos) aunque dicha función se encuentre en otro fichero tal como se indicó al final del punto 9.2.1.

9.2.4.c Llamadas a una funciones en otros ficheros

En un fichero se escribe el código siguiente que envía la variable “*eltiempo*” a la función *traducir*. Para ello se escribe: *eltiempo = traduccionTiempo.traducir(eltiempo);*

El código de esta función se encuentra en el fichero “*traducciónTiempo*” y es el siguiente:

```
public class traduccionTiempo {
    public static String traducir(String eltiempo){
        if (eltiempo.equals("Partly Cloudy")){
            eltiempo="Parcialmente nublado";
        }
        else if (eltiempo.equals("Sunny")){
            eltiempo="Soleado";
        }
        [...]
    }
}
```

Uso de funciones para ahorrar espacio:
En la aplicación de Gandía hay una lista de más de 10 lugares de interés. Para mostrarlos hubiera sido posible realizar una activity para cada lugar. De esta manera se obtendría diez activities con códigos fuente muy similares. Como ejemplo del uso eficiente de funciones se decidió crear una sola activity que recibe un valor (identificativo para un lugar de interés en concreto) y escoge la información correspondiente y la muestra en pantalla.

La *activity* monumentos.java cuenta en su función general con varias llamadas a funciones secundarias. Las funciones *localiza*, *masInfo* y *mostrarMonumento* que se describen a continuación ayudan a mantener el código legible.

La función *mostrarMonumento* muestra la información turística que se muestra en la pantalla del smartphone. Entre esta información se encuentra el título del lugar de interés, la imagen, texto informativo y la información de contacto u horarios. La función es muy parecida a la función *monumentos* que se usa en otras activities de esta aplicación.

El comportamiento de los dos botones de la *activity* (mostrar el monumento sobre el mapa y mostrar más información de la página web turística de Gandía) se controla con las funciones *localiza* y *masInfo*.

Dependiendo del valor identificativo que tiene la variable "numeroMonumento", se crean enlaces diferentes desde los botones a las páginas o activities siguientes. Si por ejemplo la variable numeroMonumento obtiene el valor 0 significa que el usuario escogió el Palacio Ducal. Mediante los *switch-case* en ambas funciones se crea o bien un enlace con más información a la dirección web correspondiente (en la función *masInfo*) o se ejecuta el *intent* que abre la *activity* mapa_monumento y envía a dicha *activity* el valor identificativo del lugar de interés para que sólo se muestre este lugar sobre el mapa.

```
public void localiza() {
    switch (numeroMonumento) {
        case 0:
            Intent palacio = new
                Intent(monumentos.this, mapa_monumento.class);
            palacio.putExtra("Monumento", (int) numeroMonumento);
            startActivity(palacio);
            finish();
            break;
        case 1:
            Intent colegiata = new
                Intent(monumentos.this, mapa_monumento.class);
            colegiata.putExtra("Monumento", (int) numeroMonumento);
            startActivity(colegiata);
            finish();
            break;
        [...]
    }
}
```

```
public void masInfo() {
    switch (numeroMonumento) {
        case 0:
            Intent palacio = new
                Intent(monumentos.this, monumentoWeb.class);
            [...]
    }
}
```

```

        palacio.putExtra("Monumento", (int) numeroMonumento);
        startActivity(palacio);
        finish();
        break;
    case 1:
        Intent colegiata = new
        Intent(monumentos.this, monumentoWeb.class);
        colegiata.putExtra("Monumento", (int) numeroMonumento);
        startActivity(colegiata);
        finish();
        break;
    [...]

```

La *activity* monumento engloba de esta forma más de cuatrocientas líneas de código Java. Debido al uso de las funciones su estructura y código es más legible que si se hubiesen programado 10 *activities* de 200 líneas de código. Otra ventaja es que a la hora de cambiar algo sobre el código fuente, en el caso actual sólo hay que cambiar el código en un fichero y no en 10 diferentes. A medida que el número de *activities* aumenta se entiende la necesidad esta forma de programar.

9.2.4.d Uso de funciones estándar

Además de las funciones creadas para esta aplicación se usaron muchas funciones ya existentes. En este apartado se introducen brevemente algunas de ellas.

La función *equals* equivale en Java a dos símbolos iguales (==) en C++. Se usan para comparaciones de dos parámetros. Ejemplo: `localName.equals("condition")`

La función que se encarga de encontrar en una lista el elemento sobre el que pulsó el usuario es *onListItemClick*. La función devuelve un valor entero (llamado *position*) con la posición del elemento seleccionado en la lista. Al tratarse de un *array* el primer elemento tendrá el valor 0.

```

public void onListItemClick(
    ListView parent, View v,
    int position, long id)
{ [...]

```

En todas las *activities* de la aplicación de Gandía se usa el botón (hard-key) de retroceso para llegar a otra *activity* de nivel superior. Esto se realiza mediante la función *onBackPressed*. Dentro de la función sólo queda añadir el código de la tarea que tiene que realizar la aplicación en el momento de la pulsación.

```

public void onBackPressed() {
    setResult(RESULT_OK);
    finish();
    return;
}

```

Mediante la función *valueOf* es posible convertir un *string* (de números) a un entero. Se tiene por ejemplo un *string* llamado *abc* con la cadena de texto 12. En este caso 12 no es el número doce sino una cadena de caracteres “no numéricos”.

Como aclaración: `String` cadena = "12"; e `int` numero="12"; Pero cadena != numero (cadena es diferente a número). Una comparación de la forma `if(cadena == numero){ [...]}` daría lugar a un error de todas formas al no estar permitida ni posible la comparación entre variables de diferentes tipos.

Para la realización de cálculos por ejemplo, se tiene que pasar esa cadena de caracteres a un número entero, 12 en ese caso. Esta conversión se realiza mediante la función `valueOf` en la que se especifica que "abc" es de tipo `string` y que el resultado devuelto es tipo `integer` (adicionalmente en este ejemplo se guarda en un `int` llamado "in").

```
int in = Integer.valueOf(abc.toString());
```

Otra función muy usada en Android es la función `startActivity`. Ésta se usa para llamar actividades nuevas. A esa función se le envía además un `intent` que contiene la información de la `activity` actual y el nombre de la `activity` que se tiene que abrir.

```
startActivity(intent);
```

Estas son unas de las funciones más usadas y que se deben comentar en este lugar. Para buscar información sobre otras funciones que pueden ser de ayuda, el programador puede encontrar más información sobre su uso y características en la página web oficial de desarrolladores de Android (developer.android.com) en la pestaña "Reference".

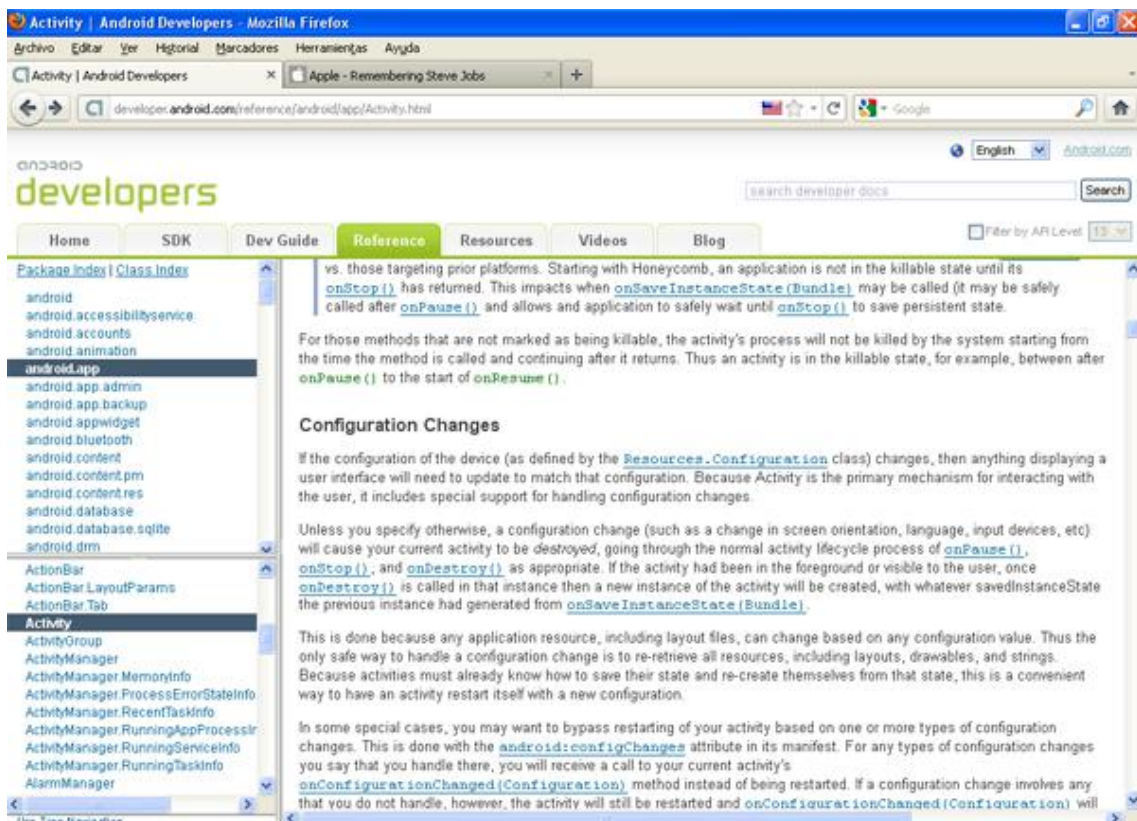


Imagen 9.11 Información para desarrolladores

9.3 Otros códigos

En los siguientes subapartados se encuentra una descripción de fragmentos de códigos de actividades que no se comentaron en los apartados anteriores al no encontrarse entre los objetivos secundarios de este proyecto.

9.3.1 ListView personalizado

Listviews son elementos de actividades que permiten mostrar listas en la pantalla del smartphone. El usuario puede seleccionar elementos de la lista pulsando encima de ellos. Los *listviews* tienen por defecto un fondo negro y letra de color gris. Cambiar estos colores para personalizar el *listview*

no es tan fácil como en otros *views*. Debido a esto la mayoría de aplicaciones en usar *listviews* los usa con los colores estándar. En la aplicación de Gandía se decidió cambiar estos colores por dos razones. En primer lugar es interesante conocer los pasos necesarios (creación de un nuevo fichero y los códigos fuentes necesarios) y en segundo lugar la aplicación da una imagen general más agradable y limpia.



Imagen 9.12 List view original y personalizado

En la aplicación de Gandía se usan dos *listviews*, uno muestra la lista principal (lista que aparece después de arrancar la aplicación) y la segunda contiene la lista de los lugares de interés de Gandía. Al tener ambas actividades un comportamiento similar, se explica en este punto sólo es código de la *activity* de la lista principal.

En primer lugar, cómo en todas las actividades, se fija el diseño del fichero xml. En éste caso el fichero contendrá un *LinearLayout* de orientación vertical y dentro de éste elemento un *ListView*. Este *listview* cuenta con los siguientes parámetros:

```
<ListView android:id="@+id/android:list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:divider="#D7E9F8"
    android:dividerHeight="1sp"
/>
```

Los cambios de color del texto y del fondo en este lugar no dan el resultado esperado por lo que no se incluye en este lugar. El único efecto que el programador consigue en éste punto es el color y grosor del “divider”. El divider es la línea divisora entre dos elementos de la lista que en este caso tiene un color azulado muy claro (casi blanco).

El fichero Java comienza con la declaración de una variable “menuprincipal” tipo *string array* que incluye todos los elementos de la lista.

```
String[] menuprincipal = { "Mapa" , "El Tiempo" ,"Noticias", [...]};
```

A continuación se enlaza el fichero de diseño con esta *activity* (igual cómo se hace en todas las *activities*). `setContentView(R.layout.lista_principal);`

Antes de fijar el comportamiento del *listview* se llama a la función `setListAdapter` para personalizar el diseño del *listview*. Para ello hace falta indicar en que fichero se encuentra el código que describe o fija el diseño. En este caso se llama `list_black_text.xml`.

```
setListAdapter(new ArrayAdapter<String>(this,
    R.layout.list_black_text,R.id.list_content, menuprincipal));
```

Sin la llamada a la función `setListAdapter`, el *listview* aparecería con los colores estándar pero su funcionamiento sería el mismo por lo que primero se describen las siguientes líneas de código fuente del fichero Java antes de pasar al nuevo fichero de diseño `list_black_text.xml`

La siguiente función usada es `onListItemClick` la que devuelve la posición del elemento seleccionado en el *string* “menuprincipal”. Dependiendo del valor de la variable “position” y mediante un *switch-case* se activa la siguiente *activity* mediante el intent correspondiente.

```
public void onListItemClick(
    ListView parent, View v,
    int position, long id)
{
    switch (position){
        case 0:
            finish();
            Intent mapa = new Intent(lista_principal.this,
                mapa_gandia.class);
            startActivity(mapa);
            break;
        [...]
    }
```

El fichero de diseño del *ListView* (no de la *activity* completa) se llama en este caso `list_black_text.xml` y es usado tanto por la *activity* `lista_principal.java` cómo por la *activity* `lista_monumentos.java`.

El contenido de este nuevo fichero es un elemento de *LinearLayout* y un *TextView*. El fondo del *LinearLayout* (fondo de la lista) recibe el color blanco mediante: `android:background="#FFFFFF"`. Mientras que el color de texto del *textView* recibe el color negro con: `android:textColor="#000000"`

Con estos cambios, la lista aparece en los colores deseados. El problema viene que al deslizar con el dedo por la pantalla del teléfono y desplazarse la lista en vertical, se vuelve a

poner el color de fondo negro. Para evitar eso se incluyen dos líneas adicionales que hacen que el color de fondo se haga transparente. `android:cacheColorHint="#00000000">` Como se observa en el valor hexadecimal del código anterior, hay dos posiciones adicionales (normalmente el código de color tiene sólo seis posiciones). Al `textView` se le añade la misma línea con otro valor: `android:cacheColorHint="#ffffff00"`.

Con estos ajustes, al desplazarse por la pantalla, el fondo que ve el usuario será siempre blanco y la letra siempre negra.

9.3.2 Webviews

Android permite mostrar páginas webs en un navegador propio de Android o dentro de la misma aplicación. Para el segundo caso se utiliza la vista `WebView` que se fija en el fichero xml de la `activity`. En dispositivos móviles es preferible ver páginas webs adaptadas para móviles (URLs con dominio `.mobi`) debido al tamaño y a la resolución de las páginas. Esta aplicación accede tanto a webs tradicionales como a una web `*.mobi`. En la aplicación de Gandía la información básica está guardada en la misma aplicación (aplicación nativa, para una ejecución rápida) mientras que para adquirir información adicional es necesaria una conexión de Internet para acceder al contenido de varias páginas webs.



Imagen 9.13 `WebView` horizontal de `aplicaeinnovacion.com` con zoom inicial para mostrar el contenido importante

En el fichero `layout` simplemente se crea un `LinearLayout` y un `WebView` que ocupe la pantalla completa (`layout_width` y `layout_height` obtienen el valor `fill_parent`). La `id` del `WebView` en este caso es "Webview" que es llamado desde el código Java.

Para mostrar una web determinada hacen falta varias líneas de código. En primer lugar se declara la variable "Web" de tipo `WebView`. A continuación se incluye en la función principal las siguientes líneas.

```
Web = (WebView) findViewById( R.id.Webview );

Web.getSettings().setJavaScriptEnabled(true);
Web.getSettings().setSupportZoom(true);
Web.getSettings().setBuiltInZoomControls(true);
Web.setWebViewClient(new InsideWebViewClient());
Web.setInitialScale(50);
Web.loadUrl("http://www.gandiaturistica.com");
```

La primera línea asigna el *WebView* (con *id* "Webview") del fichero xml con el *WebView* "Web" del fichero actual. A continuación se toman todos los ajustes. Mediante *setJavaScriptEnabled* con valor *true* se activa o permite el uso de JavaScript. Muchas páginas webs tienen código activo de JavaScript y si no se activa, puede que algunas páginas webs no se muestren de la forma deseada o que no se comporten bien.



Imagen 9.14 *WebView* gandiaturistica.com izq. zoom inicial y botones de zoom, der. zoom realizado con gestos

Mediante las siguientes opciones de zoom se permite que el usuario puede realizar un zoom para ampliar o reducir la pantalla actual. Esto se puede hacer de dos formas. La primera es utilizando los botones de control que aparecen en la pantalla al pulsar sobre ella. La segunda forma es la más usada al ser más intuitiva y novedosa. Se trata del uso de "gestos". Las pantallas capacitivas son capaces de detectar pulsaciones múltiples lo que permite identificar gestos. Usando dos dedos al mismo tiempo y haciendo el gesto de abrir o cerrar unas "pinzas" permite realizar las acciones de zoom. En el punto 9.3.4 se explica el uso de otro gesto. Además la pantalla táctil permite con el dedo moverse por la página web en todas las direcciones. Como las páginas webs normales no están adaptadas para pantallas de móviles (relativamente pequeñas), si las pantallas ocupan la pantalla completa, la letra es tan pequeña que no se puede leer. Debido a eso en la aplicación de Gandía se realiza un zoom inicial del 50% para mostrar mejor el contenido de la página web. Mediante la función *loadUrl* se indica a Android la URL (dirección de página web) que tiene que abrirse.

Finalmente mediante la clase *InsideWebViewClient* y *shouldOverrideUrlLoading* se indica que todas las páginas y enlaces se tienen que abrir en esta *activity* (sobrescribir la pantalla anterior y no abrir otra ventana).

```
private class InsideWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url){
        Web.loadUrl(url);
        return true;
    }
}
```

En el caso de las webs con dominio “mobi” no hace falta realizar un zoom inicial ya que los tamaños de las webs están adaptadas para pantallas de móviles. En la aplicación de Gandía es el caso de los horarios de trenes de RENFE. Los usuarios de la aplicación pueden acceder de forma cómoda a un formulario de la web renfe.mobi. El formulario permite al usuario seleccionar el origen y destino de su viaje. Esta información hace que la aplicación de Gandía no sólo sea interesante para turistas, sino también para habitantes de la ciudad que viajen en tren a otras ciudades. El botón para acceder a este *WebView* se encuentra dentro de la categoría “transporte” de la lista principal.



Imagen 9.15 WebView con horarios de RENFE

9.3.3 Galería de imágenes

Desde la lista principal de la aplicación de Gandía se accede a una activity que contiene una galería de imágenes de Gandía para que el turista se pueda hacer una idea rápida de lo que le espera en el destino sin necesidad de leer textos largos. La galería muestra las imágenes en tamaños diferentes dependiendo de la posición del smartphone (horizontal y vertical) siendo la posición horizontal la recomendada. El uso de los ficheros xml en relación al sensor de aceleración se explicó en el punto 9.2.3.

Este apartado en cambio estudia el código fuente Java con el que se consigue el control del paso de las imágenes (como en una presentación) con el dedo.

Los dos ficheros xml contienen la información referente al diseño horizontal y vertical e incluyen entre otros elementos un “Gallery” dónde aparecen las imágenes.

```
<Gallery xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gallery"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
```

Para que aparezcan las imágenes como en una presentación de diapositivas es necesario crear un *array* un poco peculiar. Se trata de un array de enteros que corresponden a las ids de las imágenes.

```
private Integer[] mImageIds = {
    R.drawable.colegiata,
    R.drawable.castillo_bayren,
    R.drawable.torreon,
    R.drawable.antigua_universidad,
    R.drawable.ayuntamiento,
    R.drawable.alqueria_duc,
    [...]
```

Además se usa la función *setAdapter* y la clase *ImageAdapter*.

Al principio de este proyecto se vio que uno de los problemas para los programadores de Android es conseguir la compatibilidad con el mayor número posible de dispositivos. Los teléfonos móviles con sistema operativo Android, a diferencia del iPhone, son construidos por varios fabricantes. Estos teléfonos pueden tener hardwares diferentes así como diferentes pantallas. Para lograr una buena reproducción de imágenes, no es suficiente con mostrarlas en un tamaño determinado, sino que hace falta ajustarlas a los tamaños de los diferentes dispositivos.

En esta aplicación se resuelve el problema de la siguiente forma:

Después del código fuente que se encarga de mostrar las imágenes en el orden correcto y de las transiciones, antes de “dibujar” las imágenes en pantalla, la aplicación recibe del smartphone, a través de la función *getDefaultDisplay*, las dimensiones en pixeles de la pantalla.

Para ellos se crea una variable tipo *Display*. A continuación se guardan los datos de altura y anchura mediante las funciones *getWidth* y *getHeight* en dos variables con valores enteros.

```
Display display = getWindowManager().getDefaultDisplay();
int widthpx = display.getWidth();
int heightpx = display.getHeight();
```

A continuación, antes de mostrar las imágenes en pantalla, se configura el *ImageView*. Los dos parámetros interesantes para este apartado son los valores usados durante las llamadas a las funciones *setLayoutParams* y *setScaleType*.

```
i.setLayoutParams(new Gallery.LayoutParams(widthpx, heightpx));
i.setScaleType(ImageView.ScaleType.FIT_CENTER);
```

En la primera se usan los valores de las variables “widthpx” y “heightpx” obtenidos anteriormente. La misma operación se realiza en el caso de la imagen principal al arrancar la aplicación de Gandía (splashscreen). En este caso también es posible introducir unos valores fijos como por ejemplo: *LayoutParams(500, 300)*. De esta forma la imagen siempre tendría el valor de 500x300px lo que en este caso no se desea.

En la segunda función (*setScaleType*) se ajusta el tipo de escalado que se escoge para las imágenes. En este caso se usó *FIT_CENTER* para centrar la imagen. Otras options serial: *FIT_END*, *FIT_START*, *FIT_XY*, etc.



Imagen 9.16 Transición de imágenes (Colegiata/Castillo Bayren) en la galería de fotos

Para que se ejecute la *activity* de forma correcta se añaden varias librerías como la *content.res.TypedArray*, *widget.AdapterView*, *widget.AdapterView.OnItemClickListener*, *widget.BaseAdapter* o la *widget.Gallery*.

El resultado del código completo de esta *activity* es una galería de imágenes por la que el usuario puede moverse realizando gestos con sus dedos (para indicar la dirección en la que se tiene que mover la imagen) como se observa en la imagen 9.16.

9.3.4 Gestos (en la *activity* monumentos)

Para aprovechar el uso de la pantalla táctil en varias *activities*, se decidió incluir un nuevo “*touchevent*” en la *activity* “monumentos”. Al deslizarse sobre la barra superior de la *activity*, Android registrará la dirección y longitud del desplazamiento y mostrará la información del siguiente o anterior monumento.

Para realizar esto, la tarea consiste en calcular las coordenadas del lugar sobre el que el usuario pulsa, guardar estas coordenadas temporalmente, detectar las coordenadas del lugar en el que el usuario suelta la pantalla y calcular la diferencia entre ambos puntos. A partir de los datos recibidos es posible calcular la distancia y la dirección.

Durante el diseño de la *activity* se incluye en el fichero xml un elemento llamado *ViewFlipper* capaz de detectar gestos. A continuación se declaran varias variables en el código Java. Las variables necesarias para calcular los gestos son:

```
private ViewFlipper vf;
private float oldTouchValueX;
private float oldTouchValueY;
private double difX;
private double difY;
```

El código responsable en procesar el evento se llama *onTouchEvent* (usado ya en otra ocasión). En este caso se añaden un *switch-case* que detecta las acciones de pulsar y soltar la pantalla táctil. Dentro del *case* se obtienen y guardan las coordenadas del punto de pulsación. Para ello se usan los eventos *touchevent.getX()* y *touchevent.getY()*.

```
@Override
public boolean onTouchEvent (MotionEvent touchevent) {

    switch (touchevent.getAction())
    {
        case MotionEvent.ACTION_DOWN:
        {
            oldTouchValueX = touchevent.getX();
            oldTouchValueY = touchevent.getY();
        }

        case MotionEvent.ACTION_UP:
        {
            float currentX = touchevent.getX();
            difX= currentX - oldTouchValueX;
            float currentY = touchevent.getY();
            difY= currentY - oldTouchValueY;

            String abc="" + difX;

        } //case
    }
```

En el segundo caso (*case*), que se cumple al soltar la pantalla, se realiza la resta entre el primer y el segundo punto (pulsación y abandono) para ambos ejes. De esta manera es fácil calcular la

longitud del trazo y la dirección. Para evitar que en contactos accidentales con la pantalla se cambie de lugar de interés, se establece que un gesto tiene que ser más largo que 10 píxeles (sobre la pantalla).

Como explicado a final del punto 9.2.4.c, la *activity* monumento usa una variable tipo `int` para determinar el monumento a mostrar en pantalla. Esta variable se usa también en este lugar para mostrar el siguiente monumento de la lista. En el caso de que el desplazamiento (diferencia entre punto inicial y final) sea positivo, se muestra el siguiente monumento de la lista (`numeroMonumento++`), en el caso de un valor negativo se muestra el monumento anterior (`numeroMonumento--`).

El código `numeroMonumento--` equivale a `numeroMonumento=numeroMonumento-1`.

```
if (difX >= 10){
    numeroMonumento++;
    if (numeroMonumento>=16){
        numeroMonumento=0;
    }
    mostrarMonumento();
}
else if (difX<=-10){
    numeroMonumento--;
    if (numeroMonumento<=-1){
        numeroMonumento=16;
    }
    mostrarMonumento();
}
```

Tras determinar el siguiente monumento a mostrar en pantalla se realiza una llamada a la función `mostrarMonumento` con lo que se actualiza la pantalla del smartphone.

El uso de este gesto no hubiera hecho falta en esta *activity* ya que al atrasar (usando el botón `hardkey` correspondiente) se llega a la lista de monumentos desde dónde se puede elegir el siguiente monumento a mostrar. La acción ésta se eligió sólo para probar su funcionamiento y las dificultades que conlleva ya que se pueden generar conflictos.

El problema encontrado en este punto fue el siguiente:

El *viewflipper* parece ser diseñado para ser utilizado en *activities* que no usen un *scrollview*.

Un *scrollview* puede detectar un movimiento vertical u horizontal (dependiendo de su configuración). Si dentro de ese *scrollview* se encuentra un *viewflipper*, uno de los dos elementos puede no funcionar de forma correcta al poder recibir informaciones contradictorias (dependiendo del uso que se le quiera dar a una acción).

En este caso por ejemplo hubiera sido muy difícil conseguir que al detectar la acción de bajar o subir por la pantalla, ésta se desplazase verticalmente y moviendo de derechas a izquierdas sobre la ventana principal se cambiase de información mostrada. El *scrollview* vertical no es capaz de interpretar el movimiento horizontal y para el *viewflipper* sería difícil realizar el desplazamiento vertical.

Una alternativa sería escribir la acción del *scrollview* como una acción dentro de un *viewflipper* lo que conllevaría un trabajo muy complejo que en este caso no estaría justificado. Se trataría de reinventar un *scrollview* complejo.

Debido a este problema sólo se incluyó en la barra superior. Para este proyecto no fue necesario pero puede ser útil para otra aplicación en la que no se hace uso de un *scrollview*.

9.3.5 Temporizador e intent automático

Al arrancar la aplicación de Gandía, se muestra una foto de la playa con el Mondúver al fondo. Se trata de una pantalla llamada splashscreen. Tras unos segundos la imagen desaparece y se accede automáticamente a la lista principal.

Para ello hace falta usar un temporizador y un intent automático. En primer lugar se declara una variable con un valor entero. Este valor indica el tiempo en milisegundos que debe de temporizar. El valor es usado en la función *postDelayed*.

```
private final int SPLASH_DISPLAY LENGHT = 3000;
```

La acción que se realizará tras pasar el tiempo indicado (en esta activity en concreto) será la de activar una activity nueva através de un *intent*.

```
new Handler().postDelayed(new Runnable() {  
    @Override  
    public void run() {  
        Intent mainIntent = new  
        Intent(splashscreen.this, lista_principal.class);  
        splashscreen.this.startActivity(mainIntent);  
        splashscreen.this.finish();  
    }  
}, SPLASH_DISPLAY LENGHT);
```

De esta manera se consigue que la ventana de bienvenida desaparezca a cabo de unos instantes. Para una aplicación turística es muy habitual incluir una imagen o ventana de bienvenida de este tipo. Alternativamente se podría haber incluido un botón de “seguir” en esta activity pero no es muy recomendado.

10. Prueba en varios dispositivos

Durante el proceso de programación la aplicación fue probada en el emulador de Android que ofrece el entorno de programación Eclipse. El emulador permite probar las principales funciones de la aplicación pero está limitado al simular difícilmente la posición GPS, el estado de la brújula, etc.

Debido a eso, antes de publicar una aplicación es necesario probarla en dispositivos reales.

En las oficinas de la empresa Aplicae fue posible probar la aplicación con smartphones de diferentes fabricantes.

El primer smartphone en el que se testeó la aplicación fue el modelo Desire de HTC. Se comprobaron todas las funciones de la aplicación con éxito.

A continuación se repitió las mismas pruebas con el dispositivo Nexus S en el que apareció un problema. Después de la primera instalación y el primer uso del mapa de Gandía, la aplicación se bloqueaba al encontrarse con una variable indefinida en el código fuente. Después de cambiar el código fuente y darle un valor predefinido a la variable que causaba el error, la aplicación funcionaba bien en el teléfono Samsung Nexus S.

El siguiente dispositivo en el que se probó la aplicación fue el P500 de LG. El resultado fue satisfactorio. Todo funciona tal como planeado. El teléfono de LG tiene, a diferencia de los móviles anteriores, una pantalla resistiva y se pudo comprobar las características descritas en el punto 2.1.1 de este proyecto.

Adicionalmente se probó la aplicación en dos modelos distintos de la marca HTC (Wildfire y Wildfire S) en ambos dispositivos la aplicación funcionó tal como se esperaba. Finalmente se instaló la aplicación de Gandía también en un Optimus Black de la marca LG.

Tras probar la aplicación satisfactoriamente en el emulador y en seis dispositivos de diferentes fabricantes, la aplicación está preparada para ser distribuida.

La aplicación se distribuye como aplicación gratuita en la tienda online GooglePlay (antes AndroidMarket). La aplicación sirve como publicidad e instrumento de promoción para la empresa Aplicae y la web turística de Gandía.



Imagen 10.1 LG P500 y htc Wildfire S

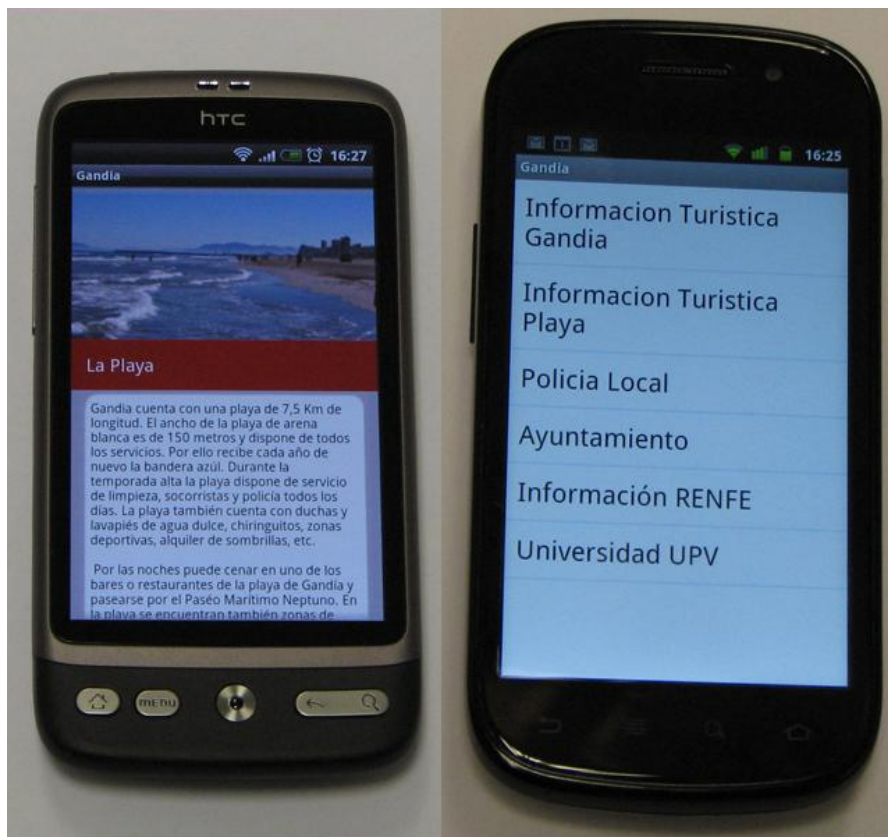


Imagen10.2 htc Desire y Nexus S

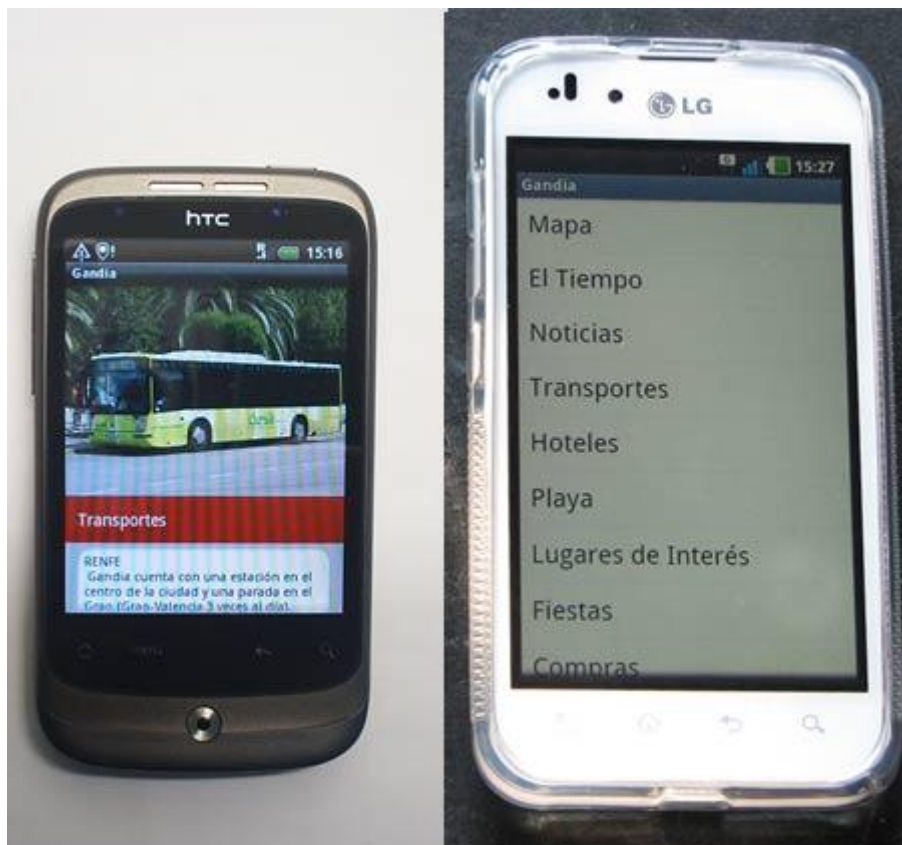


Imagen 10.3 htc Wildfire y LG Optimus Black

11. Publicación de una aplicación

Una vez se termine de programar y testear una aplicación es posible o bien publicarla en una web personal o en el GooglePlay (antes Android Market). Para ello es necesario crear un certificado para la aplicación, registrarse en Google Checkout y darse de alta como programador.

En primer lugar se crea el fichero *.apk (que se instalará en los móviles) con sus correspondientes certificados. El certificado debe ser válido por lo menos para 25 años. Para crear el certificado se pincha sobre la carpeta del paquete con el botón derecho y se selecciona la opción “exportar” de la ventana emergente. A continuación se abren diferentes pantallas en las que se introduce la información requerida y se pulsa sobre “Next >”.

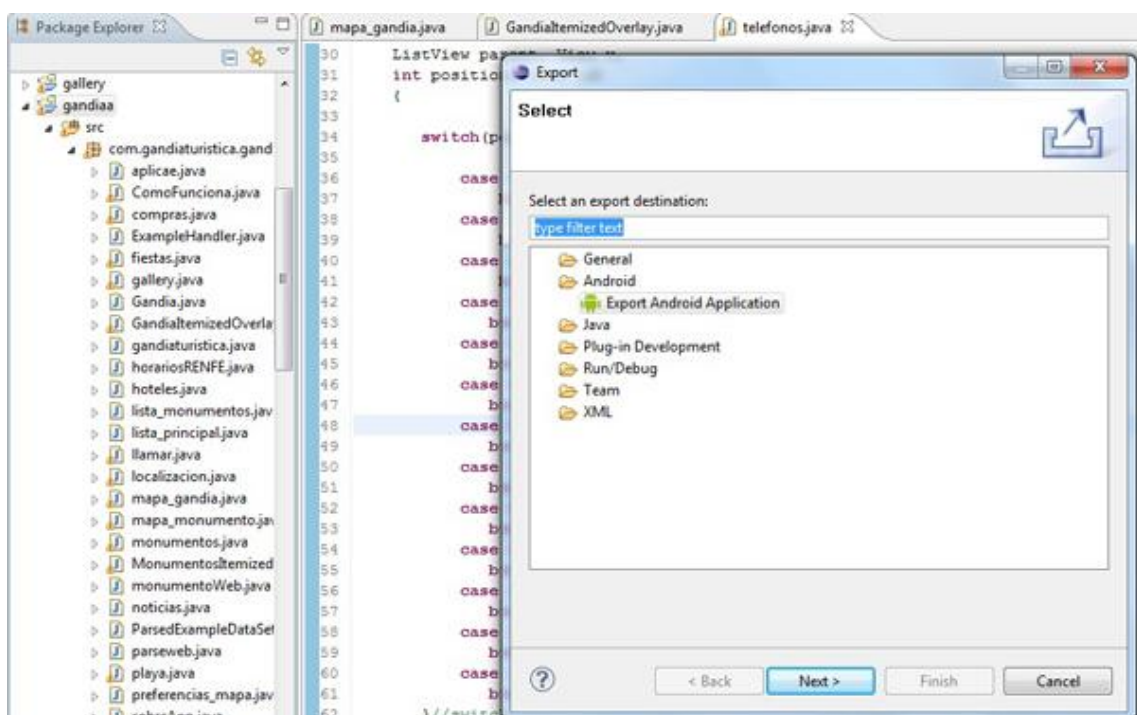


Imagen 11.1 Exportar

La ventana más importantes es la “key creation” en la que se ponen los datos del programador y la empresa creadora así como la contraseña, necesaria para futuras actualizaciones.

Una vez introducidos todos los datos necesarios, se pulsa en el botón “Next >” y se accede a la última ventana en la que se indica en que lugar del disco duro se guarda el fichero *.apk. Sólo los ficheros

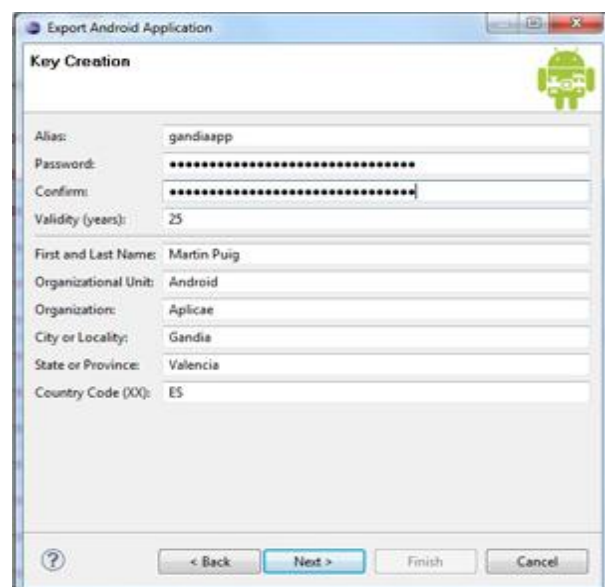


Imagen 11.2 Key Creation

apk con certificados pueden ser publicados en el Market.

Una vez creada el fichero *.apk, se accede a la página web del Android Market con la cuenta Checkout de creador de aplicaciones.

Para conseguir la cuenta de creador, en primer lugar es necesario conseguir una cuenta normal de Google y posteriormente registrarse en la web <http://market.android.com/publish>. En los formularios de esta web se introducen los datos personales y se realiza el pago de los 25 \$ mediante el Google Checkout o con tarjeta de crédito. El pago de los 25\$ es un pago único. Para publicar aplicaciones de Apple es necesario pagar 100\$ cada año.

El proceso de registro se hace una sola vez. Para la publicación de la segunda y las continuas aplicaciones ya no hace falta. El siguiente paso para publicar la aplicación es subirla al Android Market. Para ello se accede con los datos de usuario a la web <https://market.android.com/publish/Home>, se sube el fichero *.apk e imágenes de la aplicación y se rellenan todos los campos del formulario. Los campos del formulario contienen información básica de la aplicación y de su creador. Finalmente el creador de la aplicación puede elegir si desea que su aplicación se distribuya de forma gratuita o en caso contrario puede fijar el precio de venta (del que Google se queda una tercera parte).

A cabo de poco tiempo se publica la aplicación de forma automática (o tras una muy breve revisión) en el Android Market. A partir de ese momento el creador de la aplicación puede seguir su evolución mediante gráficas y estadísticas. Entre las estadísticas se encuentra información cómo por ejemplo el número de descargas realizadas.

12. Estadísticas

Después de publicar la aplicación en GooglePlay (antiguo Android Market) durante la temporada baja y sin anunciar la nueva aplicación se han producido las primeras descargas (realizadas por usuarios que buscaban una aplicación sobre Gandia) en GooglePlay. Los datos no son representativos pero ofrecen información interesante sobre el comportamiento de los usuarios. Si a lo largo del próximo año se anuncia la aplicación y se descarga ésta miles de veces, la información obtenida (feedback) de los usuarios y de su comportamiento puede ser usada para la toma de decisiones para futuros proyectos turísticos en relación con las nuevas tecnologías.



Imagen 12.1 Evolución de instalaciones activas

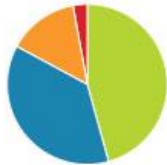
El continuo crecimiento del número de descargas es debido a que la aplicación de Gandía mejoró su posicionamiento en la lista de resultados de GooglePlay hasta llegar a la primera posición para el término “Gandia” y al acercamiento a la temporada alta. Se estima que a medida que llega el verano, el número de usuarios con smartphones aumenta y se anuncia la aplicación, el número de descargas seguirá subiendo. En el caso de la aplicación de Gandía se trata de un crecimiento natural. Aproximadamente una tercera parte del número total de descargas sigue activo en los dispositivos. Esto es la media para la mayoría de aplicaciones.

Las estadísticas ofrecen información valiosa sobre los usuarios (desde el punto de vista tecnológico). Así es posible determinar las versiones del sistema operativo para las que se descargaron las aplicaciones. Esta información es muy valiosa ya que ayudan en la toma de decisiones para la creación de futuras aplicaciones. Conociendo el “nivel tecnológico” y el tipo de smartphones usados por ciudadanos y turistas de Gandía, se puede averiguar por ejemplo si el pago a través de NFC (ver 2.1.4 Protocolos) hará sentido de aquí uno o dos años.

Instalaciones activas de dispositivos de Versión de Android



Instalaciones activas de dispositivos el 9 de mayo de 2012



	Tu aplicación		Todas las aplicaciones de Viajes y guías
<input checked="" type="checkbox"/> Android 2.3.3+	32	45,71 %	59,25 %
<input checked="" type="checkbox"/> Android 2.2	26	37,14 %	25,14 %
<input checked="" type="checkbox"/> Android 4.0.3 - 4.0.4	10	14,29 %	4,87 %
<input type="checkbox"/> Android 2.1	2	2,86 %	6,25 %

Top 10 Versiones de Android para Viajes y guías	
Android 2.3.3+	59,25 %
Android 2.2	25,14 %
Android 2.1	6,25 %
Android 4.0.3 - 4.0.4	4,87 %
Android 3.2	2,08 %
Android 1.6	0,74 %
Android 3.1	0,66 %
Android 4.0 - 4.0.2	0,57 %
Android 2.3	0,23 %
Android 1.5	0,11 %

Imagen 12.2 Instalaciones activas en relación con las versiones de Android

En la imagen 12.2 se puede observar que la versión Android 2.1 (para la que se programó esta aplicación) ya no se usa en la mayoría de los dispositivos (a diferencia que cuando se comenzó a programar). La tecnología evoluciona muy rápido y en la próxima aplicación se podrá descartar la versión 2.1 ya que en éste caso sólo un 2,86% de los usuarios de la aplicación lo están usando (en otras aplicaciones relacionadas con viajes y guías la usan un 6,25%). La ventaja de Android es que todos los dispositivos con versiones nuevas (cómo la 2.2 o 4.0.4) pueden usar aplicaciones creadas para versiones anteriores. También se puede observar en la gráfica superior que el número de dispositivos nuevos (Android 4.0.4) está creciendo rápidamente, mientras que el crecimiento de dispositivos con android 2.2 y 2.3 se está frenando.



<input checked="" type="checkbox"/> Samsung Galaxy S2	14	19,44 %
<input checked="" type="checkbox"/> Samsung Galaxy S (GT-I...	8	11,11 %
<input checked="" type="checkbox"/> HTC Wildfire	5	6,94 %
<input type="checkbox"/> SEMC Xperia Neo V	5	6,94 %
<input type="checkbox"/> SEMC Xperia Arc S	4	5,56 %
<input type="checkbox"/> Samsung Galaxy S Plus	4	5,56 %
<input type="checkbox"/> Samsung Galaxy S (GT-I...	3	4,17 %
<input type="checkbox"/> LG Optimus 2X	2	2,78 %
<input type="checkbox"/> Samsung Galaxy Ace	2	2,78 %
Otras	25	34,72 %

Imagen 12.3 Dispositivos con instalaciones activas

Cómo se ve en la imagen 12.3 es posible conocer cuales son los smartphones mas usados para la aplicación de Gandía. En el momento que los datos sean representativos será posible averiguar cuales de éstos smartphones permiten usar la tecnología NFC por ejemplo y planear nuevos medios de pago o "llaves de habitaciones de hoteles" a través del smartphone en Gandía.

Los ejemplos anteriores han mostrado cómo la interpretación de datos de las nuevas tecnologías pueden ser aplicadas en el sector turístico. Otro ejemplo en el que se relaciona el turismo con la interpretación de datos tecnológicos es el volumen de búsquedas del término “Gandia” en el buscador de Google. Se observa que el comportamiento de búsqueda se repite año tras año y que se incrementa durante el verano. La evolución está completamente relacionada con el comportamiento de los turistas y visitantes de Gandía. El número de turistas aumenta considerablemente durante el verano (temporada baja) y baja hasta un valor mínimo durante el invierno. Estas estadísticas de Google permiten conocer también los lugares geográficos desde que se realizaron las consultas. La gran mayoría (aproximadamente el 95%) se realizaron desde ordenadores localizados en España. Analizando los datos de los usuarios de la página web gandiaturisticacom (que debido a su gran volumen son representativos) pueden afirmar éstos datos con la única diferencia que cada año el número de visitantes de ésta web sigue creciendo mientras que las búsquedas globales de Google para ese término se encuentran en recesión. Como comentado anteriormente, la gran mayoría de usuarios son españoles, lo que está en concordancia con los visitantes de la ciudad de Gandia y el origen de los usuarios de la aplicación de Gandia (ver imagen 12.5).

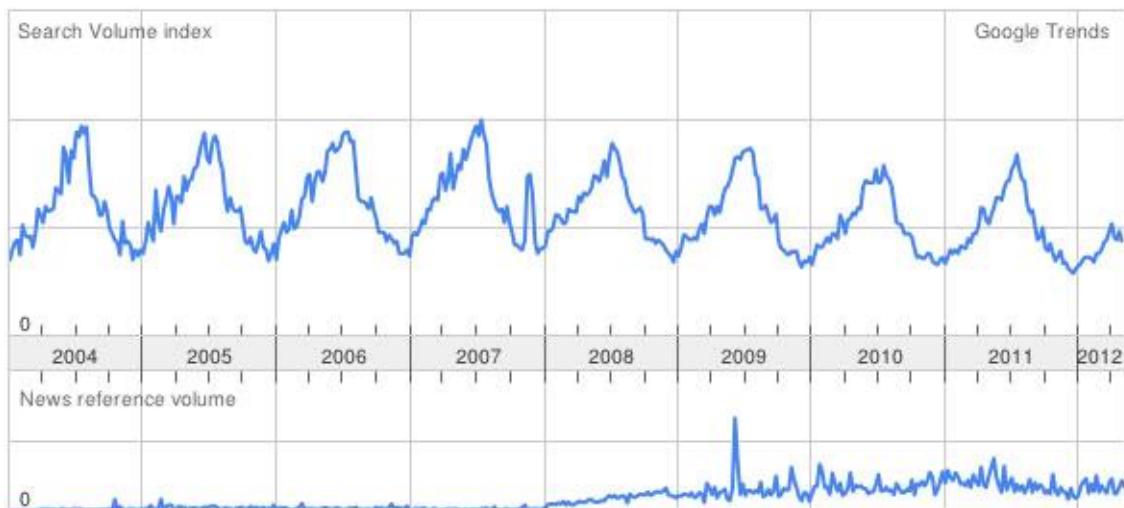


Imagen 12.4 Frecuencia de búsqueda del término “Gandia” en Google

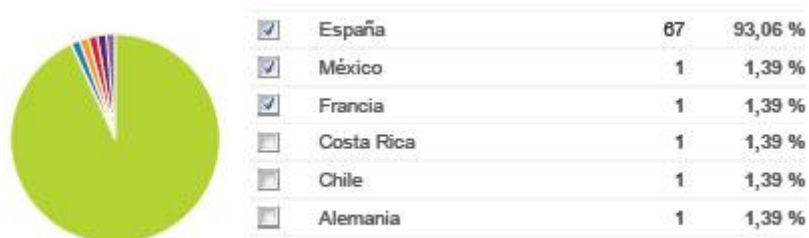


Imagen 12.5 Origen de los usuarios de la aplicación de Gandia

13. Conclusiones

Tras el proceso de diseño y programación de la aplicación de Gandía para dispositivos con sistema operativo Android se pueden comentar varias conclusiones.

Programación:

El proceso de programación ha sido interesante al haber sido posible aprovechar algunos de los conocimientos obtenidos durante la carrera de Sistemas Electrónicos. Aun así fue necesario aprender muchas cosas nuevas específicas del lenguaje de programación Java y de la programación de dispositivos móviles, conocimientos que no se adquieren durante la carrera. Parte de los nuevos conocimientos se adquirieron durante las prácticas en la empresa Aplicae. Para funciones básicas de programación existe abundante información documentada y códigos ejemplo en Internet, mientras que otros aspectos más difíciles, como algunos usados en esta aplicación, no están lo suficientemente documentado. Muchos apartados de la página web oficial para desarrolladores de Android podrían estar mejor documentados.

Rentabilidad:

Desde el punto de vista económico la programación de aplicaciones para el sistema Android no cumple las expectativas. El usuario de estos teléfonos, cómo se vió posteriormente, está acostumbrado a recibir información (buscadores de Google) y dispositivos de forma gratuita o a muy bajo precio. La mayoría de aplicaciones descargadas y ofrecidas son versiones gratuitas. Esto es una situación muy diferente al caso de los usuarios del iPhone. Sólo pocos programadores pueden vivir de la programación para dispositivos Android. Aun así parece un mercado interesante para destinos turísticos o empresas y páginas webs importantes para promocionarse (o a través de las estadísticas conocer mejor a los usuarios para ofrecerles productos individualizados). El mejor ejemplo para ello son páginas webs de redes sociales como Facebook y Twitter que ofrecen aplicaciones gratuitas.

Turismo:

La aplicación puede funcionar como un valor añadido para la promoción del destino turístico ya que es la primera de su índole en Gandía. En cuanto al destino turístico se llega a la conclusión que el Ayuntamiento de Gandía se tiene que abrir más a las nuevas tecnologías e involucrar a más personas formadas tanto en el sector de la innovación de tecnologías como el turístico en las tareas diarias. Se trata de un problema comentado sobre todo por diferentes profesores y compañeros de la carrera de turismo, experiencias vividas durante prácticas en el Ayuntamiento de Gandía y en el trato como administrador de gandiaturistica.com. En mi opinión la ciudad no aprovecha los conocimientos y la formación de la universidad que tiene en la propia ciudad.

Tras analizar las estadísticas de descarga y compararlas con el modelo turístico actual de la ciudad y con las estadísticas de gandiaturistica.com y las búsquedas de Google, se llega a la conclusión de que una aplicación de éste tipo es una buena fuente de información para conocer a los usuarios. El conocimiento adquirido se puede aprovechar para mejorar la oferta turística del destino y con ello mejorar la rentabilidad de determinados negocios.

14. Bibliografía:

Libros:

JESÚS TOMÁS GRIONÉS (2011), *El gran libro de Android*. Barcelona: Marcombo

ED BURNETTE (2011), *Programación Android*. Madrid: Anaya Multimedia

FRANK ABLESON, CHARLIE COLLINS y ROBISEN (2010), *Android, Guía para desarrolladores*. Madrid: Anaya Multimedia

JAVIER CABALLOS (2001), *El lenguaje de programación Java*. Madrid: Ra-Ma

W.CLAY RICHARDSON, DONALD AVONDOLIO, JOE VITALE, SCOT SCHRAGER, MARK W. MITCHELL y JEFF SCANLON (2005), *Profesional Java 2 v5.0*. Madrid: Anaya Multimedia

Páginas webs:

[1 pag. 1] AREAMOBILE. *Android im Weltall*. 2010. <<http://www.areamobile.de/news/17535-android-im-weltall-neue-messdaten-und-videos-veroeffentlicht>>(19 de Octubre de 2011)

[2 pag. 4] NIELSEN. *Smartphones Insight Q3*. 2011.
< <http://recursos.anuncios.com/files/467/35.pdf> >(9 de mayo de 2012).

[3 pag. 4] NIELSEN. *Nota Prensa Smartphones*. 2011.
<http://es.nielsen.com/trends/documents/NotaPrensaSmartphone_v3.pdf >(9 de septiembre de 2011).

[4 pag. 4] LG. *Las Pantallas táctiles*.2010. <<http://www.lgblog.es/2010/07/23/las-pantallas-tactiles-pantalla-capacitiva-vs-resistiva/> > (9 de Septiembre de 2011)

[5 pag. 5] WIKIPEDIA. *Indium tin oxide*. 2011.< http://en.wikipedia.org/wiki/Indium_tin_oxide> (22 de Septiembre de 2011)

[6 pag. 5] HOWSTUFFWORKS. *iPhone*. 2010
<<http://electronics.howstuffworks.com/iphone1.htm>>(12 de Septiembre de 2011)

[7 pag. 7] WIKIPEDIA. *UMTS*. 2011.
<http://de.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System> (9 de Septiembre de 2011)

[7 pag. 9] FR_ONLINE. *iPhone- "Spitzel für die Hosentasche"*. 2011. <<http://www.fr-online.de/kultur/fr-fernsehkritik/berichte-aus-dem-therapiebereich/-/1473344/8385040/-/index.html> > (9 de Septiembre de 2011)

[8 pag. 9] APPLE. *iPhone*.2011.

<http://store.apple.com/es/browse/home/shop_iphone/family/iphone> (9 de Septiembre de 2011)

[10 pag. 10] GARTNER. *Q2 2009-2011*. 2010.

<<http://www.thetelecomblog.com/wp-content/uploads/2010/08/gartner-q2-2010.png>> (12 de Septiembre de 2011)

[11 pag. 10] GARTNER. Table 2 Worldwide Smartphone Sales Q3 2011.

2011.<<http://www.gartner.com/it/page.jsp?id=1848514> (9 de Mayo de 2012)

[12 pag. 21] ANDROID. *Platform-versions*. 2011.

<<http://developer.android.com/resources/dashboard/platform-versions.html>> (13 de Septiembre de 2011)

[13 pag. 29] GOOGLE. *SensorSimulator*. 2011.

<<http://code.google.com/p/openintents/wiki/SensorSimulator>> (16 de Septiembre de 2011)

[14 pag. 50] GOOGLE. *Maps API*. 2011. <<http://code.google.com/intl/es/android/add-ons/google-apis/mapkey.html>>(26 de Septiembre de 2011)

[15 pag. 53] WIKIPEDIA. *Array*. 2011. <<http://es.wikipedia.org/wiki/Array>> (1 de Octubre de 2011)

ANDROID. *Android Developers*.2011. <<http://developer.android.com/>> (Diferentes artículos)

Videos interesantes:

GOOGLENEXTUS. *Android in Space - the full story*. 2010.

<<http://www.youtube.com/watch?v=x8XjldUQns>>(19 de Octubre de 2011)

GOOGLENEXTUS. *Android in Space - Nexus S on Space Shuttle Atlantis*. 2011.

<<http://www.youtube.com/watch?v=RESEgrhmMjc>> (19 de Octubre de 2011)

15. Índice de imágenes

General:

2.1 Pantalla resistiva	pag. 5
2.2 Pantalla capacitiva	pag. 5
2.3 Teclados	pag. 7
2.4 Cuotas de mercado	pag.10
2.5 Relación precios	pag. 10
4.1 Logotipo Java	pag. 13
4.2 tabla resumen de programación en Java	pag.14
4.3 Demanda de lenguajes de programación	pag. 16
5.1 Ciclo de vida de una activity	pag.18
6.1 Android SDK and AVD Manager	pag.20
6.2 Relación del uso de plataformas Android	pag.21
6.3 Entorno de programación	pag. 25
6.4 Graphical Layout	pag. 25
6.5 Problemas en el código	pag.26
6.6 Ejecución de una aplicación	pag 26
6.7 Emulador Android del entorno Eclipse	pag.27
6.8 La consola	pag.28
8.1 Mapa	pag.32
8.2 El tiempo	pag.32
8.3 Noticias	pag. 33
8.4 Transportes	pag. 33
8.5 hoteles	pag. 33
8.6 Playa	pag. 34
8.7 Lugares	pag. 34
8.8 Fiestas	pag. 34
8.9 Compras	pag. 35
8.10 Localización	pag. 35
8.11 Galería de fotos	pag. 35
8.12 Teléfonos	pag. 36
8.13 Web Gandia	pag. 36
8.14 Web Aplicae	pag. 36
9.1 Activity Playa	pag. 37
9.2 Activity del tiempo	pag. 45
9.3 Mapa con menú	pag. 51
9.4 Preferences y ListPreference	pag. 53
9.5 vista satélite	pag. 54
9.6 Más información	pag. 56
9.7 brújula	pag. 57
9.8 Vista horizontal y vertical splashscreen	pag. 59
9.9 Vista horizontal y vertical activity	pag. 59
9.10 Lista telefónica y pantalla para llamar	pag. 61
9.11 Información para desarrolladores	pag. 67

9.12 List view original y personalizado	pag. 68
9.13 Webview aplica	pag. 70
9.14 Webview gandiaturistica	pag. 71
9.15 Webview renfe	pag. 72
10.1 LG P500 y htc Wildfire	pag. 78
11.1 Exportar	pag. 79
11.2 Key Creation	pag. 79
12.1 Evolución de instalaciones activas	pag. 82
12.2 Instalaciones activas en relación con las versiones de Android	pag. 83
12.3 Dispositivos con instalaciones activas	pag. 83
12.4 Frecuencia de búsqueda del término "Gandia" en Google	pag. 84
12.5 Origen de los usuarios de la la aplicación de Gandia	pag. 84

De otras fuentes:

2.1 Pantalla táctil resistiva

CPU Toaster. *smartphone screen confused*. 2011.

<<http://gputoaster.wordpress.com/2011/06/09/smartphone-screens-confused/>> (22 de Septiembre de 2011)

2.2 Pantalla táctil capacitiva

CPU Toaster. *smartphone screen confused*. 2011.

<<http://gputoaster.wordpress.com/2011/06/09/smartphone-screens-confused/>> (22 de Septiembre de 2011)

2.3 Teclados

HTC. *Smartphones*. 2011. <<http://www.htc.com>> (24 de Septiembre de 2011)

NOKIA. *productos*. 2011. <<http://www.nokia.es>> (24 de Septiembre de 2011)

2.5 DISTIMO. *Proportions apps*. 2010.

<<http://www.mobilemarketingwatch.com/wordpress/wp-content/uploads/2010/09/Distimo-60-Percent-Of-Android-Apps-Are-Free-Vs.-29-Percent-Of-iOS-Apps-Prices-Continue-To-Rise.png>> (12 de Septiembre de 2011)

4.1 SUN. *Java Logo*. 2011. <<http://www.java.sun.com>>(13 de Septiembre de 2011)

4.3 INDEED. *Job Trends*. 2011.

<<http://www.indeed.com/trendgraph/jobgraph.png?q=Java%2C+C%2B%2B%2C+C%23%2C+Perl%2C+PHP%2C+Ruby%2C+Groovy%2C+Scala>> (18 de Octubre de 2011)

5.1 ANDROID. *Activity Lifecycle*. 2011.

<http://developer.android.com/images/activity_lifecycle.png> (6 de Octubre de 2011)

12.1, 12.2, 12.3 y 12.5 GOOGLEPLAY. *Consola del desarrollador*.2012.

<<http://play.google.com>> (8 de Mayo de 2012)

12.4 GOOGLE. *Google Trends*. 2012. < <http://www.google.com/trends/?q=gandia>> (12 de Mayo de 2012)

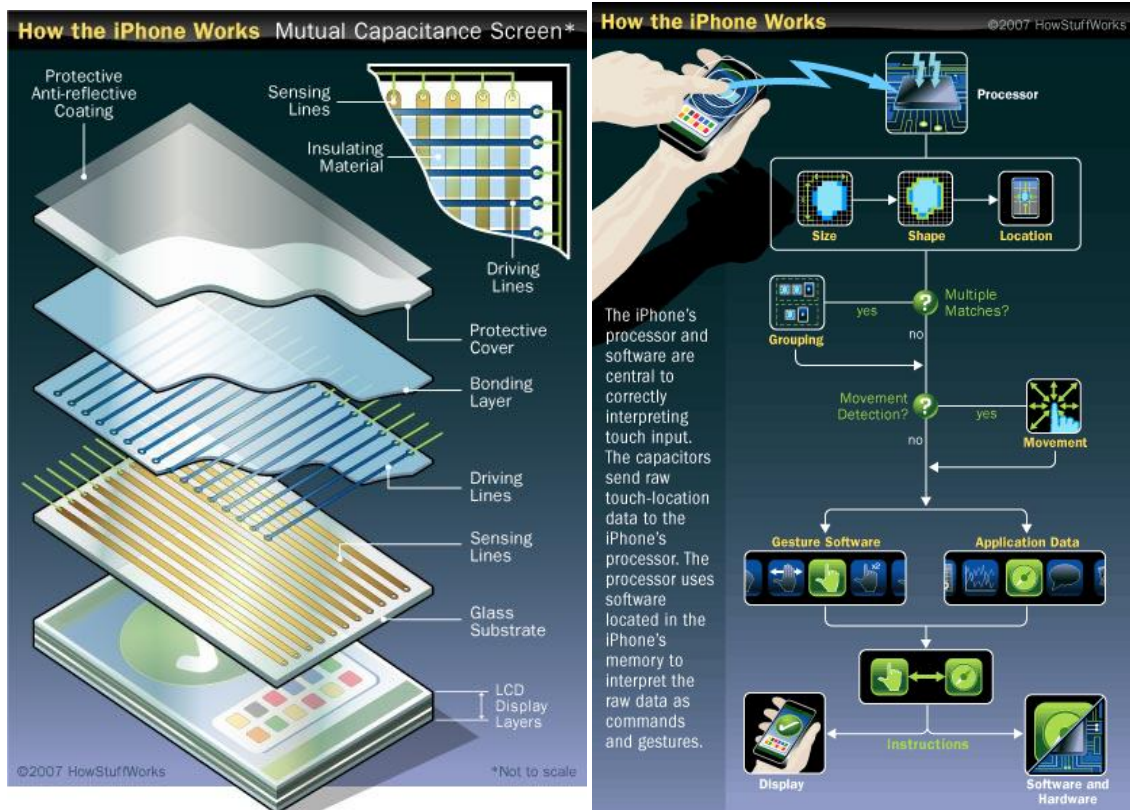
16. Anexos

Anexo 1: Pantallas táctiles

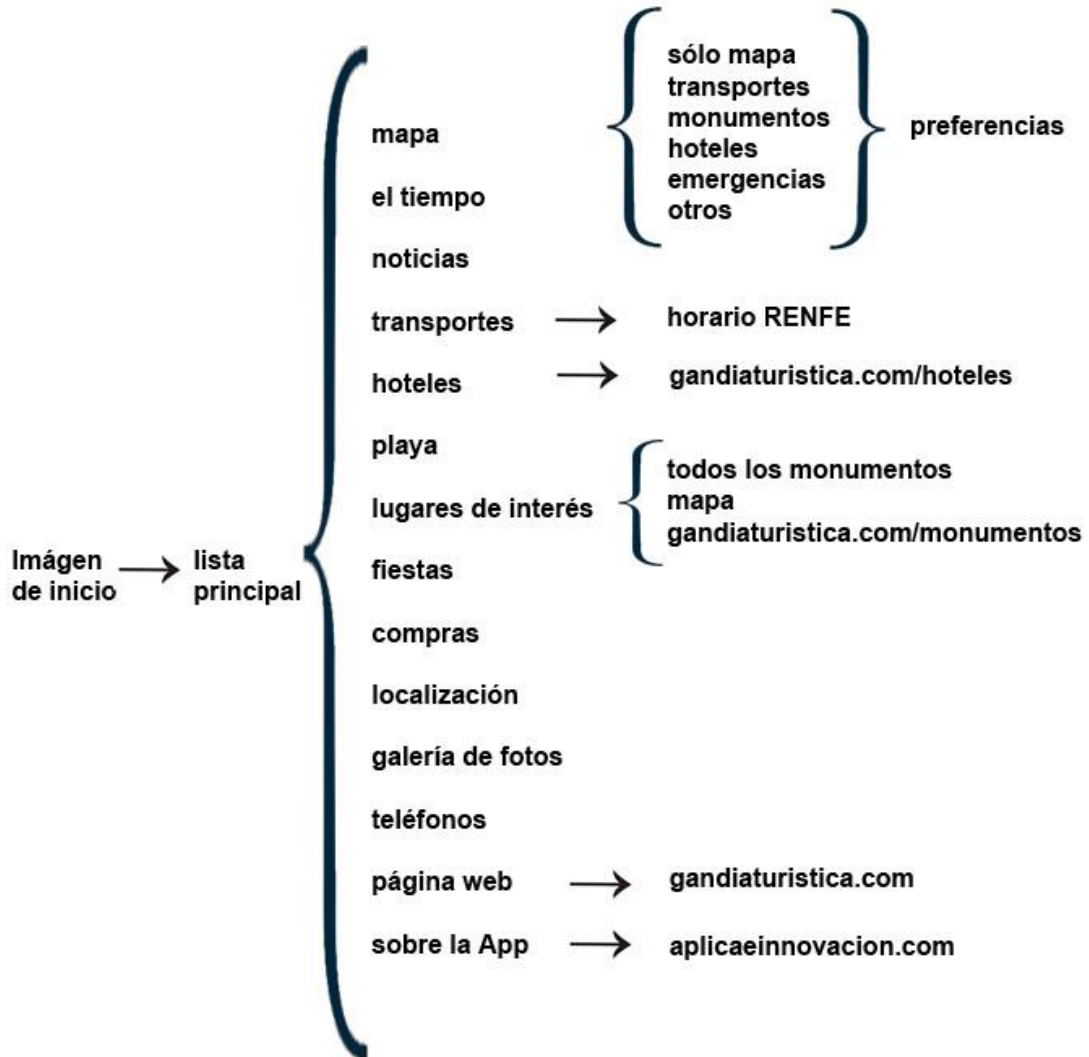
Artículo interesante de nueve páginas sobre el funcionamiento de pantallas táctiles (en inglés)

HOWSTUFFWORKS. *iPhone*. 2007-2010

<<http://electronics.howstuffworks.com/iphone1.htm>>(12 de Septiembre de 2011)



Anexo 2: Estructura de la aplicación de Gandia



Anexo 3 Código xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="0px"
    android:background="#BEBABA">

    <ScrollView android:id="@+id/ScrollView01"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent">

        <RelativeLayout android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:background="#BEBABA">

            <ImageView android:id="@+id/foto_monumento"
                android:layout_below="@id/ScrollView01"
                android:src="@drawable/colegiata"
                android:layout_height="wrap_content"
                android:layout_width="fill_parent"
                android:layout_marginTop="0px"
                android:layout_marginRight="0px"
                android:layout_marginLeft="0px"
            </ImageView>

            <TextView
                android:id="@+id/titulo"
                android:layout_below="@id/foto_monumento"
                android:background="#A70505"
                android:layout_width="fill_parent"
                android:textColor="#ffffff"
                android:textSize="19dip"
                android:padding="15dip"
                android:layout_height="wrap_content"
            />

            <TextView
                android:id="@+id/informacion"
                android:background="@drawable/text_back400"
                android:layout_below="@id/titulo"
                android:layout_width="fill_parent"
                android:textColor="#000000"
                android:layout_height="wrap_content"
                android:padding="10dip"
                android:layout_marginLeft="10dip"
                android:layout_marginRight="10dip"
            />

        </RelativeLayout>
    </ScrollView>
</RelativeLayout>
```

Anexo 4: Código fuente de playa.java

```
package com.gandiaturistica.gandia;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.widget.ImageView;
import android.widget.ScrollView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ViewFlipper;

public class playa extends Activity {
    private TextView CampoTitulo;
    private TextView CampoInformacion;
    private TextView CampoContacto;
    private ImageView CampoPlaya;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.playa);

        CampoTitulo = (TextView) findViewById(R.id.titulo);
        CampoInformacion = (TextView) findViewById(R.id.informacion);
        CampoPlaya = (ImageView) findViewById(R.id.foto_monumento);

        mostrarMonumento();
    }

    public void mostrarMonumento(){
        CampoTitulo.setText(R.string.titulo_playa);
        CampoInformacion.setText(R.string.informacion_playa);
        CampoPlaya.setImageResource(R.drawable.playa);
    }

    @Override
    public void onBackPressed() {
        Intent intent = new Intent(playa.this,lista_principal.class);
        startActivity(intent);
        finish();
    }
    return;
}
}
```

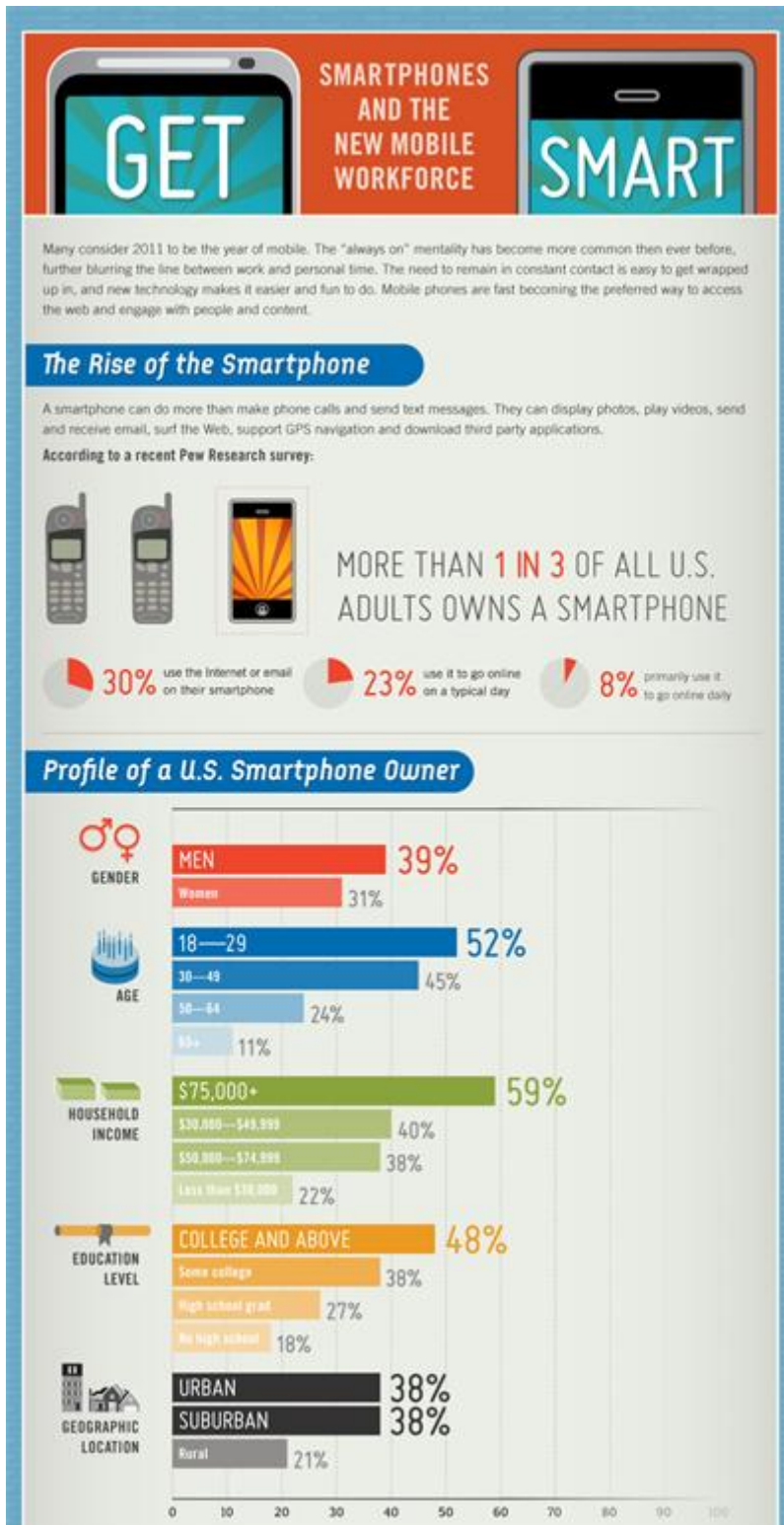
Anexo 5 Extracto del código de la Weather API de Google

```
- <xml_api_reply version="1">
- <weather module_id="0" tab_id="0" mobile_row="0" mobile_zipped="1" row="0" section="0">
- <forecast_information>
  <city data="Gandia, Valencia"/>
  <postal_code data="Gandia, Spain"/>
  <latitude_e6 data=""/>
  <longitude_e6 data=""/>
  <forecast_date data="2011-09-19"/>
  <current_date_time data="2011-09-19 10:30:00 +0000"/>
  <unit_system data="SI"/>
</forecast_information>
- <current_conditions>
  <condition data="Parcialmente nublado"/>
  <temp_f data="75"/>
  <temp_c data="24"/>
  <humidity data="Humedad: 29%"/>
  <icon data="/ig/images/weather/partly_cloudy.gif"/>
  <wind_condition data="Viento: SO a 8 km/h"/>
</current_conditions>
- <forecast_conditions>
  <day_of_week data="lun"/>
  <low data="15"/>
  <high data="27"/>
  <icon data="/ig/images/weather/mostly_sunny.gif"/>
  <condition data="Mayormente soleado"/>
</forecast_conditions>
- <forecast_conditions>
  <day_of_week data="mar"/>
  <low data="16"/>
  <high data="28"/>
  <icon data="/ig/images/weather/mostly_sunny.gif"/>
  <condition data="Mayormente soleado"/>
```

Anexo 6: Estudio demográfico de usuarios de smartphones (Socialcast)

SOCIALCAST. *Smartphones and the Mobile Workforce*. 2011

<<http://mobilemetrics.de/2011/08/14/infografik-smartphones-soziodemographie-oss-work-life-und-apps-im-unternehmenseinsatz-in-den-usa/>>(18 de Octubre de 2011)



More Users Choose Android

According to a 2011 Pew Internet research study, in the U.S. "Android is the most common smartphone platform, followed by iPhone and BlackBerry."



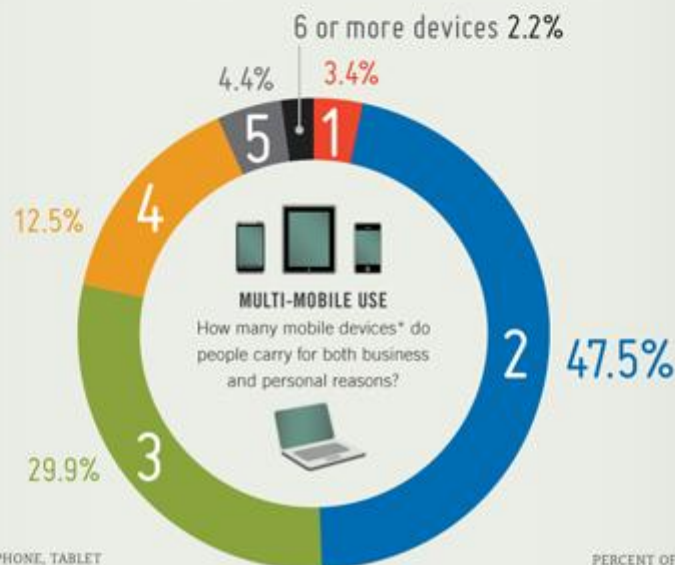
Hyperconnected

U.S. Employees are becoming increasingly tied to their smartphones, ready to respond to work related requests at any time. This trend is pointing to an ever growing on-demand workforce.



WORK/LIFE BALANCE

Most U.S. mobile employees think that the way smartphones affect their work/life balance is situational, sometimes helping and sometimes negatively affecting it.



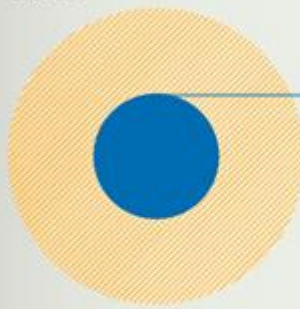
Mobile Apps in the Enterprise

U.S. SMALL BUSINESSES EMBRACE MOBILE

According to a 2011 AT&T survey that asked small businesses about their mobile application use:



BY 2015



The mobile app market is positioned to grow from **\$6.8 billion** to **\$38 billion.**

More than **HALF**

of all IT professionals expect developers to focus more on creating **mobile apps** rather than traditional **enterprise computing platforms.**



GO-TO APPS FOR U.S. EMPLOYEES

PROJECT MANAGEMENT



SOCIAL NETWORKING



NEWS



Sources: "Smartphone Adoption and Usage." (Pew Research, 2011) | "Mobile Workforce Report." (iPass, 2010) Forrester Research | Small Business Trends | IBM | AT&T | techfarms.com

© 2011 SOCIALCAST INC. ALL OTHER TRADEMARKS HEREIN ARE RECOGNIZED TO BE THE PROPERTY OF THEIR RESPECTIVE OWNERS.

Anexo 7: Estudio demográfico de usuarios de teléfonos móviles (Wilson Electronics)

WILSON ELECTRONICS. *The shocking demographics of cell phone use*. 2010
<<http://www.24mobile.de/blog/wp-content/uploads/2010/12/handynutzung-wahrheit2.jpg>>(18 de Octubre de 2011)

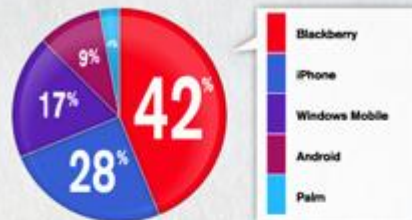


Sources:
http://en.wikipedia.org/wiki/List_of_countries_by_number_of_mob
<http://www.guardian.co.uk/business/interactive/2009/mar/02/mobile-phones>
<http://www.reuters.com/article/idUSL2712199720070627>
<http://www.mapsofworld.com/world-top-ten/countries-with-highest-ratio-of-mobile-phone-users.html>
<http://pewinternet.org/reports/2010/social-media-and-young-adults/part-2/1-cell-phones.aspx?r=1>
http://reviews.cnet.com/8301-13870_7-10454065-78.html
<http://maristpoll.marist.edu/wp-content/misc/us090308/Technology/Have%20Cell%20Phone.htm>





tick tick tick!
Americans use approximately **Minutes Per Day**
(That's about 21 minutes per person)



Top Smartphone Platforms in the United States

Cell phones around the WORLD

Highest average number of cell phones per person

- | | | | |
|-------------------------|------|-------------------|------|
| 1. United Arab Emirates | 1.95 | 6. Lithuania | 1.41 |
| 2. Estonia | 1.94 | 7. Portugal | 1.39 |
| 3. Hong Kong | 1.61 | 8. Singapore | 1.36 |
| 4. Italy | 1.52 | 9. Czech Republic | 1.34 |
| 5. Bulgaria | 1.47 | 10. Russia | 1.33 |



MULTI-TEXTING

Texting while doing other things. How acceptable is it?

over 25 years old under 25 years old



TEXT Popularity



Anexo 8: Acrónimos usados

ADSL	Asymmetric Digital Subscriber
ADT	Android Development Tools
AIDL	Android Interface Definition Language
API	Application Programming Interface
AR	Argumented Reality
AVD	Android Virtual Device
DOM	Document Object Model
GMS	Global System for Mobile Communications
GNU	General Public Licence
GPS	Global Positioning System
HTTP	Hyertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITO	Indium Tin Oxide
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
NDK	Native Development Kit
NFC	Near Field Communication
OpenGL	Open Graphic Library
OS	Operating System
PDA	Personal Digital Assistant
PET	Polyethylene terephthalate
RFID	Radio Frequency IDentification
RAM	Random Access Memory
SAX	Simple API for XML
SDK	Software Development Kit
SOAP	Simple Object Access protocol
UI	User Interface
UMTS	Universal Mobile Telecommunication System
URL	Universal Resource Locator
URI	Uniform Resource Identifier
WLAN	Wireless Local Area Network
WWW	World Wide Web
XML	Extensible Markup Language

Creación de una aplicación, programada en Java, para smartphones basados en el sistema operativo Android para un portal turístico.



Gratis en el Android Market

Ing. Téc. Telecomunicación Sistemas Electrónicos

Autor: Martín Puig Tenschart
 Tutor UPV: Jordi Mauri Castelló
 Tutora Empresa: Esther Ramos Escalera

¿Cómo funciona?

La aplicación Android fue programada en lenguaje Java y hace uso de los diferentes sensores y funciones de los diferentes dispositivos basados en el sistema operativo Android. La aplicación accede al sensor de aceleración, brújula digital, GPS, pantalla táctil y se conecta a diferentes servidores para recibir información actual. Para ello usa un SAX parser que permite la interpretación de código xml que es analizado para ser presentado al usuario final de forma legible.

El sistema operativo Android está basado en el sistema Linux por lo que para la ejecución es necesario la otorgación de permisos especiales. Esto lo convierte en un sistema estable, seguro y gratuito (Open Source).






El crecimiento exponencial de ventas de dispositivos basados en Android, lo convierten en el sistema operativo líder de mercado.



Desarrollo del proyecto:

1. Análisis de diferentes sistemas operativos
2. Estudio de los sensores electrónicos de los dispositivos
3. Diseño y programación de la aplicación
4. Testeo en smartphones de diferentes fabricantes
5. Publicación en el Android Market





¿A quien va dirigida?

La aplicación se creó tanto para turistas como para habitantes de Gandía. La aplicación ofrece no sólo información turística sobre el destino (mapas, lugares de interés, hoteles, fiestas, etc.) sino también información actual como noticias, pronóstico de tiempo o los horarios actuales de los trenes. Esta aplicación es la primera de su índole para el municipio de Gandía.



XV
edición

premios Bancaja



GANDIATURISTICA
.COM



