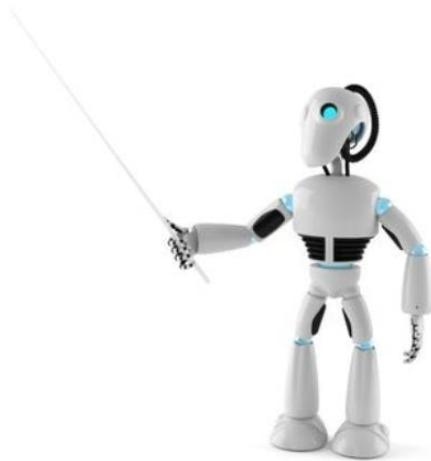


Apuntes del curso Introducción a la Programación con Android en Java



Estos apuntes se han elaborado siguiendo el curso:

Android: Introducción a la Programación.

Y su autoría intelectual corresponde a quienes crearon dicho curso.

<https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/course/>

UNIDAD 1. Java

introducción

video [Características de Java](#)

Creación de clases en Java

video [Creación y utilización de clases](#)

Java es un lenguaje orientado a objetos, donde casi todo se realiza utilizando objetos. Por lo tanto va a resultar vital conocer cómo definir clases de objetos de Java.

Antes conviene definir estos términos:

Objeto: entidad que dispone de unas propiedades (atributos) y comportamiento (métodos).

Clase: define un tipo de objeto concreto.

Por ejemplo, si quisieramos escribir un programa en Java para gestionar los libros de una biblioteca, crearíamos la clase Libro, y luego un objeto de esta clase, por cada libro que tengamos en la biblioteca.

Una clase en Java está formada por:

- **Atributos:** (o campos/propiedades) Almacenan algún tipo de información del objeto. Definen su estado.
- **Constructor(es):** Método que se utiliza para inicializar un objeto.
- **Métodos:** Son utilizados para modificar o consultar el estado de un objeto. Equivalentes a las funciones o procedimientos de otros lenguajes.

NOTA: En Java solo hay dos excepciones al uso de objetos, además de estos podemos utilizar variables simples y arrays. La forma de hacerlo es muy similar a como lo haríamos en C o C++. En la siguiente tabla se muestra una tabla con los tipos simples disponibles en Java.

tipo	tamaño	rango
byte	8 bits	-128 127
short	16 bits	-32.768 32.767
int	32 bits	-2.147.483.648 2.147.483.647
long	64 bits	-9.223.372.10 ¹² 9.223.372.036.854.775.807
float	32 bits	-3.4·10 ³⁸ 3.4·10 ³⁸ (mínimo 1.4·10 ⁻⁴⁵)
double	64 bits	-1.8·10 ³⁰⁸ 1.8·10 ³⁰⁸ (mínimo 4.9·10 ⁻³²⁴)
boolean		false true
char	16 bits	Unicode 0 Unicode 2 ¹⁶ -1

Para crear una nueva clase, lo primero que tienes que hacer es crear un fichero que ha de llamarse exactamente igual que la clase y con extensión .java. Recuerda, siempre un fichero por clase.

Al principio de este fichero se suele indicar el paquete al que pertenecerá esta clase. Por ejemplo:

```
package org.upv.cursoandroid.unidad0;  
seguido de otros paquetes definidos previamente que queremos utilizar:
```

```
import android.app.Activity;
import android.content.BroadcastReceiver;
import ...
Cómo las clases son agrupadas en paquetes será explicado más tarde. De momento no es
imprescindible para trabajar con los primeros ejemplos.
Para declarar una clase sigue el siguiente esquema:
class <clase> {
    //declaración de atributos
    [visibilidad] [modificadores] <tipo> <atributo> [= valor];
    ...
    //declaración de constructor
    public <clase>(<argumentos>) {
        <instrucciones>;
    }
    //declaración de métodos
    [visibilidad] [modificadores] <tipo> <método>(<argumentos>) {
        <instrucciones>;
    }
    ...
}
```

donde:

[visibilidad] = public, protected o private

[modificadores] = final, static y abstract

El significado de estas palabras se explica más adelante.

Una clase comienza por la palabra reservada **class** seguida del nombre de la clase. Por convenio los nombres con la inicial mayúsculas. Si queremos usar varias palabras, las concatenamos sin espacios con las iniciales en mayúscula. A continuación se definen los atributos, seguido del constructor/es y de la declaración de métodos.

Un ejemplo de una clase se muestra a continuación:

```
/** Clase que representa un número complejo. */
class Complejo {

    //declaración de atributos
    private double real, imaginario;

    //declaración de constructor
    public Complejo(double real, double imaginario) {
        this.real= real;
        this.imaginario= imaginario;
    }

    //declaración de métodos
    /** Transcribe el complejo a String.
     * @return un string con la parte entera y la imaginaria
     */

    public String toString() {
        return real + "+" + imaginario + "i";
    }

    /** Suma al complejo de este objeto otro complejo.
```

```
* @param v el complejo que sumamos
*/
public void suma(Complejo v) {
    real = real + v.real;
    imaginario = imaginario + v.imaginario;
}
}
```

Una clase necesita almacenar algún tipo de información que defina el objeto en cuestión. Esto se hará en los atributos o campos de la clase. En el ejemplo queremos representar un número complejo, y por lo tanto vamos a necesitar almacenar la parte real y la parte imaginaria del número.

El siguiente paso suele ser declarar el constructor. Un constructor es un método que tiene el mismo nombre de la clase y que se utiliza para inicializar un objeto. Tiene una serie de parámetros separados por comas, en nuestro ejemplo, la parte real y la parte imaginaria del número complejo. El código de este método (las instrucciones entre {...}) se limita a copiar estos valores en los atributos del objeto. Aunque aquí aparece un problema, los parámetros del método (real, imaginario) y los atributos de la clase (real, imaginario) se llaman igual. Para resolverlo podemos anteponer a los atributos la palabra reservada this. De esta forma indicamos que nos referimos a los atributos del objeto.

Para terminar declaramos los métodos de la clase. Los métodos son equivalentes a las funciones o procedimientos de otros lenguajes y son utilizados para modificar o consultar el estado de nuestro objeto.

Los métodos suelen comenzar indicando su visibilidad (public,protected o private). El significado de cada una de estas palabras se explica más adelante. A continuación los métodos indican el tipo de datos que devuelven. En el ejemplo el primer método devuelve un String y el segundo void, o lo que es lo mismo nada. Luego sigue el nombre del método y los parámetros. Dentro del primer método se utiliza la palabra reservada return para indicar el valor devuelto por el método.

Comentarios y documentación javadoc

Objetivos:

- Describir la forma de introducir comentarios y documentar nuestro código.

NOTA. Este apartado no es esencial para el desarrollo del curso. Si no dispones de tiempo puedes saltártelo.

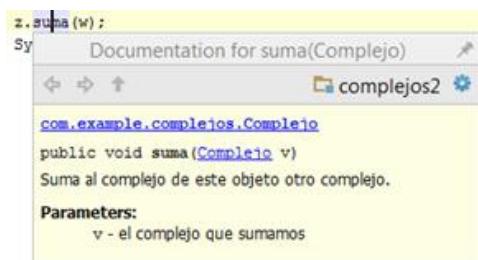
Los comentarios en Java se introducen de la misma forma que en C o en C++.

```
// Comentario de una línea
/* Comentario de
varias líneas */
```

La documentación del código resulta vital si queremos que este sea reutilizable. Java nos permite realizar esta importante tarea utilizando en mínimo esfuerzo. Para descubrir cómo se utiliza te proponemos el siguiente ejercicio.

Ejercicio paso a paso: *Documentación del código utilizando javadoc.*

1. Desde **Android Studio**, accede a *File > Settings.. >Editor > General* y activa la opción *Show quick doc on mouse move*.
2. En el proyecto Complejos visualizar el código de la clase Principal.
3. Sitúa el ratón sobre la palabra `suma`. Aparecerá una ventana como la siguiente:



Te preguntarás cómo es posible que Eclipse ya disponga de la documentación de una clase que acabamos de escribir.

4. Para responder a esta pregunta visualiza la clase `Complejo` y comprueba que el comentario introducido antes de método `suma` es:

```
/** Suma al complejo de este objeto otro complejo.
 * @param v el complejo que sumamos
 */
```

Todo comentario que introduzcamos de la forma `/** ... */` será utilizado por Java para documentar el elemento que es definido a continuación. Este elemento puede ser una clase, una variable o un método. A este sistema de documentación automática se le conoce como *javadoc*.

El encapsulamiento y la visibilidad en Java

Video [Encapsulamiento y visibilidad en Java](#)

Para un diseño adecuado del software resulta imprescindible un correcto encapsulamiento del código. El mayor problema reside en que el software tiende a cambiar con mucha facilidad, (siempre encontramos mejores formas de resolver un problema) y resulta imprescindible que estos cambios afecten lo menos posible a otras partes de código. Pasemos a definir algunos términos de forma más precisa.

Cuando diseñamos un software hay dos aspectos que resultan fundamentales:

Interface: Cómo este software puede ser utilizado.

Implementación: El código utilizado para resolver los distintos algoritmos.

El concepto de interfaz se define en Java como los elementos de una clase que son visibles desde fuera de esta (con visibilidad public). La implementación se define creando unos determinados atributos y escribiendo el código de los diferentes métodos.

El encapsulamiento consiste en ocultar la implementación y los atributos de un objeto, de manera que sólo se puede cambiar su estado mediante ciertas operaciones definidas en el interface del objeto. Dicho de otra forma, procurar que el interface sea lo más independiente posible de la implementación.

En Java el encapsulamiento está estrechamente relacionado con la visibilidad. Para indicar la visibilidad de un elemento (tanto atributos como métodos) podemos anteceder una de las siguientes palabras reservadas:

public: accesibles desde cualquier clase.

private: sólo son accesibles por la clase actual.

protected: sólo por la clase actual, sus descendientes y clases de nuestro paquete.

<si no indicamos nada> sólo son accesibles por clases de nuestro paquete.

Acceso a los atributos de la clase

Los atributos de una clase están estrechamente relacionados con su implementación. Por esta razón conviene marcarlos como private e impedir su acceso desde fuera. De esta forma en un futuro podremos cambiar la representación interna del objeto sin alterar su interface.

Aunque en la teoría esta afirmación parece lógica, en la práctica resulta difícil de seguir. Pensemos en las dos clases que hemos diseñado. En la de los números complejos al haber puesto private en los atributos no vamos a permitir que acceder la parte entera e imaginaria del objeto:

```
class Complejo {  
    private double real, imaginario;  
    ...
```

Lo mismo ocurre con longitud y latitud de una coordenada geográfica. Por lo tanto, resulta imposible consultar o modificar esta información desde fuera de la clase. Para solucionar este problema vamos a utilizar los métodos getters y setters (obtención y establecimiento). Estos métodos son utilizados para consultar el valor de un atributo (getter) o para modificarlo (setter). Resulta imprescindible indicar que tienen una visibilidad public para que puedan ser invocados desde fuera de la clase. Por ejemplo, para el atributo real escribiríamos los métodos:

```
public double getReal() {  
    return real;  
}  
  
public void setReal(double real) {  
    this.real = real;  
}
```

Esta forma de trabajar puede parecer algo engorrosa, pero tiene sus ventajas:

- Como veremos en el siguiente apartado, podemos cambiar la representación interna de la clase sin alterar el interface
- Verificar que los valores son correctos:

```
public void setReal(double real) {  
    if (real>100000) {lanzamos una excepción}  
    else this.real= real;  
}  
Modificar otros aspectos del objeto o lanzar eventos:  
public void setReal(double real) {  
    contadorModificaciones++; //Con fines estadísticos  
    lanzamos el evento: onComplejoChange  
    this.real= real;  
}
```

Resulta tan frecuente su utilización que Eclipse incorpora una herramienta para crearlos de forma automática. Realiza el siguiente ejercicio para aprender su funcionamiento:

Ejercicio paso a paso: Creación automáticas de getters y setters

1. En el proyecto Complejos, abre la clase Complejo.
2. Pulsa con el botón derecho sobre el código y selecciona la siguiente opción: con Android Studio Generate.../ Getter and Setter y con Eclipse Source / Generate Getters and Setters...
3. Selecciona los dos atributos de la clase y pulsa OK. Verás cómo se inserta el siguiente código:

```
public double getReal() {  
    return real;  
}  
  
public void setReal(double real){  
    this.real = real;  
}  
  
public double getImaginario(){  
    return imaginario;  
}  
  
public void setImaginario(double imaginario) {  
    this.imaginario = imaginario;  
}
```

4. Pulsa en el botón Guardar  para almacenar el fichero.

Practica: Getters y setters en la clase coordenadas geográficas

Inserta en la clase GeoPunto los getters y setters necesarios para acceder a sus atributos.

Cambio de la representación interna de una clase

Tras insertar los cuatro métodos del ejercicio anterior, uno podría pensar que el resultado es el mismo que si hubiéramos puesto public delante de los dos atributos. No es exactamente así, imaginemos que más adelante queremos verificar que la latitud de un GeoPunto esté en el rango [-90, 90], y en caso contrario lanzar una excepción. Si los usuarios de la clase acceden directamente al atributo esto no será posible. Más todavía, imagina que en un momento determinado queremos cambiar la representación interna de un número complejo para utilizar módulo y fase en lugar de parte entera y parte imaginaria. Si hemos tenido la precaución de no publicar los atributos, será posible realizar este cambio sin cambiar el interface de la clase.

```
class Complejo {  
    //declaración de atributos  
    private double modulo, fase;  
  
    //declaración de constructor  
    public Complejo(double real, double imaginario) {  
        modulo = Math.hypot(real, imaginario);  
        fase = Math.atan2(imaginario, real);  
    }  
  
    //declaración de métodos  
    public double getReal() {  
        return modulo * Math.cos(fase);  
    }  
    ...  
}
```

Cuando decimos que no cambiamos el interface de la clase, queremos decir que no hemos modificado ninguna de las llamadas marcadas como public. Por lo tanto, no tendremos que modificar ninguna línea del código que utiliza la clase Complejo. Aunque la forma interna de trabajar ha cambiado radicalmente. En la nueva implementación algunas operaciones son mucho más rápidas (como obtener el módulo), pero otras son mucho más lentas (como la suma). Si quieres puedes ver el ejemplo completo de cómo se podría implementar esta clase en el siguiente fichero.

Practica: Cambio de representación de la clase coordenadas geográficas

Modifica la clase GeoPunto para que los atributos longitud, latitud sean representados mediante un int en lugar de con double. Para disponer de una precisión adecuada ahora representarán millonésimas de grado en lugar de grados. De esta forma tenemos una representación más compacta. Como veremos más adelante esta representación es la utilizada en el API de Google Maps. IMPORTANTE, el interfaz de la clase ha de permanecer exactamente igual. Para el constructor puedes utilizar el siguiente código.

```
public GeoPunto(double longitud, double latitud) {  
    this.longitud = (int) (longitud * 1E6);  
    this.latitud = (int) (latitud * 1E6);  
}
```

La clase Lugar

Objetivos:

- Crear la clase fundamental de la aplicación Mis Lugares.

La aplicación Mis Lugares permite gestionar una colección de lugares. Para cada lugar vamos a poder almacenar mucha información: nombre, dirección, posición geográfica, etc. El primer paso a realizar va a ser crear una clase que nos permita trabajar en Java con este tipo de información. Este tipo de clase se conoce como POJO.

Ejercicio paso a paso: Creación de la clase Lugar

Android Studio está pensado exclusivamente para crear aplicaciones Android. Sin embargo, si sigues los siguientes pasos podrás crear una aplicación 100% Java o Kotlin.

1. Crea un nuevo proyecto (*File > New > New Project..*) con los siguientes datos:

Phone and Tablet / Add No Activity

Name: Mis Lugares Java

Package name: com.example.mislugares

Language: Java ó Kotlin

Minimum API level: API 19 Android 4.4 (KitKat)

NOTA: Deja el resto de los parámetros con su valor por defecto.

2. Pulsa en *File > New > New Module*. Selecciona *Java Library* y pulsa *Next*.

3. Introduce en Library name Mislugares, como Java pakage name: com.example.mislugares, y en Java class name: Lugar. Pulsa el botón *Finish*. Se creará un nuevo módulo Java dentro de tu proyecto Android.

4. Para Kotlin en el explorador de proyecto busca la clase Java y con el botón derecho selecciona *Convert Java File to Kotlin File*.

5. Reemplaza el código de la clase Lugar por el siguiente:

```
package org.example.mislugares;
public class Lugar {
    private String nombre;
    private String direccion;
    private GeoPunto posicion;
    private String foto;
    private int telefono;
    private String url;
    private String comentario;
    private long fecha;
    private float valoracion;

    public Lugar(String nombre, String direccion, double longitud,
                double latitud, int telefono, String url, String comentario,
                int valoracion) {
```

```

        fecha = System.currentTimeMillis();
        posicion = new GeoPunto(longitud, latitud);
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
        this.url = url;
        this.comentario = comentario;
        this.valoracion = valoracion;
    }
}

```

En Java observa como se definen los atributos de la clase y como en el constructor se inicializa para un objeto concreto según los parámetros indicados. En estos parámetros no se indica el atributo fecha. Este representa el día y la hora en que visitamos ese lugar por última vez. Se codifica mediante un long (número entero de 64 bits), que supondremos en formato *Epoch time* o tiempo Unix [\[1\]](#). Es decir, número de milisegundos transcurridos desde 1970. El método System.currentTimeMillis() nos devuelve la fecha y hora actual en este formato. Por lo tanto, siempre que usemos este constructor, en fecha se almacenará el instante en que el objeto fue creado.

6. Crea los métodos *getters* y *setters* para acceder a todos los atributos de la clase. Solo tienes que pulsar con el botón derecho y seleccionar la opción *Generate... > Getter and Setter* y *selecciona todos los atributos mientras mantienes pulsada la tecla Ctrl*.
7. Pulsa con el botón derecho sobre el código y selecciona la opción *Generate... > toString()*... Selecciona todos los atributos y pulsa en *OK*. Se añadirá un método similar a:

```

@Override
public String toString() {
    return "Lugar [nombre="+ nombre+ ", direccion="+ direccion+
           + ", posicion="+ posicion+ ", foto="+ foto+ ", telefono="
           + telefono+ ", url="+ url+ ", comentario="+ comentario+
           + ", fecha="+ fecha+ ", valoracion="+ valoracion+ "]";
}

```

NOTA: El significado de *@Override* se explica más adelante.

8. Dentro del explorador del proyecto *mislugares* > *java* > *com.example.mislugares*, pulsa con el botón derecho y selecciona *New > Java Class o Kotlin File/Class*.
9. Introduce en el campo Name: *GeoPunto* y pulsa Ok. Reemplaza el código por el siguiente (dejando la línea del package):

```

public class GeoPunto {

    private double longitud, latitud;

    static public GeoPunto SIN_POSICION = new GeoPunto(0.0,0.0);

    public GeoPunto(double longitud, double latitud) {
        this.longitud= longitud;
        this.latitud= latitud;
    }

    public String toString() {
        return new String("longitud:" + longitud + ", latitud:"+ latitud);
    }

    public double distancia(GeoPunto punto) {

```

```

        final double RADIO_TIERRA = 6371000; // en metros
        double dLat = Math.toRadians(latitud - punto.latitud);
        double dLon = Math.toRadians(longitud - punto.longitud);
        double lat1 = Math.toRadians(punto.latitud);
        double lat2 = Math.toRadians(latitud);
        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
            Math.sin(dLon/2) * Math.sin(dLon/2) *
            Math.cos(lat1) * Math.cos(lat2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        return c * RADIO_TIERRA;
    }
}

```

El objeto `SIN_POSICION`, será utilizado cuando se quiera indicar que un lugar no tiene posición asignada. Observa que es un objeto de tipo estático. En Java se indica con `static` y en Kotlin con `companion object`. Esto significa que solo va a haber una instancia de este objeto creada desde el principio. Para acceder a ella usaremos `GeoPunto.SIN_POSICION`.

10. Solo para Java crea en esta clase los métodos *getters* y *setters* para acceder a los dos atributos. Igual que antes, pulsa con el botón derecho y seleccionar la opción *Generate... > Getter and Setter*. Realiza la misma operación para *equals()* and *hashCode()*.

11. Para Java y Kotlin crea una nueva clase Java con nombre: `Principal`. Android Studio no permite que la clase principal esté en Kotlin.

12. Reemplaza el código por el mostrado (dejando la línea del package):

```

class Principal {
    public static void main(String[] main) {
        Lugar lugar = new Lugar("Escuela Politécnica Superior de Gandía",
            "C/ Paranimf, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
            962849300, "http://www.epsg.upv.es",
            "Uno de los mejores lugares para formarse.", 3);
        System.out.println("Lugar " + lugar.toString());
    }
}

```

La clase `Principal` es algo atípica: no tiene atributos ni constructor, únicamente el método `main`. Cuando en un proyecto existe una clase que tiene un método con este perfil, es el que se llama para comenzar la ejecución. Como parámetros, este método recibe un *array* de `Strings`. Esta información tiene interés cuando el programa se ejecuta desde la línea de comandos con parámetros.

13. Pulsa en el botón desplegable a la derecha del botón *Run* . Selecciona *Edit Configurations...*

14. En la nueva ventana, haz clic en el signo `+` de la esquina superior izquierda y selecciona *Application*. Aparecerá una nueva configuración de aplicación. Selecciona en *Name: mislugares*, en *Main class: com.example.mislugares.Principal* y en *Use classpath of module: mislugares*. Pulsa en *OK*.

15. Pulsa el botón *Ejecución* y verifica que el resultado que aparece en la ventana de *Run* es similar a:

```

"C\Program ...
Lugar {nombre=Escuela Politécnica Superior de Gandía,
dirección=C/ Paranimf, 1 46730 Gandia (SPAIN),
posición=longitude:-0.166093, latitud:38.995656, foto=null,

```

Apuntes del curso Introducción a la Programación con Android en Java

```
telefono=962849300, url=http://www.epsg.upv.es,  
comentario=Uno de los mejores lugares para formarse.,  
fecha=1392332854758, valoracion=3.0}  
Process finished with exit code 0
```

La herencia en Java

Objetivos:

- Definir el concepto de herencia
- Describir la forma en que podemos crear una nueva clase en Java extendiendo otra ya existente

Video [La herencia en Java](#)

Uno de los grandes atractivos de la programación orientada a objetos es la facilidad que nos ofrece para la reutilización de código. La programación puede llegar a ser muy repetitiva. La reutilización de código consiste en no volver a escribir una y otra vez los mismos algoritmos para resolver situaciones similares. La clave para conseguir esto en Java va a ser la herencia. Si alguna vez ha tratado de reutilizar un código ya escrito para tratar de modificar su comportamiento, habrás comprobado lo complejo que resulta y cómo es muy frecuente cometer errores.

Veamos un ejemplo: Imagina que dispones de un sistema de información geográfica donde cada punto geográfico es representado por la clase GeoPunto. Los atributos de esta clase son longitud y latitud. Nos vemos obligados a cambiar el software dado que ahora necesitamos representar también la altitud de cada punto.

Una opción podría ser reescribir la clase GeoPunto y modificar todos sus métodos. Puede parecer sencillo, pero imagina que esta clase tuviera 10.000 líneas de código. El tiempo empleado y los errores cometidos serían frecuentes. Además todo el código que ya utilizaba GeoPunto tendría que ser modificado.

Java nos proporciona una forma mejor de trabajar. La idea consiste en definir una nueva clase, por ejemplo NuevoGeoPunto, que herede de GeoPunto y que añada el nuevo atributo. De esta forma podremos añadir nuevas funcionalidades sin tener que modificar la clase de partida. Más todavía, es posible que ni siquiera dispongamos del código de una clase para poder heredar de ella. En este apartado vamos a describir como la herencia nos permite resolver el problema de la reutilización de código de forma sencilla, pero a su vez eficiente.

Cuando queremos construir una nueva clase es muy frecuente que ya dispongamos de otra que realice parte de la que queremos hacer. La herencia consiste en crear la nueva clase a partir de otra que llamaremos padre. La clase hija va a heredar los atributos y métodos de la clase padre y podrá crear nuevos atributos y métodos para completar su comportamiento.

La clase hija también puede volver a definir los métodos de la clase padre. En tal caso es recomendable (no obligatorio) indicarlo con `@Override`; de esta forma evitamos errores

habituales cuando cambiamos algún carácter o parámetro. Si un método ha sido sobreescrito podemos acceder al de la clase padre con el siguiente prefijo super.<método>(<parámetros>). Para llamar al constructor del padre super.(<parámetros>).

El constructor de la clase hija suele comenzar llamando al constructor de la clase padre, para ello escribiremos como primera línea de código: super.<método>(<parámetros>);

En el siguiente ejemplo se muestra una clase que hereda de Complejo. Esta nueva clase dispone del nuevo atributo esReal que nos indicará si este número pertenece a los números reales, es decir, la parte imaginaria es igual a cero:

La clase que acabamos de ver no tiene acceso a los atributos de la clase padre, dado que estos han sido declarados con private. Recuerda que si quieres dejar que las clases que hereden de ti tengan acceso a estos atributos, pero no así el resto de las clases, has de declararlos como protected. Dado que el acceso directo a los atributos es más eficiente que invocar los métodos getters y setters, puede resultar interesante declarar los atributos como protected. Aunque claro, en este caso la modificación de los atributos de la clase padre obligará a modificar todos los herederos que accedan directamente a estos atributos.

```
class ComplejoAmpliado extends Complejo {  
    private boolean esReal;  
    public ComplejoAmpliado(double re, double im) {  
        super(re, im);  
        esReal = im == 0;  
    }  
    public ComplejoAmpliado(double re) {  
        super(re, 0);  
        esReal = true;  
    }  
    @Override  
    public void suma(Complejo v) {  
        esReal = getImaginario() == - v.getImaginario();  
        super.suma(v);  
    }  
    @Override  
    public String toString() {  
        if(esReal) {  
            return getReal() + " ¡real!";  
        } else {  
            return super.toString();  
        }  
    }  
    public boolean esReal(){  
        return esReal;  
    }  
}
```

Ejercicio paso a paso: La clase ComplejoAmpliado

1. Dentro del proyecto Complejos, crea la clase ComplejoAmpliado.
2. Copia el código que aparece en el ejemplo anterior.
3. Modifica la clase Principal para que en lugar de crear dos objetos como Complejo lo haga como ComplejoAmpliado.. Verifica su funcionamiento.
4. Modifica la suma para que el resultado sea un número real.
5. Abre la clase Complejo y modifica la visibilidad de los atributos reemplazando private por protected.
6. En la clase reemplaza la línea:

```
esReal = getImaginario() == - v.getImaginario();
```

por:

```
esReal = imaginario == - v.imaginario;
```

7. Verifica que no se ha introducido ningún error y que podemos ejecutar el proyecto.

Practica: Nueva clase coordenadas geográficas con altura

Crea la clase GeoPuntoAlt que herede de GeoPunto de forma que además de la longitud y latitud se almacena la altura de un punto geográfico. Para calcular la distancia teniendo en cuenta la diferencia de alturas, si suponemos que las distancias son cortas, podemos despreciar la curvatura de la tierra. De esta forma podemos aplicar Pitágoras para calcular la nueva distancia. Es decir:

$$\text{distancia_nueva} = \sqrt{\text{distancia_anterior}^2 + \text{diferencia_alturas}^2}$$

La sobrecarga en Java

Objetivos:

- Introducir el concepto de sobrecarga en Java
-

NOTA: Este apartado no es esencial para el desarrollo del curso. Si no dispones de tiempo puedes saltártelo.

Java permite varias versiones de un mismo método, siempre que cambien entre sí los parámetros. Es decir dispondremos de dos métodos distintos con un mismo nombre. Java los diferenciará al tener parámetros diferentes. Esta forma de trabajar se conoce como sobrecarga.

Ejercicio paso a paso: Sobre carga en la clase Complejos

1. En el proyecto Complejos, abre la clase Complejo.
2. Añade los siguientes métodos. Observa como su nombre coincide con un método ya existente.

```
public void suma(double re, double im) {  
    real = real + re;  
    imaginario = imaginario + im;  
}
```

```
public void suma(double re) {  
    real = real + re;  
}
```

3. Modifica la clase Principal para que se usen estos métodos.

Practica: Nuevo constructor en la clase coordenadas geográficas

Modifica la clase GeoPunto para que disponga de un nuevo constructor. Este ha de permitir inicializar el objeto a partir de dos enteros que representen millonésimas de grado.

El polimorfismo en Java

Video [El Polimorfismo en Java](#)

Objetivos:

- Introducir la clase Object y repasar el concepto de herencia
- Definir el concepto de polimorfismo

La clase Object

En las clases que hemos creado al principio de este módulo (Complejo y GeoPunto) no se utilizaba la palabra extends para heredar de otra clase. Sin embargo, en estos casos Java crea la nueva clase heredando de la clase Object. La clase Object es la raíz del árbol genealógico de herencias en Java. Esta clase dispone de una serie de métodos que por tanto van a estar disponibles en todos los objetos que creamos. Entre estos métodos se incluye: `toString()`, `getClass()` o `equals(Object o)`.

Un objeto de la clase hija también lo es de la clase padre y de todos sus antecesores. Por ejemplo, un objeto de la clase ComplejoAmpliado también lo será de la clase Complejo y también de Object. De hecho todo objeto en Java será necesariamente de la clase Object.

Esta característica que acabamos de comentar es una propiedad de la herencia, en este apartado vamos a estudiar otro aspecto importante de la programación orientada a objetos, el polimorfismo.

Polimorfismo

El polimorfismo consiste en declarar un objeto de una clase, pero instanciarlo como un descendiente de dicha clase (lo contrario no es posible). Es decir, utilizaremos la siguiente expresión:

```
<Clase_padre> <objeto> = new <Clase_hija>(<parámetros>);
```

A primera vista no parece demasiado útil, pero en la práctica va a tener una gran potencia. Para explicarlo utilizaremos un ejemplo real. Cuando estemos desarrollando una aplicación Android y queremos introducir un botón en el interfaz de usuario, vamos a poder indicar un gráfico de fondo. Para indicar este fondo tendremos que indicar un objeto de la clase Drawable. La clase Drawable representa algo que se puede dibujar, tiene muchos descendientes: BitmapDrawable (un fichero PNG o JPG), ShapeDrawable (un gráfico a partir de primitivas vectoriales), GradientDrawable (degradado de color), AnimationDrawable (animaciones frame a frame), ... La

clase Drawable no permite declarar un objeto directamente de esta clase. Un Drawable es algo que se puede dibujar pero que no sabemos cómo. Un objeto real ha de declararse de algunos de sus descendientes, que si saben cómo dibujarse. Esto da mucha potencia a la hora de escoger el tipo de fondo para nuestro botón, podemos usar un fichero PNG, un degradado, una animación,... Y más todavía, podríamos crear nuestra propia clase, AnimacionLucesNavidad, que descenda de AnimationDrawable. Un objeto de esta clase también lo sería de Drawable y por lo tanto vamos a poder utilizarlo como fondo de nuestro botón.

Si reflexionamos sobre lo que hemos conseguido gracias al polimorfismo veremos que no es poca cosa. Los diseñadores de la clase botón (realmente se llama Button) pudieron tomar una decisión de diseño, concretamente cómo podríamos dibujar el fondo del botón, sin la necesidad de especificar exactamente cómo hacerlo. Más adelante hemos podido diseñar nuevas clases para dibujar este fondo simplemente creando descendientes de la clase Drawable.

Conversión de tipos

Imaginemos que hemos declarado un objeto de una clase pero lo hemos construido utilizando un constructor de una clase descendiente, tal y como se muestra a continuación:

```
Complejo c = new ComplejoAmpliado(12.4);
```

Vamos a poder utilizar cualquier método de c definido en la clase Complejo o cualquiera de sus descendientes:

```
c.toString();  
c.getClass();
```

Pero la siguiente método no podrá ser utilizado:

```
c.esReal()..
```

Esto es debido a que este método no está definido para la clase Complejo. No obstante, estamos seguros de que c es de la clase ComplejoAmpliado y por lo tanto este método sí que puede ser llamado. Para solucionar este problema, podemos utilizar lo que en Java se conoce como typecast (cambio de tipo).

```
(<Clase>) <objeto>
```

A partir de este momento <objeto> será considerado de la clase <Clase>. Aunque no haya sido declarada de esta clase. Por lo tanto podemos escribir:

```
((ComplejoAmpliado)c).esReal()  
Uso de instanceof
```

Podemos averiguar si un objeto es de una determinada clase usando la siguiente expresión:

```
<objeto> instanceof <Clase>
```

A continuación se muestran algunos ejemplos:

```
Complejo c = new ComplejoAmpliado(12.4);
if (c instanceof Object)... //siempre true
if (c instanceof Complejo)... //true
if (c instanceof ComplejoAmpliado)... //true
if (((ComplejoAmpliado)c).esReal())...
```

Ejercicio paso a paso: Polimorfismo con la clase Complejo

1. Abre la clase Principal del proyecto Complejos.
2. Reemplaza el código del método main() por el siguiente:

```
Complejo lista[] = new Complejo[4];
lista[0] = new Complejo(-1.5, 3.0);
lista[1] = new Complejo(-1.2, -3.0);
lista[2] = new ComplejoAmpliado(-1.5, 3.0);
lista[3] = new ComplejoAmpliado(-1.2, 0);
for (int i = 0; i < lista.length; i++) {
    System.out.println("Complejo: " + lista[i]);
}
```

Observa como en la primera línea definimos un array de 4 complejos. En las siguientes líneas inicializamos las 4 posiciones del array, las 2 primeras con objetos de la clase Complejo y las dos siguientes con objetos de la clase ComplejoAmpliado. Finalmente, realizamos un bucle para todos los elementos del array mostrando en la consola el valor de cada complejo.

3. Trata de resolver la siguiente pregunta: Recuerda que mostrar un complejo supone una llamada al método `toString()` que ha sido definido de manera diferente en la clase `Complejo` y en `ComplejoAmpliado`. Teniendo en cuenta que los elementos del array han sido declarados como `Complejo`, pero algunos han sido creados como `Complejo` y otros como `ComplejoAmpliado`, Cuando tenga que llamar a `toString()` ¿A cuál de los dos métodos se llamará?

4. Para salir de dudas pulsa el botón Ejecución y observa el resultado:

Complejo: -1.5+3.0i

Complejo: -1.2+-3.0i

Complejo: -1.5+3.0i

Complejo: -1.2 ¡Real!

Uno podría pensar que siempre se va a llamar al método `toString()` definido en `Complejo`. Pero, el último resultado nos indica que no siempre es así, si no que depende de la clase usada en el constructor.

5. Reemplaza el bucle for por el siguiente:

```
for(int i = 0; i < lista.length; i++) {  
    System.out.println("Complejo: " + lista[i]);  
    if(lista[i] instanceof ComplejoAmpliado){  
        System.out.println(" esReal=" + ((ComplejoAmpliado) lista[i]).esReal());  
    }  
}
```

Observa como hemos añadido un if que verifica si el objeto es de la clase ComplejoAmpliado. En caso afirmativo se le aplica un typecast al objeto para considerarlo de esta clase y así poder llamar al método esReal(). El resultado es mostrado en la consola.

6. Ejecuta el proyecto y observa el resultado:

Complejo: -1.5+3.0i

Complejo: -1.2+-3.0i

Complejo: -1.5+3.0i

esReal=false

Complejo: -1.2 ¡Real!

esReal=true

Practica: Polimorfismo con la clase Object

Crea un array de 4 objetos de la clase Object. Construye el primero como Complejo y el segundo como Geopunto y los dos últimos como GeoPuntoAlt. Implementa un bucle que recorra el array mostrando los objetos. En caso de tratarse de un punto geográfico con una altitud superior a 1000 m, muestra un mensaje con el texto “punto elevado”.

Clases abstractas, interfaces y herencia múltiple

Objetivos:

- Definir las clases abstractas.
- Definir el concepto de interfaz.
- Introducir la herencia múltiple.

Clases abstractas

Una clase abstracta es aquella que no implementa alguno de sus métodos. Veamos un ejemplo:

```
abstract class Figura {  
    protected double x;  
    protected double y;  
  
    public Figura(double x, double y){  
        this.x = x;  
        this.y = y;  
    }  
    abstract double perimetro();  
}
```

Observa como el método `perimetro()` de esta clase no se implementa por lo que hay que utilizar la palabra reservada `abstract`. Este método se dice que es abstracto. Toda clase que contenga algún método abstracto también es abstracta, e igualmente, hay que etiquetarla con la palabra reservada `abstract`.

No se puede instanciar un objeto de una clase abstracta, dado que no se ha definido todo su comportamiento.

Una clase abstracta solo se utiliza para definir subclases. Veamos un ejemplo:

```
class Circulo extends Figura {  
    private double radio;  
    public Circulo(double x, double y, double radio){  
        super(x,y);  
        this.radio = radio;  
    }  
    public double perimetro(){  
        return 2*Matth.PI*radio;  
    }  
}
```

Se puede crear una referencia a una clase abstracta:

```
Figura miFigura;
```

Pero no se puede crea un objeto de una clase abstracta:

Podemos asignar cualquier objeto descendiente a una referencia de una clase abstracta, siempre que el descendiente no sea abstracto:

```
miFigura = new Circulo(100,100,50);  
Recuerda que este tipo de operaciones se conoce como polimorfismo.
```

Interfaces

La interfaz es una clase abstracta pura, es decir, una clase donde se indican los métodos, pero no se implementa ninguno. Permite al programador establecer una estructura que ha de seguir toda clase que implemente esta interfaz. En la interfaz solo se indica los nombres de los métodos, sus parámetros y tipos que retornan, pero no el código de cada método. Una interfaz también puede contener constantes, pero nunca pude campos de otro tipo. **En Java** estos campos serán de tipo static y final. **En Kotlin** estos campos se declaran dentro de un companion object. Veamos un ejemplo:

```
public interface Dibujable  
{  
    int AZUL = 0x0000FF;  
    void dibuja(int color);  
    void borra();  
    boolean estaDibujado();  
}
```

Aunque **en Java** no se haya indicado, AZUL es declarado de tipo static y final. Es decir, una interfaz puede definir constantes, pero nunca encapsular datos.

Una clase puede implementar una interface. En este caso está obligada a definir todos los métodos indicados. Veamos un ejemplo:

```
public class Dibujo implements Dibujable {  
    ...  
    void dibuja(int color){  
        ...  
    }  
    void borra(){  
        ...  
    }  
    boolean estaDibujado(){  
        ...  
    }  
}
```

La clase Dibujo implementa la interface, por lo que ha de declarar todos los métodos y escribir el código correspondiente.

NOTA: En Kotlin y Java 8 una interface puede contener métodos estáticos.

Herencia múltiple

Algunos lenguajes orientados a objeto, como C++, Perl o Python, soportan la herencia múltiple. Es decir, pueden extender simultáneamente más de una clase. Esta característica presenta cierta ambigüedad a la hora de acceder a los atributos de los padres, lo que hace que aumente la complejidad del lenguaje.

Otros lenguajes, como Java, C#, Delphi, Objective-C o Kotlin, no soporta herencia múltiple. Por lo que solo pueden extender de una clase; lo que simplifica en gran medida el lenguaje. Lo que sí que van a permitir estos lenguajes es que una clase además de extender de su padre puedan implementar varios interfaces.

La implementación de una interface resulta mucho más sencilla dado que no incorpora variables, solo obliga a la clase a definir una lista de métodos. Veamos un ejemplo:

```
class Circulo extends Figura implements Dibujable {  
    ...  
}
```

En este caso Circulo extiende Figura; por lo que va a incorporar las variables x e y además del método abstracto perímetro(). Además, implementa Dibujable por lo que ha de definir los tres métodos de esta interfaz.

De esta forma la clase Circulo tiene un doble comportamiento el heredado de Figura, además nos aseguramos que puede ser dibujado al implementar Dibujable:

Ejercicio paso a paso: La interfaz RepositorioLugares

En este ejercicio vamos a crear una interfaz que nos permita almacenar una lista de objetos *Lugar*. A lo largo del curso esta interfaz será implementada por dos clases. En esta unidad usaremos una lista almacenada en memoria y en la última unidad una base de datos. Usar esta interface nos va a permitir desacoplar la forma en la que almacenamos los datos del resto de la aplicación. Por ejemplo, si en un futuro queremos que los datos se almacenen en la nube, solo será necesario cambiar la implementación de esta interface, dejando idéntica el resto de la aplicación.

1. Dentro del explorador del proyecto *mislugares* > *java* > *com.example.mislugares*, pulsa con el botón derecho y selecciona *New* > *Java Class* o *New* > *Java Class*.
2. Introduce en la nueva ventana en *Name*: *RepositorioLugares*, y en *Kind*: *Interface*.
3. Reemplaza el código por el siguiente (dejando la línea del package):

```
public interface RepositorioLugares {  
    Lugar elemento(int id); //Devuelve el elemento dado su id  
    void anyade(Lugar lugar); //Añade el elemento indicado  
    int nuevo(); //Añade un elemento en blanco y devuelve su id  
    void borrar(int id); //Elimina el elemento con el id indicado  
    int tamanyo(); //Devuelve el número de elementos  
    void actualiza(int id, Lugar lugar); //Reemplaza un elemento  
}
```

Una clase que implemente esta interface va a almacenar una lista de objetos de tipo Lugar. Mediante los métodos indicados vamos a poder acceder y modificar esta lista. Una interfaz también puede tener funciones estáticas, como `añadeEjemplos()`. En Java solo está permitido con API mínima >24, por lo que lo añadiremos esta función en una clase no abstracta.

4. Esta interface será usada en uno de los siguientes apartados.

Enumerados: La clase TipoLugar

video [Tipo enumerados en Java](#)

Objetivos:

- Aprender a utilizar colecciones en Java
- Crear la clase Lugares que utilizaremos en la aplicación *Mis Lugares*

Enumerados: La clase TipoLugar

Marcar esta página

Ejercicio paso a paso: El enumerado *TipoLugar*

En este ejercicio vamos a crear un tipo enumerado para diferenciar entre diferentes tipos de establecimientos en la aplicación *Mis Lugares*. Además, a cada tipo de lugar le asociaremos un String con el nombre y un recurso gráfico.

1. Vamos a crear un nuevo tipo enumerado. Para ello pulsa con el botón derecho sobre com.example.mislugares. Selecciona New > Java Class e introduce en Name: TipoLugar en Kind: Enum. y pulsa OK.
2. Reemplaza el código por el siguiente, (dejando la línea de package).

```
enum class TipoLugar private constructor(val texto:String, val recurso:Int)  
{  
    OTROS("Otros", 5),  
    RESTAURANTE("Restaurante", 2),  
    BAR("Bar", 6),  
    COPAS("Copas", 0),  
    ESPECTACULO("Espectáculo", 0),  
    HOTEL("Hotel", 0),  
    COMPRAS("Compras", 0),  
    EDUCACION("Educación", 0),  
    DEPORTE("Deporte", 0),  
    NATURALEZA("Naturaleza", 0),  
    GASOLINERA("Gasolinera", 0)  
}
```

Si quieres puedes definir otros tipos de lugares para adaptar la aplicación a tus necesidades. Observa como a cada tipo le asociamos un String con el nombre del tipo de lugar y un entero. El entero se utilizará más adelante para indicar un recurso gráfico en Android con un ícono representativo del tipo.

- 3 Añade el siguiente atributo a la clase:

```
private TipoLugar tipo;
```

4. Añade el parámetro TipoLugar en el constructor de la clase e inicializa el atributo anterior con este parámetro:

```
public Lugar(String nombre, String direccion, double longitud,
```

```
        double latitud, TipoLugar tipo, int telefono,  
        String url, String comentario, int valoracion) {  
    this.tipo = tipo;  
    ...  
}
```

5. Añade los métodos getter y setter correspondientes. Para ello pulsa con el botón derecho y seleccionar en la opción *Generate > Getters and Setters*.

```
public TipoLugar getTipo() {  
    return tipo;  
}  
  
public void setTipo(TipoLugar tipo) {  
    this.tipo = tipo;  
}
```

6. Vamos a volver a generar el método *toString()*. Para ello pulsa con el botón derecho y seleccionar la opción *Generate > toString()*. Pulsa en *Yes* para reemplazar el método actual.
7. Abre la clase Principal y modifica la inicialización del objeto para que se incluya el nuevo parámetro (*TipoLugar.EDUCACION*) en el constructor.
8. Verifica el resultado ejecutando el proyecto.

Las colecciones en Java

video [Las Colecciones en Java](#) Video [Estructuras de datos](#)

Objetivos:

- Aprender a utilizar colecciones en Java
- Crear la clase Lugares que utilizaremos en la aplicación *Mis Lugares*

En este apartado vamos a seguir añadiendo código Java, que luego nos será de utilidad en la aplicación Mis Lugares. En concreto vamos a crear la clase ListaLugares que tiene como finalidad almacenar y gestionar un conjunto de objetos Lugar de interés para el usuario. La clase ListaLugares va a utilizar un tipo de colección para almacenar los lugares. Antes de comenzar, es interesante introducir la poderosa herramienta que nos ofrece las colecciones.

El API de Java nos proporciona el framework de las colecciones, que nos permite utilizar diferentes estructuras de datos para almacenar y recuperar objetos de cualquier clase. Dichas colecciones no forman parte del lenguaje, sino que son clases definidas en el paquete `java.util`. Para crear una colección usaremos la siguiente estructura:

```
Coleccion<Clase> nombre = new Coleccion<Clase>();
```

Donde `Coleccion` es una clase de este framework que queramos utilizar según la estructura de almacenamiento que nos interese y `Clase` representa el tipo de datos a almacenar. Por ejemplo, para crear una lista ordenada de objetos de la clase `String` escribiríamos:

```
ArrayList<String> listaNombres = new ArrayList<String>();
```

En Java podemos trabajar con clases genéricas, para ello se utiliza `<y>`. A este mecanismo se le conoce como genericidad. Para saber más sobre este aspecto recomendamos ver el siguiente Polimedia: <http://youtu.be/N3yy2pfUaE0>.

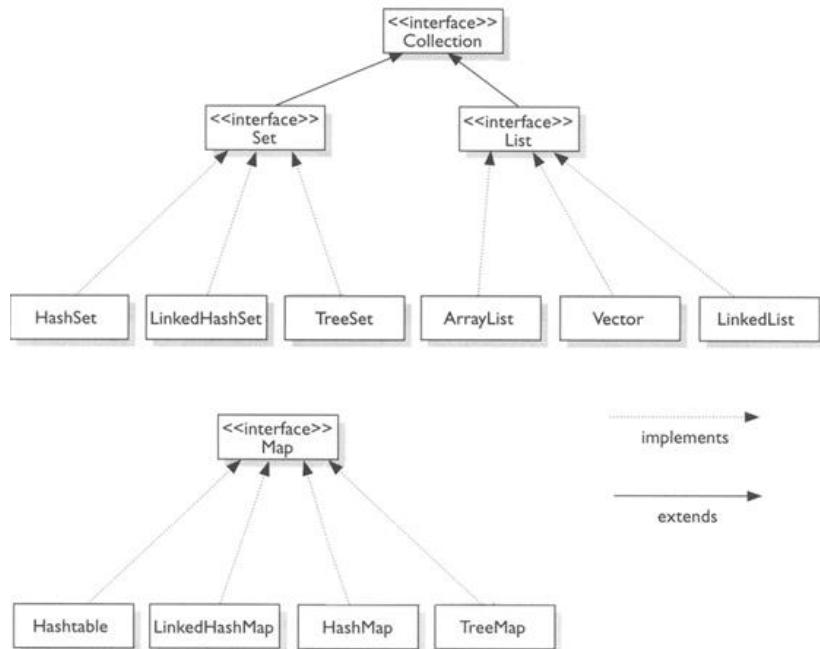
Hay tres tipos de colecciones, cada uno con un interfaz común y diferentes implementaciones. Las diferentes implementaciones de un mismo interfaz realizan la misma tarea aunque la diferencia está en que unas implementaciones son más rápidas en algunas operaciones y más lentas en otras:

Conjunto – los elementos no tienen un orden y no se permiten duplicados. Se define el interfaz Set<E>. Podemos utilizar las siguientes implementaciones: HashSet<E> (implementación con tabla hash), LinkedHashSet<E> (tabla hash +doble lista enlazada), TreeSet<E> (implementación con árbol)

Listas – estructura secuencial, donde cada elemento tiene un índice o posición. Se utiliza el interfaz List<E>. Podemos utilizar las siguientes implementaciones: ArrayList<E> (acceso rápido), LinkedList<E>(inserciones/borrado rápidas), Stack<E> (pila), Vector<E> (obsoleto)

Diccionario o Matriz asociativa – cada elemento tiene asociado una clave que usaremos para recuperarlo. Se utiliza el interfaz Map<K,V>. Podemos utilizar las siguientes implementaciones: HashMap<K,V>, TreeMap<K,V>, LinkedHashMap<K,V>

En el siguiente diagrama se muestra las relaciones de herencia entre los interfaces y clases más importantes en el framework de las colecciones.



Como puede verse en el esquema anterior los interfaces List<E> y Set<E> heredan del interface Collection<E>. A continuación se enumeran los métodos comunes a los interfaces List<E> y Set<E>, que son recogidos en el interface Collection. Supondremos que el elemento e es un objeto de la clase E:

boolean add(E e): Añade un nuevo elemento al final de la lista.

boolean remove(E e): Elimina la primera ocurrencia del elemento indicado.

boolean contains(E e): Comprueba si el elemento especificado está en la colección.

void clear(): Elimina todos los elementos de la colección.

int size(): Devuelve el número de elementos en la colección.

boolean isEmpty(Collection<?> c): Comprueba si la colección está vacía.

Los siguientes métodos combinan dos colecciones:

boolean addAll(Collection<?> c) : Añade todos los elementos de la colección c.

boolean removeAll(Collection<?> c): Elimina todos los elementos de la colección c.

boolean containsAll(Collection<?> c): Comprueba si coinciden las colecciones.

boolean retainAll(Collection<?> c): Elimina todos los elementos a no ser que estén en c. (obtiene la intersección).

Conjuntos

Los conjuntos son estructuras de datos donde los elementos no tienen un orden y no se permiten duplicados. Para definirlos se utiliza la interfaz Set<E>, que no añade nuevos métodos a la interfaz Collection<E>. Por lo tanto, con los métodos anteriores podrás manipular tus conjuntos.

Podemos utilizar diferentes implementaciones. La más eficiente es HashSet<E> (implementación con tabla hash). Pero si queremos que los elementos queden ordenados podemos usar TreeSet<E> (implementación con árbol).

El siguiente ejemplo muestra cómo crear un conjunto de Strings y luego recorrerlo para mostrarlo en consola:

```
Set<String> conjunto = new HashSet();
conjunto.add("manzana");
conjunto.add("pera");
conjunto.add("fresa");
conjunto.add("naranja");
conjunto.remove("pera");
for(String s : conjunto) {
    System.out.println(s);
}
```

Ejercicio paso a paso: La colección Set<E>

1. Crea un nuevo proyecto con nombre Colecciones.
2. Crea una nueva clase con nombre Principal.
3. Crea el siguiente método en la clase:

```
public static void main(String[] main) {}
```

4. Copia dentro del método el código que se muestra en el ejemplo anterior.
5. Pulsa Alt-Intro en Android para que se añadan los import de las clases utilizadas.
6. Ejecuta y verifica el resultado.
7. Añade más elementos al conjunto, alguno dos veces y vuelve a ejecutar. Has de observar como el orden en que se muestra los elementos del conjunto no coincide con el orden en que son insertados ni con el orden alfabético.
8. Reemplaza en la primera línea del código la clase que usamos para implementar el conjunto; en lugar de HashSet introduce TreeSet.
9. Pulsa Alt-Intro en Android Studio o Shift-Ctrl-O en Eclipse y ejecuta de nuevo. Observar como ahora se muestran los elementos en orden alfabético.
10. Prueba otros métodos del interface Collection<E>.

Listas

Una lista es una estructura secuencial, donde cada elemento tiene un índice o posición. También recibe el nombre de array o vector unidimensional. El índice de una lista es siempre un entero y el primer elemento ocupa la posición 0. Para trabajar con ellas se utiliza el interfaz List<E>. Las implementaciones más recomendables son: ArrayList<E> si queremos acceder a una posición de forma muy rápida o LinkedList<E> si queremos inserciones y borrado muy rápidos.

La interfaz List<E> hereda todos los métodos de Collection<E> y añade los siguientes:

boolean add(int indice, E e): Inserta un nuevo elemento en una posición. El elemento que estaba en esta posición y los siguientes pararán a la siguiente.

E get(int indice): Devuelve el elemento en la posición especificada.

int indexOf(E e): Primera posición en la que se encuentra un elemento; -1 si no está.

int lastIndexOf(E e): Última posición del el elemento especificado; o -1 si no está.

E remove(int indice): Elimina el elemento de la posición indicada.

E set(int indice, E e): Pone un nuevo elemento en la posición indicada. Devuelve el elemento que se encontraba en dicha posición anteriormente.

El siguiente ejemplo muestra como crear una lista de complejos y luego recorrerla:

```
List<Complejo> lista = new ArrayList<Complejo>();  
lista.add( new Complejo(1.0, 5.0) );  
lista.add( new Complejo(2.0, 4.2) );  
lista.add(1, new Complejo(3.0, 0.0) );lista.remove(0);  
for(Complejo c: lista) {
```

```
        System.out.println(c);
    }
```

Observa como las dos primeras llamadas al método add() no se indica en qué posición de la lista se inserta. En estos casos se inserta al final de la lista. La tercera llamada se inserta en la posición 1, el elemento en esta posición y los siguientes son desplazados hacia atrás.

Como puedes ver la diferencia entre un conjunto y una lista es que en el conjunto los elementos no tienen una posición asignada; solo podemos ver si están o no están. Mientras que en la lista los elementos están ordenados según su posición. Por lo tanto, podemos insertar, consultar, eliminar o reemplazar elementos de una posición determinada.

Practica:La colección List<E>

1. Abre el proyecto Colecciones y añádele la clase Complejo.
2. Reemplaza el código del método main() de Principal para seguir el ejemplo anterior.
3. Pulsa Shift-Ctrl-O para que se añadan los import de las clases utilizadas.
4. Prueba otros métodos del interface List<E>.

Diccionarios o Mapas

Nota: Este apartado no es imprescindible para el curso. Puedes saltártelo.

Los diccionarios (también conocidos como mapas o matrices asociativas) son estructuras de datos donde cada elemento tiene asociado una clave que usaremos para recuperarlo (en lugar del índice de una lista). Para definirlos se utiliza la interfaz Map<K,V>. En este caso se trabaja con dos clases una que se utiliza como clave (K) y otra para almacenar los valores (V). La idea es que cada elemento se almacena mediante un par de objetos (K,V). Esta estructura de datos nos permite obtener el objeto V muy rápidamente, a partir de su clave K. Por ejemplo, podríamos almacenar objetos de la clase Vehiculo y utilizar como clave su matrícula en un String. De esta forma, a partir de la matrícula un diccionario encontraría el vehículo asociado muy rápidamente.

Podemos utilizar las siguientes implementaciones de este interfaz:

HashMap<K,V>,TreeMap<K,V>,LinkedHashMap<K,V>

Veamos los métodos del interfaz Map<K,V> (Ojo. A diferencia de otras colecciones no hereda del interfaz Collection<E>).

V put(K key, V value): Añade un nuevo par clave-valor al diccionario.

V get(Object key): Da el valor asociado a una clave o null si no se encontró.

V remove(Object key): Elimina el par clave-valor que corresponde a la clave.

boolean containsKey(Object key): Comprueba si está la clave especificada.

boolean containsValue(Object value): Comprueba si está el valor.

Set keySet(): Devuelve un conjunto con las claves contenidas en el diccionario.

Collection values(): Devuelve una colección con solo los valores.

boolean isEmpty(): Comprueba si la colección está vacía.

int size(): Devuelve el número de elementos que contiene la colección.

void clear(): Elimina todos los elementos de la colección.

El siguiente ejemplo muestra como crear un diccionario para almacenar objetos de la clase Vehiculo utilizando como clave un String con la matrícula:

```
Map diccionario = new HashMap();
diccionario.put("V 1245", new Vehiculo());
diccionario.put("A 2455", new Vehiculo());
diccionario.get("V 1245");
```

La clase estática Collections nos ofrece herramientas para ordenar y buscar en colecciones. Los interfaces Iterator y ListIterator facilitan recorres colecciones para hacer bucles.

La clase ListaLugares

Ejercicio paso a paso: La clase LugaresLista

En este ejercicio vamos a crear la clase LugaresLista que tiene como finalidad almacenar y gestionar un conjunto de objetos Lugar dentro de una lista.

1. Dentro del paquete org.example.mislugares añade la clase LugaresLista, y reemplaza el código por el siguiente:

```
public class LugaresLista implements RepositorioLugares
{
    protected List listaLugares;
    public LugaresLista()
    {
        this.listaLugares = new ArrayList<Lugar>();
        anyadeEjemplos();

    }
    @Override
    public Lugar elemento(int id)
    {
        return (Lugar) listaLugares.get(id);
    }
    @Override
    public void anyade(Lugar lugar)
    {
        listaLugares.add(lugar);
    }
    @Override
    public int nuevo()
    {
        Lugar lugar = new Lugar();
        listaLugares.add(lugar);
        return listaLugares.size()-1;
    }
    @Override
    public void borrar(int id)
    {
        listaLugares.remove(id);
    }
    @Override
    public int tamanyo()
    {
        return listaLugares.size();
    }
    @Override
    public void actualiza(int id, Lugar lugar)
    {
        listaLugares.set(id, lugar);
    }
    public void anyadeEjemplos() {
        anyade(new Lugar("Escuela Politécnica Superior de Gandia",
                        "C/ Paraninfo, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
                        TipoLugar.EDUCACION, 962849300, "http://www.epsg.upv.es",
                        "Uno de los mejores lugares para formarse.", 3));
        anyade(new Lugar("Al de siempre",
                        "Calle de la Constitución, 1 46003 Valencia (SPAIN)", 0.170893, 39.995656,
                        TipoLugar.CIUDAD, 962849300, "http://www.epsg.upv.es",
                        "El mejor lugar para descansar.", 3));
    }
}
```

```

        "P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)",
        -0.190642, 38.925857, TipoLugar.BAR, 636472405, "",
        "No te pierdas el arroz en calabaza.", 3));
anyade(new Lugar("androidcurso.com",
        "ciberespacio", 0.0, 0.0, TipoLugar.EDUCACION,
        962849300, "http://androidcurso.com",
        "Amplia tus conocimientos sobre Android.", 5));
anyade(new Lugar("Barranco del Infierno",
        "Vía Verde del río Serpis. Villalonga (Valencia)",
        -0.295058, 38.867180, TipoLugar.NATURALEZA, 0,
        "http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-"+
        "verde-del-rio.html","Espectacular ruta para bici o andar", 4));
anyade(new Lugar("La Vital",
        "Avda. de La Vital, 0 46701 Gandía (Valencia)", -0.1720092,
        38.9705949, TipoLugar.COMPRAS, 962881070,
        "http://www.lavital.es/", "El típico centro comercial", 2));
}

}

```

- Pulsa *Alt+Intro* e para que se añadan los import de las clases utilizadas.

```
import java.util.ArrayList;
import java.util.List;
```

- Para Java** Añade la siguiente sobrecarga al constructor a la clase Lugar:

```

public Lugar() {
    fecha = System.currentTimeMillis();
    posicion = GeoPunto.SIN_POSICION;
    tipo = TipoLugar.OTROS;
}

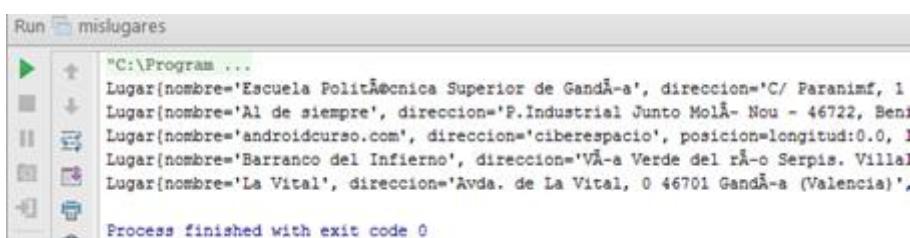
```

Esto nos permitirá crear un nuevo lugar sin indicar sus atributos.

- Abre la clase Principal y reemplaza el código del método main() por:

```
Lugares lugares = new LugaresVector();
lugares.anyadeEjemplos();
for (int i=0; i<lugares.tamanyo(); i++) {
    System.out.println(lugares.elemento(i).toString());
}
```

- Verifica que el resultado es similar al siguiente:



Unidad 2 Android

Introducción a las plataformas para móviles

video [Introducción a la plataforma para móviles Android](#)

Introducción

La telefonía móvil está cambiando la sociedad actual de una forma tan significativa como lo ha hecho Internet. Esta revolución no ha hecho más que empezar; los nuevos terminales ofrecen unas capacidades similares a un ordenador personal, lo que permite que puedan ser utilizados para leer el correo o navegar por Internet. Pero, a diferencia de un ordenador, un teléfono móvil siempre está en el bolsillo del usuario. Esto permite un nuevo abanico de aplicaciones mucho más cercanas al usuario. De hecho, muchos autores coinciden en afirmar que el nuevo ordenador personal del siglo XXI será un terminal móvil.

El lanzamiento de Android como nueva plataforma para el desarrollo de aplicaciones móviles ha causado una gran expectación y está teniendo una importante aceptación tanto por parte de los usuarios como por parte de la industria. En la actualidad se ha convertido en la alternativa dominante frente a otras plataformas como iPhone o Windows Phone.

A lo largo de este capítulo veremos las características de Android que lo hacen diferente de sus competidores. Se explicará también cómo instalar y trabajar con el entorno de desarrollo (Android Studio).

Objetivos

Conocer las características de Android, destacando los aspectos que lo hacen diferente de sus competidores.

Estudiar la arquitectura interna de Android.

Aprender a instalar y trabajar con el entorno de desarrollo (Android SDK).

Enumerar las principales versiones de Android y aprender a elegir la más idónea para desarrollar nuestras aplicaciones.

Crear una primera aplicación y estudiar su estructura de un proyecto en Android.

Conocer donde podemos conseguir documentación sobre Android.

Aprender a utilizar las herramientas disponibles para detectar errores en el código.

Como hemos comentado, existen muchas plataformas para móviles (Apple iOS, Windows Phone, BlackBerry, Palm, Java Micro Edition, Linux Mobile (LiMo, Firefox OS, etc.); sin embargo, Android presenta una serie de características que lo hacen diferente. Es el primero que combina en una misma solución las siguientes cualidades:

***Plataforma abierta.** Es una **plataforma de desarrollo libre basada en Linux** y de código abierto. Una de sus grandes ventajas es que se puede usar y customizar el sistema sin pagar royalties.

***Adaptable a diversos tipos de hardware.** Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día podemos encontrar relojes, gafas, cámaras, TV, sistema para automóviles, electrodomésticos y una gran variedad de sistemas empotrados que se basan en este sistema operativo, lo cual tiene sus evidentes ventajas, pero también va a suponer un esfuerzo adicional para el programador. La aplicación ha de funcionar correctamente en dispositivos con una gran variedad de tipos de entrada, pantalla, memoria, etc. Esta característica contrasta con la estrategia de Apple: en iOS tenemos que desarrollar una aplicación para iPhone y otra diferente para iPad.

***Portabilidad asegurada.** Las aplicaciones finales son desarrolladas en **Java**, lo que nos asegura que podrán ser ejecutadas en **cualquier tipo de CPU**, tanto presente como futuro. Esto se consigue gracias al concepto de máquina virtual.

*Arquitectura basada en componentes inspirados en Internet. Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un reloj de pantalla reducida o en un televisor.

*Filosofía de dispositivo **siempre conectado a Internet**. Muchas aplicaciones solo funcionan si disponemos de una conexión permanente a Internet. Por ejemplo, comunicaciones interpersonales o navegación con mapas.

***Gran cantidad de servicios incorporados.** Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.

***Aceptable nivel de seguridad.** Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja, que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.). Desde la versión 6.0 el usuario puede conceder o retirar permisos a las aplicaciones en cualquier momento.

***Optimizado para baja potencia y poca memoria.** En el diseño de Android se ha tenido en cuenta el hardware específico de los dispositivos móviles. Por ejemplo, **Android utiliza la máquina virtual ART (o Dalvik en versiones antiguas)**. Se trata de una implementación de Google de la máquina virtual **Java optimizada para dispositivos móviles**.

***Alta calidad de gráficos y sonido.** Gráficos vectoriales suavizados, animaciones, gráficos en 3D basados en OpenGL. Incorpora los codecs estándares más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC, etc.

Como hemos visto, Android combina características muy interesantes. No obstante, la pregunta del millón es: ¿se convertirá Android en el sistema operativo (SO) estándar para dispositivos móviles? Para contestar a esta pregunta habrá que ver la evolución del iPhone de Apple y cuál es la respuesta de Windows con el lanzamiento de su SO para móviles. No obstante, Android ha

alcanzado un 85% de cuota de mercado (90% en España), cosa que lo deja en una posición predominante que es difícil que pierda a corto plazo.

En conclusión, **Android nos ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para diferentes tipos de dispositivos**. A lo largo de este texto trataremos de mostrar de la forma más sencilla posible cómo conseguirlo.

Los orígenes

Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía, recién creada, orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (Dalvik VM).

En el año 2007 se crea el consorcio Handset Alliance[1] con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Una pieza clave de los objetivos de esta alianza es promover el diseño y difusión de la plataforma Android. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia Apache v2.0.

En noviembre de 2007 se lanza una primera versión del Android SDK. Al año siguiente aparece el primer móvil con Android (T-Mobile G1). En octubre, Google libera el código fuente de Android, principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes se abre Android Market, para la descarga de aplicaciones. En abril de 2009, Google lanza la versión 1.5 del SDK, que incorpora nuevas características como el teclado en pantalla. A finales de 2009 se lanza la versión 2.0 y a lo largo de 2010, las versiones 2.1, 2.2 y 2.3.

Durante el año 2010, Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos a iOS e incluso superando al sistema de Apple en EE.UU.

En el año 2011 se lanza la versión 3.x (Honeycomb), específica para tabletas, y la 4.0 (Ice Cream Sandwich), tanto para móviles como para tabletas. Durante ese año, Android se consolida como la plataforma para móviles más importante y alcanza una cuota de mercado superior al 50 %.

En 2012, Google cambia su estrategia en su tienda de descargas online, reemplazando Android Market por Google Play Store, donde en un solo portal unifica tanto la descarga de aplicaciones como la de contenidos. Ese año aparecen las versiones 4.1 y 4.2 (Jelly Bean). Android mantiene su espectacular crecimiento y alcanza, a finales de año, una cuota de mercado del 70 %.

En 2013 se lanzan las versiones 4.3 y 4.4 (KitKat). En 2014 se lanza la versión 5.0 (Lollipop). A finales de ese año, la cuota de mercado de Android llega al 85 %. En octubre de 2015 ha aparecido la versión 6.0, con el nombre de Marshmallow. En 2016 se lanzó la versión 7.0, Android Nougat. A finales de 2017 aparece la versión 8.0, con nombre Android Oreo. En agosto de 2018 se lanza la versión 9.0, Android Pie.

[1]<http://www.openhandsetalliance.com>

Las plataformas para móviles

Objetivos:

- Comparar las plataformas para móviles más utilizadas en la actualidad.

En este apartado vamos a describir las características de las principales plataformas móviles disponibles en la actualidad. Dado la gran cantidad de datos que se indican, hemos utilizado una tabla para representar la información. De esta forma resulta más sencillo comparar las plataformas.



	Apple iOS 12	Android 9.0	Windows Phone 10	BlackBerry 10
Compañía	Apple	Open Handset Alliance	Microsoft	BlackBerry
Núcleo del SO	Mac OS X	Linux	Windows NT	QNX
Licencia de software	Propietaria	Libre y abierto	Propietaria	Propietaria
Año de lanzamiento	2007	2008	2010	1999
Fabricante único	Sí	No	No	Sí
Variedad de dispositivos	Modelo único	Muy alta	Media	Baja
Soporte memoria externa	No	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit/Chromium (Blink)	Trident	WebKit
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry World
Número de aplicaciones*	2.400.000 (sept. 2016)	2.000.000 (jun. 2016)	700.000 (oct. 2016)	270.000 (2016)
Coste publicar	\$99 / año	\$25 una vez	\$99 / año	Sin coste
Otras tiendas sin supervisión	No	Sí	No	Sí
Familia CPU soportada	ARM	ARM, MIPS, x86	ARM	ARM
Máquina virtual	No	Dalvik / ART	.net	No
Lenguaje de programación	Objective-C, Swift	Java, C++, Kotlin	C#, Visual Basic, C++	C, C++, Java
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac
Varios usuarios	No	Sí	No	No
Modo invitado	Sí	Sí	No	No

Tabla 1: Comparativa de las principales plataformas móviles. (*fuente www.statista.com)

Otro aspecto fundamental a la hora de comparar las plataformas móviles es su cuota de mercado. En la siguiente gráfica podemos ver un estudio realizado por la empresa Gartner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Podemos destacar la desaparición de la plataforma

Symbian de Nokia, el declive continuo de BlackBerry, el estancamiento de la plataforma de Windows, que parece que no despega, y el afianzamiento de la cuota de mercado de Apple en torno al 15 %. En la gráfica se puede apreciar como Apple consigue anualmente un aumento significativo de ventas coincidiendo con el lanzamiento de un nuevo terminal. Finalmente, cabe señalar el espectacular ascenso de la plataforma Android, que en cinco años ha alcanzado una cuota de mercado superior al 80 %.

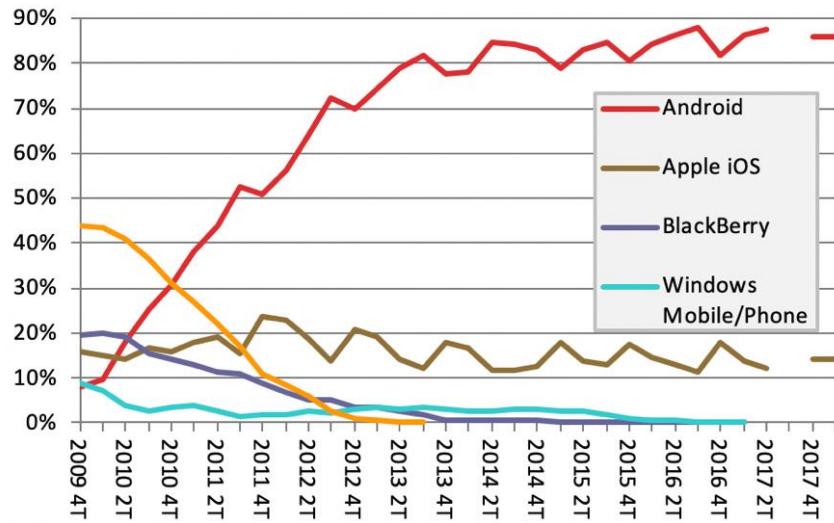


Figura 1: Porcentaje de teléfonos inteligentes vendidos en todo el mundo

hasta el primer trimestre de 2018, según su sistema operativo (fuente: Gartner Group).

video[Tutorial] [Comparativa de las principales plataformas para móviles](#)

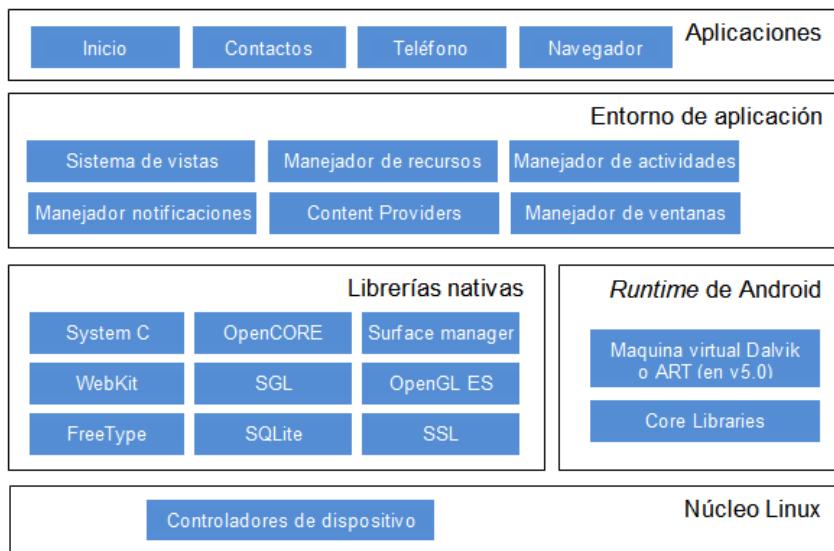
Arquitectura de Android

video [La arquitectura de Android](#)

Objetivos:

Describir la arquitectura de Android y enumerar las características de cada una de sus capas.

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en *software libre*.



Arquitectura de Android.

El núcleo linux

El núcleo de Android está formado por el sistema operativo Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del *hardware*.

Runtime de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dadas las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado), no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión **de crear una nueva, la máquina virtual Dalvik**, que respondiera mejor a estas limitaciones.

Entre las características de la máquina virtual Dalvik que facilitan esta optimización de recursos se encuentra la ejecución de ficheros Dalvik ejecutables (*.dex*) –formato optimizado para ahorrar memoria–. Además, está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

A partir de Android 5.0 se reemplaza Dalvik por ART. Esta nueva máquina virtual consigue reducir el tiempo de ejecución del código Java hasta en un 33%.

También se incluye en el *runtime* de Android el módulo Core Libraries, con la mayoría de las librerías disponibles en el lenguaje Java.

Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- ***System C library:** una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- ***Media Framework:** librería basada en OpenCORE de PacketVideo. Soporta codecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- ***Surface Manager:** maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- ***WebKit/Chromium:** soporta un moderno navegador Web utilizado en el navegador Android y en la vista Webview. En la versión 4.4, WebKit ha sido reemplazada por Chromium/Blink, que es la base del navegador Chrome de Google.
- ***SGL:** motor de gráficos 2D.
- ***Librerías 3D:** implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- ***FreeType:** fuentes en bitmap y renderizado vectorial.
- ***SQLite:** potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- ***SSL:** proporciona servicios de encriptación *Secure Socket Layer* (capa de conexión segura).

Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).

Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.

Los servicios más importantes que incluye son:

- ***Views:** extenso conjunto de vistas, (parte visual de los componentes).
- ***Resource Manager:** proporciona acceso a recursos que no son en código.
- ***Activity Manager:** maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.

- ***Notification Manager:** permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- ***Content Providers:** mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer para su estándar todo lo disponible del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de este

Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de correr en la máquina virtual Dalvik para garantizar la seguridad del sistema.

Normalmente las aplicaciones Android están escritas en Java o Kotlin. Para desarrollar este tipo de aplicaciones podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (*Native Development Kit*)[\[1\]](#).

Descripción de las versiones de Android

Objetivos:

- Enumerar las diferentes versiones que han aparecido de Android y describir sus aportaciones.
-

Antes de empezar a programar en Android hay que elegir la versión del sistema para la que deseamos realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una versión; si las vamos a usar, hemos de conocer la versión mínima necesaria.

Cuando se ha lanzado una nueva plataforma, siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades, y en el caso de modificar alguna funcionalidad, no se elimina, sino que se etiqueta como obsoleta, pero se puede continuar utilizando.

A continuación se describen las plataformas lanzadas hasta la fecha, con una breve descripción de las novedades introducidas. Las plataformas se identifican de tres formas alternativas: versión, nivel de API y nombre comercial. El nivel de API corresponde a números enteros, comenzando desde 1. Para los nombres comerciales se han elegido postres en orden alfabético: Cupcake (v1.5), Donut (v1.6), Éclair (v2.0), Froyo (v2.2), Gingerbread (v2.3), etc. Las dos primeras versiones, que hubieran correspondido a las letras A y B, no recibieron nombre.

video [Descripción de las versiones de Android](#)

Android 1.0 Nivel de API 1 (septiembre 2008)

Primera versión de Android. Nunca se utilizó comercialmente, por lo que no tiene mucho sentido desarrollar para esta plataforma.

Android 1.1 Nivel de API 2 (febrero 2009)

No se añadieron apenas funcionalidades: simplemente se arreglaron algunos errores de la versión anterior. Es la opción a escoger si queremos desarrollar una aplicación compatible con todos los dispositivos Android. No obstante, apenas existen usuarios con esta versión.

Cupcake Android 1.5 Nivel de API 3 (abril 2009)

Es la primera versión con algún usuario, aunque en la actualidad apenas quedan. Como novedades, se incorpora la posibilidad de teclado en pantalla con predicción de texto (ya no es necesario que los terminales tengan un teclado físico), así como la capacidad de grabación avanzada de audio y vídeo. También aparecen los *widgets* de escritorio y *live folders*. Incorpora

soporte para Bluetooth estéreo, por lo que permite conectarse automáticamente a auriculares Bluetooth. Las transiciones entre ventanas se realizan mediante animaciones.

[Donut Android 1.6 Nivel de API 4 \(septiembre 2009\)](#)

Permite capacidades de búsqueda avanzada en todo el dispositivo. También se incorpora gestos y la síntesis de texto a voz. Asimismo, se facilita que una aplicación pueda trabajar con diferentes densidades de pantalla. Soporte para resolución de pantallas WVGA. Aparece un nuevo atributo XML, onClick, que puede especificarse en una vista. Soporte para CDMA/EVDO, 802.1x y VPNs.

[Éclair Android 2.0 Nivel de API 5 \(octubre 2009\)](#)

Esta versión de API apenas cuenta con usuarios, dado que la mayoría de fabricantes pasaron directamente de la versión 1.6 a la 2.1. Como novedades cabría destacar que incorpora un API para manejar el *bluetooth* 2.1. Nueva funcionalidad que permite sincronizar adaptadores para conectarlo a cualquier dispositivo. Ofrece un servicio centralizado de manejo de cuentas. Mejora la gestión de contactos y ofrece más ajustes en la cámara. Se ha optimizado la velocidad de *hardware*. Se aumenta el número de tamaños de ventana y resoluciones soportadas. Nueva interfaz del navegador y soporte para HTML5. Mejoras en el calendario y soporte para Microsoft Exchange. La clase MotionEvent ahora soporta eventos en pantallas multitáctil.

[Android 2.1 Nivel de API 7 \(enero 2010\)](#)

Se considera una actualización menor, por lo que la siguieron llamando Éclair. Destacamos el reconocimiento de voz, que permite introducir un campo de texto dictando sin necesidad de utilizar el teclado. También permite desarrollar fondos de pantalla animados. Se puede obtener información sobre la señal de la red actual que posea el dispositivo. En el paquete WebKit se incluyen nuevos métodos para manipular bases de datos almacenadas en Internet.

[Froyo Android 2.2 Nivel de API 8 \(mayo 2010\)](#)

Como característica más destacada se puede indicar la mejora de velocidad de ejecución de las aplicaciones (ejecución del código de la CPU de 2 a 5 veces más rápido que en la versión 2.1 de acuerdo a varios *benchmarks*). Esto se consigue con la introducción de un nuevo compilador JIT de la máquina Dalvik.

Se añaden varias mejoras relacionadas con el navegador web, como el soporte de Adobe Flash 10.1 y la incorporación del motor Javascript V8 utilizado en Chrome.

El desarrollo de aplicaciones permite las siguientes novedades: se puede preguntar al usuario si desea instalar una aplicación en un medio de almacenamiento externo (como una tarjeta SD), como alternativa a la instalación en la memoria interna del dispositivo; las aplicaciones se actualizan de forma automática cuando aparece una nueva versión; proporciona un servicio para la copia de seguridad de datos que se puede realizar desde la propia aplicación para garantizar al usuario el mantenimiento de sus datos; y por último, se facilita que las aplicaciones interaccionen con el reconocimiento de voz y que terceras partes proporcionen nuevos motores de reconocimiento.

Se mejora la conectividad: ahora podemos utilizar nuestro teléfono para dar acceso a Internet a otros dispositivos (*tethering*), tanto por USB como por Wi-Fi. También se añade el soporte a Wi-Fi IEEE 802.11n y notificaciones *push*.

Se añaden varias mejoras en diferentes componentes: en la API gráfica OpenGL ES; por ejemplo, se pasa a soportar la versión 2.0. Para finalizar, permite definir modos de interfaz de usuario («automóvil» y «noche») para que las aplicaciones se configuren según el modo seleccionado por el usuario.

Gingerbread Android 2.3 Nivel de API 9 (diciembre 2010)

Debido al éxito de Android en las nuevas tabletas ahora soporta mayores tamaños de pantalla y resoluciones (WXGA y superiores).

Incorpora un nuevo interfaz de usuario con un diseño actualizado. Dentro de las mejoras de la interfaz de usuario destacamos la mejora de la funcionalidad de “cortar, copiar y pegar” y un teclado en pantalla con capacidad multitáctil. Se incluye soporte nativo para varias cámaras, pensado en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha propiciado la inclusión de reconocimiento facial para identificar el usuario del terminal.

La máquina virtual de Dalvik introduce un nuevo recolector de basura que minimiza las pausas de la aplicación, ayudando a garantizar una mejor animación y el aumento de la capacidad de respuesta en juegos y aplicaciones similares. Se trata de corregir así una de las lacras de este sistema operativo móvil, que en versiones previas no ha sido capaz de cerrar bien las aplicaciones en desuso. Se dispone de mayor apoyo para el desarrollo de código nativo (NDK). También se mejora la gestión de energía y control de aplicaciones. Y se cambia el sistema de ficheros, que pasa de YAFFS a ext4.

Entre otras novedades destacamos en soporte nativo para telefonía sobre Internet VoIP/SIP. El soporte para reproducción de vídeo WebM/VP8 y codificación de audio AAC. El soporte para la tecnología NFC. Las facilidades en el audio, gráficos y entradas para los desarrolladores de juegos. El soporte nativo para más sensores (como giroscopios y barómetros). Un gestor de descargas para las descargas largas.

Honeycomb Android 3.0 Nivel de API 11 (febrero 2011)

Para mejorar la experiencia de Android en las nuevas tabletas se lanza la versión 3.0 optimizada para dispositivos con pantallas grandes. La nueva interfaz de usuario ha sido completamente rediseñada con paradigmas nuevos para la interacción y navegación. Entre las novedades introducidas destacan: Los *fragments*, con los que podemos diseñar diferentes elementos del interfaz de usuario. La barra de acciones, donde las aplicaciones pueden mostrar un menú siempre visible. Las teclas físicas son reemplazadas por teclas en pantalla; se mejoran las notificaciones, arrastrar y soltar y las operaciones de cortar y pegar.

La nueva interfaz se pone a disposición de todas las aplicaciones, incluso las construidas para versiones anteriores de la plataforma. Esto se consigue gracias a la introducción de librerías de compatibilidad[\[1\]](#) que pueden ser utilizadas en versiones anteriores a la 3.0.

Se mejoran los gráficos 2D/3D gracias al renderizador OpenGL acelerado por hardware. Apacere el nuevo motor de gráficos Rederscript, que saca mayor rendimiento al hardware e incorpora su

propia API. Se incorpora un nuevo motor de animaciones mucho más flexible, conocido como animación de propiedades.

Primera versión de la plataforma que soporta procesadores multinúcleo. La máquina virtual Dalvik ha sido optimizada para permitir multiprocesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.

Se incorporan varias mejoras multimedia, como listas de reproducción M3U a través de HTTP Live Streaming, soporte a la protección de derechos musicales (DRM) y soporte para la transferencia de archivos multimedia a través de USB con los protocolos MTP y PTP.

En esta versión se añaden nuevas alternativas de conectividad, como las nuevas APIs de Bluetooth A2DP y HSP con streaming de audio. También, se permite conectar teclados completos por USB o Bluetooth.

Se mejora el uso de los dispositivos en un entorno empresarial. Entre las novedades introducidas destacamos las nuevas políticas administrativas con encriptación del almacenamiento, caducidad de contraseña y mejoras para administrar los dispositivos de empresa de forma eficaz.

A pesar de la nueva interfaz gráfica optimizada para tabletas, Android 3.0 es compatible con las aplicaciones creadas para versiones anteriores.

[Android 3.1 Nivel de API 12 \(mayo 2011\)](#)

Se permite manejar dispositivos conectados por USB (tanto host como dispositivo). Protocolo de transferencia de fotos y vídeo (PTP/MTP) y de tiempo real (RTP).

[Android 3.2 Nivel de API 13 \(julio 2011\)](#)

Optimizaciones para distintos tipos de tableta. Zoom compatible para aplicaciones de tamaño fijo. Sincronización multimedia desde SD.

[Ice Cream Sandwich Android 4.0 Nivel de API 14 \(octubre 2011\)](#)

La característica más importante es que se unifican las dos versiones anteriores (2.x para teléfonos y 3.x para tabletas) en una sola compatible con cualquier tipo de dispositivo. A continuación destacamos algunas de las características más interesantes.

Se introduce una nueva interfaz de usuario totalmente renovada; por ejemplo, se reemplazan los botones físicos por botones en pantalla (como ocurría en las versiones 3.x). Nueva API de reconocimiento facial que, entre otras muchas aplicaciones, permite al propietario desbloquear el teléfono. También se mejora en el reconocimiento de voz; por ejemplo, se puede empezar a hablar sin esperar la conexión con el servidor.

Aparece un nuevo gestor de tráfico de datos por Internet, donde podremos ver el consumo de forma gráfica y donde podemos definir los límites de ese consumo para evitar cargos inesperados con la operadora. Incorpora herramientas para la edición de imágenes en tiempo real, para distorsionar, manipular e interactuar con la imagen en el momento de ser capturada. Se mejora la API para comunicaciones por NFC y la integración con redes sociales.

En diciembre de 2011 aparece una actualización de mantenimiento (versión 4.0.2) que no aumenta el nivel de API.

Android 4.0.3 Nivel de API 15 (diciembre 2011)

Se introducen ligeras mejoras en algunas APIs incluyendo el de redes sociales, calendario, revisor ortográfico, texto a voz y bases de datos entre otros. En marzo de 2012 aparece la actualización 4.0.4.

Jelly Bean Android 4.1 Nivel de API 16 (julio 2012)

En esta versión se hace hincapié en mejorar un punto débil de Android: la fluidez de la interfaz de usuario. Con este propósito se incorporan varias técnicas: sincronismo vertical, triple búfer y aumento de la velocidad del procesador al tocar la pantalla.

Se mejoran las notificaciones con un sistema de información expandible personalizada. Los *widgets* de escritorio pueden ajustar su tamaño y hacerse sitio de forma automática al situarlos en el escritorio. El dictado por voz puede realizarse sin conexión a Internet (de momento, solo en inglés).

Se introducen varias mejoras en Google Search. Se potencia la búsqueda por voz con resultados en forma de ficha. La función Google Now permite utilizar información de posición, agenda y hora en las búsquedas.

Se incorporan nuevos soportes para usuarios internacionales, como texto bidireccional y teclados instalables. Para mejorar la seguridad, las aplicaciones son cifradas. También se permiten actualizaciones parciales de aplicaciones.

Android 4.2 Nivel de API 17 (noviembre 2012)

Una de las novedades más importantes es que podemos crear varias cuentas de usuario en el mismo dispositivo. Aunque esta característica solo está disponible en tabletas. Cada cuenta tendrá sus propias aplicaciones y su propia configuración.

Los *Widgets* de escritorio pueden aparecer en la pantalla de bloqueo. Se incorpora un nuevo teclado predictivo deslizante al estilo Swype. Posibilidad de conectar dispositivo y TVHD mediante wifi (Miracast). Mejoras menores en las notificaciones. Nueva aplicación de cámara que incorpora la funcionalidad Photo Sphere para hacer fotos panorámicas inmersivas (en 360º).

Android 4.3 Nivel de API 18 (julio 2013)

Esta versión introduce mejoras en múltiples áreas. Entre ellas los *perfiles restringidos* (disponible sólo en tabletas) que permiten controlar los derechos de los usuarios para ejecutar aplicaciones específicas y para tener acceso a datos específicos. Igualmente, los programadores pueden definir restricciones en las apps, que los propietarios puedan activar si quieren. Se da soporte para Bluetooth Low Energy (BLE) que permite a los dispositivos Android comunicarse con los periféricos con bajo consumo de energía. Se agregan nuevas características para la codificación, transmisión y multiplexación de datos multimedia. Se da soporte para OpenGL ES 3.0. Se mejora la seguridad para gestionar y ocultar las claves privadas y credenciales.

KitKat Android 4.4 Nivel de API 19 (octubre 2013)

Aunque se esperaba la versión número 5.0 y con el nombre *Key Lime Pie*, Google sorprendió con el cambio de nombre, que se debió a un acuerdo con Nestlé para asociar ambas marcas.

El principal objetivo de la versión 4.4 es hacer que Android esté disponible en una gama aún más amplia de dispositivos, incluyendo aquellos con tamaños de memoria RAM de solo 512 MB. Para ello, todos los componentes principales de Android han sido recortados para reducir sus requerimientos de memoria, y se ha creado una nueva API que permite adaptar el comportamiento de la aplicación en dispositivos con poca memoria.

Más visibles son algunas nuevas características de la interfaz de usuario. El modo de inmersión en pantalla completa oculta todas las interfaces del sistema (barras de navegación y de estado) de tal manera que una aplicación puede aprovechar el tamaño de la pantalla completa. WebViews (componentes de la interfaz de usuario para mostrar las páginas Web) se basa ahora en el software de Chrome de Google y por lo tanto puede mostrar contenido basado en HTML5.

Se mejora la conectividad con soporte de NFC para emular tarjetas de pago tipo HCE, varios protocolos sobre Bluetooth y soporte para mandos infrarrojos. También se mejoran los sensores para disminuir su consumo y se incorpora un sensor contador de pasos. Se facilita el acceso de las aplicaciones a la nube con un nuevo marco de almacenamiento. Este marco incorpora un tipo específico de content provider conocido como document provider, nuevas intenciones para abrir y crear documentos y una ventana de diálogo que permite al usuario seleccionar ficheros. Se incorpora un administrador de impresión para enviar documentos a través de Wi-Fi a una impresora. También se añade un content provider para gestionar los SMS.

Desde una perspectiva técnica, hay que destacar la introducción de la nueva máquina virtual ART, que consigue tiempos de ejecución muy superiores a la máquina Dalvik. Sin embargo, todavía está en una etapa experimental. Por defecto se utiliza la máquina virtual Dalvik, y se permite a los programadores activar opcionalmente ART para verificar que sus aplicaciones funcionan correctamente.

video [Android 4.4 KitKat](#)

Lollipop Android 5.0 Nivel de API 21 (noviembre 2014)

La novedad más importante de Lollipop es la extensión de Android a nuevas plataformas, incluyendo Android Wear, Android TV y Android Auto. Hay un cambio significativo en la arquitectura, al utilizar la máquina virtual ART en lugar de Dalvik. Esta novedad ya había sido incorporada en la versión anterior a modo de prueba. ART mejora de forma considerable el tiempo de ejecución del código escrito en Java. Además se soporta dispositivos de 64 bits en procesadores ARM, x86, y MIPS. Muchas aplicaciones del sistema (Chrome, Gmail,...) se han incorporado en código nativo para una ejecución más rápida.

Desde el punto de vista del consumo de batería, hay que resaltar que en Lollipop el modo de ahorro de batería se activa por defecto. Este modo desconecta algunos componentes en caso de que la batería esté baja. Se incorpora una nueva API (`android.app.job.JobScheduler`) que nos permite que ciertos trabajos se realicen solo cuando se cumplen determinadas condiciones (por ejemplo con el dispositivo cargando). También se incluyen completas estadísticas para analizar el consumo que nuestras aplicaciones hacen de la batería.

En el campo Gráfico Android Lollipop incorpora soporte nativo para OpenGL ES 3.1. Además esta versión permite añadir a nuestras aplicaciones un paquete de extensión con funcionalidades gráficas avanzadas (fragment shader, tessellation, geometry shaders, ASTC,...).

Otro aspecto innovador de la nueva versión lo encontramos en el diseño de la interfaz de usuario. Se han cambiado los iconos, incluyendo los de la parte inferior (Retroceder, Inicio y Aplicaciones), que ahora son un triángulo, un círculo y un cuadrado.



El nuevo enfoque se centra en Material Design (<http://www.google.com/design/material-design.pdf>). Consiste en una guía completa para el diseño visual, el movimiento y las interacciones a través de plataformas y dispositivos. Google pretende aplicar esta iniciativa a todas las plataformas, incluyendo wearables y Google TV. La nueva versión también incluye varias mejoras para controlar las notificaciones. Ahora son más parecidas a las tarjetas de Google Now y pueden verse en la pantalla de bloqueo.

Se incorporan nuevos sensores como el de pulso cardíaco, el de inclinación (para reconocer el tipo de actividad del usuario), y sensores de interacción compuestos para detectar ciertos gestos.

Como curiosidad la nueva versión introduce un modo de bloqueo que impide al usuario salir de una aplicación y bloquea las notificaciones. Esto podría utilizarse, por ejemplo, para que mientras un usuario realiza un examen, no pueda ver las notificaciones, acceder a otras aplicaciones, o volver a la pantalla de inicio.

Video [Android 5.0 Lollipop](#)

Android 5.1 Nivel de API 22 (marzo 2015)

Se añaden algunas mejoras a nivel de usuario en los ajustes rápidos. A nivel de API se añade soporte para varias tarjetas SIM en un mismo teléfono; la clase `AndroidHttpClient` se marca como obsoleta; y se añade un API para que las empresas proveedoras de servicios de telecomunicación puedan distribuir software de forma segura a través de Google Play. La característica más interesante es que para poder acceder a esta API la aplicación ha de estar firmada con un certificado que coincide con el que el usuario tiene en su tarjeta UICC.

Mashmallow Android 6.0 Nivel de API 23 (octubre 2015)

Una de las novedades más interesantes es el administrador de permisos. Los usuarios podrán conceder o retirar ciertos permisos a cada aplicación. Con esto el sistema da mucha más protección a la privacidad de los usuarios.

Ahora, el sistema realiza una copia de seguridad automática de todos los datos de las aplicaciones. Esto resulta muy útil al cambiar de dispositivo o tras restaurar valores de fábrica. Para disponer de esta funcionalidad simplemente usa el target Android 6.0. No es necesario agregar código adicional.

Android 6.0 integra el asistente por voz Now on Tap. Es una evolución de Google Now más integrada con las aplicaciones. Se activa con pulsación larga de home. Aparecerán tarjetas sobre la aplicación actual y lo que muestra. La aplicación actual podrá aportar información al asistente.

En esta misma línea, se añade un API que permite interacciones basadas en voz. Es decir, si nuestra aplicación ha sido lanzada por voz, podremos solicitar una confirmación de voz del usuario, seleccionar de una lista de opciones o cualquier información que necesite.

Se introducen los enlaces de aplicación con los que podremos asociar la aplicación que abre una URL en función de su dominio web. Aunque muchos dispositivos ya lo permitían, en esta actualización se añade autenticación por huella digital a la API. Tu aplicación puede autenticar al usuario usando las credenciales para desbloquear su dispositivo (pin, patrón o contraseña). Esto libera al usuario de tener que recordar contraseñas específicas de la aplicación. Y te evita tener que implementar tu propia interfaz de autenticación.

Compartir con otros usuarios ahora es más fácil con Direct Share. Permite no solo escoger la aplicación con la que compartes, sino también el usuario. Si tu aplicación es un posible destino para compartir vas a poder indicar al sistema la lista de usuarios que pueden recibir información.

En Android 6.0 podemos utilizar parte de un dispositivo de almacenamiento externo, para que sea usado como almacenamiento interno. Podemos fragmentar, formatear y encriptar una tarjeta SD para ser usada como memoria interna. También podemos montar y extraer lápices de memoria USB de forma nativa

Se incorpora la plataforma de pagos abierta *Android Pay* que combina NFC y Host Card Emulation. El nuevo gestor de batería, Doze, realiza un uso más eficiente de los recursos, con lo que podemos obtener dos horas extras de autonomía. Se da soporte de forma nativa a pantallas 4 K, lápices Bluetooth, múltiples tarjetas SIM y linterna. Mejoras de posicionamiento utilizando redes WiFi y dispositivos Bluetooth

video [Android 6.0 Marshmallow](#)

Android Nougat Android 7.0 Nivel de API 24 (julio 2016)

Ahora los usuarios pueden abrir varias aplicaciones al mismo tiempo en la pantalla. Puedes configurar tu aplicación para que se visualice con unas dimensiones mínimas o inhabilitar la visualización de ventanas múltiples.

Las notificaciones han sido rediseñadas para un uso más ágil. Hay más opciones para personalizar el estilo de los mensajes (*MessageStyle*). Puedes agrupar notificaciones por temas o programar una respuesta directa.

En la versión anterior se utilizaba una estrategia de compilación *Ahead of Time* (AOT): cuando se descargaba una aplicación, su código era traducido de *bytecodes* a código nativo, lo que mejoraba los tiempos de ejecución. En la nueva versión se incorpora también la compilación *Just in Time* (JIT), donde no se compila hasta que el código va a ser ejecutado. Android 7.0 propone un planteamiento mixto según el perfil del código. Los métodos directos se compilan previamente (AOT), mientras que otras partes no se compilan hasta que se usan (JIT). Aunque AOT puede introducir retardos en ejecución, ahorra tiempo en la precompilación y en memoria. El mayor impacto de esta técnica se nota en la instalación de las aplicaciones y actualizaciones del sistema. Mientras que en Android 6.0 una instalación podría usar varios minutos, ahora se instala en cuestión de segundos.

Android Nougat incorpora la plataforma de realidad virtual *Daydream*. Se trata de una propuesta de Google que complementa la iniciativa *Cardboard*. Incluye

especificaciones *software* y *hardware* que nos permitirán diferenciar a los dispositivos compatibles. Los principales fabricantes de móviles se han unido a esta iniciativa.

En la versión anterior, el gestor de batería Doze solo se activaba cuando el dispositivo estaba en reposo. Ahora, se activa poco tiempo después de apagarse la pantalla. Esto permite ahorrar batería cuando llevamos el dispositivo en el bolsillo.

También se ha añadido la nueva API para gráficos 3D, Vulcan, como alternativa a OpenGL. Minimiza la sobrecarga de CPU en el controlador, lo que permite aumentar la velocidad de los juegos.

El usuario va a poder activar el modo de ahorro de datos cuando se encuentre en itinerancia o cuando esté a punto de agotar un paquete de datos. En este caso, tanto el sistema como las aplicaciones han de tratar de minimizar al máximo las transferencias de datos.

Android 7.1 Nivel de API 25 (diciembre 2016)

La principal novedad son los accesos directos a aplicaciones. Desde el icono de la aplicación, con una pulsación prolongada, aparecen varias opciones que podremos seleccionar. Por ejemplo, podremos iniciar una navegación privada con Chrome de forma directa. Los accesos directos que quieras incorporar a tu aplicación, podrás configurarlo por medio de *intens*, que han de especificarse en un fichero de configuración[\[2\]](#).

Se incorporan otras novedades como la posibilidad de insertar imágenes desde el teclado, de la misma forma que ahora insertamos emoticonos.

video [Android 7.0 Nougat](#)

Android Oreo Android 8.0 Nivel de API 26 (agosto 2017)

Destacan las siguientes mejoras en seguridad: Se introduce Google Play Protect, que escanea regularmente las aplicaciones en busca de malware. La opción "Orígenes desconocidos" desaparece. Ahora podemos indicar que aplicaciones pueden instalar apks y cuales no. Desde la opción "Acceso especial de aplicaciones" podemos configurar que aplicaciones pueden realizar ciertas acciones.

El sistema limita más los procesos en segundo plano, para conseguir ahorro en la batería. Se mejora el tiempo de arranque del sistema

Pensando en los países emergentes, se lanza Android Go, una distribución adaptada para dispositivos de gama baja (1 GB de RAM o menos). Se preinstalan apps ligeras y en Google Play Store destaca aplicaciones ligeras adecuadas para estos dispositivos. Estas aplicaciones han de cubrir 3 requisitos, trabajar sin red, pesar menos de 10 MB y un buen rendimiento de batería. Con el fin de reducir la fragmentación de Android, aparece el proyecto Treble que facilitará las actualizaciones a los fabricantes. Se reestructura la arquitectura de Android para definir una interfaz clara entre capa del Núcleo Linux (con sus drivers) de las capas del Framework. Esto permite actualizar Android sin tener que tocar la capa del Núcleo Linux.

Las notificaciones presentan varias mejoras: Podemos añadir color de fondo. Se ordenan por importancia. Las aplicaciones pueden crear canales de notificaciones y el usuario decidir cuales quiere recibir. Podemos posponer una notificación o verlas pulsando sobre el icono de la aplicación.

Los iconos tendrán que estar diseñados en dos capas: el icono y el fondo del icono. Esto permite adaptarse al dispositivo. Además el usuario podrá escoger entre iconos circulares, cuadrados o de esquinas redondeadas.

Ahora podemos reproducir un vídeo en una ventana flotante mientras utilizamos otras aplicaciones. Al seleccionar un texto se nos sugieren acciones cuando se trata de un número de teléfono o una dirección. El Autocompletar de Google, que antes estaba disponible en Chrome para guardar contraseñas, ahora se puede usar en cualquier aplicación Android.

[Android Pie Android 9.0 Nivel de API 28 \(agosto 2018\)](#)

Una de las novedades más interesantes es el nuevo API WiFi RTT introducido en IEEE 802.11mc. Permite estimar la distancia entre nuestro dispositivo y los puntos de acceso cercanos, lo que permite sistemas de posicionamiento en interiores con una precisión de 1 a 2 metros. Otro importante cambio es la navegación por gestos. Se reemplazan los tres botones en pantalla (triángulo, círculo y cuadrado) por solo 2  (retroceder e inicio). El botón de inicio admite diferentes gestos que nos permite ir al asistente de Google, cambiar entre apps recientes o abrir el menú de apps.

Una interesante innovación es el uso de Inteligencia Artificial, para mejorar diferentes aspectos. La idea consiste en aprender nuestros hábitos a la hora de usar las aplicaciones. Con esta información se puede quitar preferencia sobre el uso de la CPU a las apps menos utilizadas, consiguiendo una reducción de hasta un 30 %. Este menor uso de la CPU prolongará la vida de la batería. Usando técnicas similares se pretende aprender cuando el usuario va a arrancar una aplicación o una acción de esta. De esta forma el sistema puede cargar en memoria la aplicación antes incluso que el usuario decida utilizarla.

Se introducen algunas mejoras que fomentan un uso responsable y saludable del móvil. Por ejemplo, desde el Dashboard podemos consultar el uso que hacemos cada día, en cada aplicación. Podemos establecer alarmas de uso excesivo muy interesantes para el control parental. En esta línea, se introducen nuevos modos de relajación y no molestar para favorecer la desconexión digital.

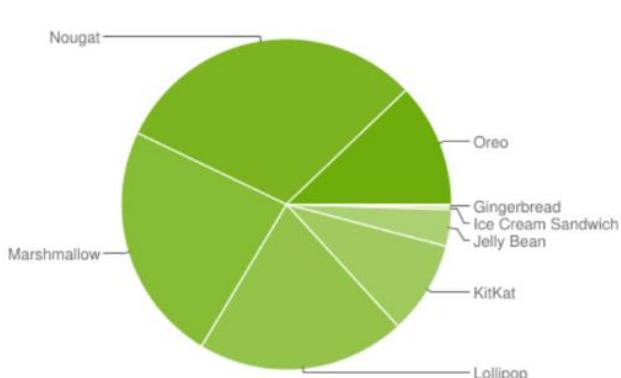
Elegir la versión en una aplicación Android

Vídeo [Elegir la versión en una aplicación Android](#)

Objetivos:

Mostrar las claves para seleccionar la versión de desarrollo en una aplicación Android.

A la hora de seleccionar la plataforma de desarrollo hay que consultar si necesitamos alguna característica especial que solo esté disponible a partir de una versión. Todos los usuarios con versiones inferiores a la seleccionada no podrán instalar la aplicación. Por lo tanto, es recomendable seleccionar la menor versión posible que nuestra aplicación pueda soportar. Por ejemplo, si en nuestra aplicación queremos utilizar el motor de animaciones de propiedades, tendremos que utilizar la versión 3.0, al ser la primera que lo soporta. El problema es que la aplicación no podrá ser instalada en dispositivos que tengan una versión anterior a la 3.0. Para ayudarnos a tomar la decisión de qué plataforma utilizar, puede ser interesante consultar los porcentajes de utilización:



Plataforma	Nivel API	Porcent.
2.3 Gingerbread	10	0,2 %
4.0 Ice Cream S.	15	0,3 %
4.1 Jelly Bean	16	1,2 %
4.2	17	1,9 %
4.3	18	0,5 %
4.4 KitKat	19	9,1 %
5.0 Lollipop	21	4,2 %
5.1	22	16,2 %
6.0 Marshmallow	23	23,5 %
7.0 Nougat	24	21,2 %
7.1	25	9,6 %
8.0 Oreo	26	10,1 %
8.1	27	2,0 %

Dispositivos Android, según la plataforma instalada, que han accedido a Google Play Store el 27 de julio de 2018 y los 7 días anteriores.

Las versiones con porcentajes inferiores al 0,1 % no se muestran.

Tras estudiar la gráfica podemos destacar el reducido número de usuarios que utilizan la versión 2.3 (0,2%). Por lo tanto, puede ser buena idea utilizar como versión mínima la 4.1, 4.2 o 4.4 para desarrollar nuestro proyecto, dado que daríamos cobertura al 99%, 98% o 96% de los terminales. Las versiones 3.x han tenido muy poca difusión, por lo que no aparecen en la tabla. Las versiones 6.0 y 7.x son mayoritarias. La versión 8.x todavía no dispone de un número importante de usuarios. No obstante, estas cifras cambian mes a mes, por lo que recomendamos consultar los siguientes enlaces antes de tomar decisiones sobre las versiones a utilizar.

Enlaces de interés:

- **Android developers: Platform Versions:** Estadística de dispositivos Android según plataforma instalada, que han accedido a Android

- Market. <http://developer.android.com/about/dashboards/index.html>
- **Android developers: about:** En el menú de la izquierda aparecen links a las principales versiones de la plataforma. Si pulsas sobre ellos encontrarás una descripción exhaustiva de cada plataforma. <http://http://developer.android.com/about/index.html>

Las librerías de compatibilidad (support library)

Objetivos:

- Describir el sistema de librerías de compatibilidad (support library) de Android
 - Compararlas con el sistema tradicional de niveles de API
 - Enumerar las librerías aparecidas hasta la fecha
 - Conocer como desde la versión 9.0 se recomienda utilizar estas librerías desde los paquetes androidx.*
-

Tal y como se ha descrito, la filosofía tradicional de Android ha sido que las novedades que aparecen en una API solo puedan usarse en dispositivos que soporten esa API. Como acabamos de ver, la fragmentación de las versiones de Android es muy grande, es decir, actualmente podemos encontrar dispositivos con una gran variedad de versiones. Con el fin de que la aplicación pueda ser usada por el mayor número posible de usuarios hemos de ser muy conservadores a la hora de escoger la versión mínima de API de nuestra aplicación. La consecuencia es que las novedades que aparecen en las últimas versiones de Android no pueden ser usadas.

En la versión 3.0 aparecieron importantes novedades que Google quería que se incorporaran en las aplicaciones lo antes posible (*fragments*, nuevas notificaciones, etc.). Con este fin creó las librerías de compatibilidad para poder incorporar ciertas funcionalidades en cualquier versión de Android.

video [Las librerías de compatibilidad \(support library\)](#)

Desde la versión 9.0 las librerías de compatibilidad también se incluyen en la librerías AndroidX[1], que son parte del proyecto Jetpack[2]. En las librerías AndroidX se incluye tanto las librerías de compatibilidad como los componentes de Jetpack.

A diferencia de la librería de compatibilidad, cada paquete de AndroidX tiene su propia versión, y se mantienen y actualizan de manera separada. Todos los paquetes están en un espacio de nombre que empieza por androidx.*.

- * **v4 Support Library: (androidx.legacy:legacy-support-v4)** Esta librería permitía utilizar muchas clases introducidas en la versión 3.0 cuando trabajamos con un API mínimo anterior. En la actualidad ya no es necesaria utilizarla, dado que ya es recomendable utilizar como API mínimo la versión 4.0 o incluso superior. Puede usarse en una aplicación con nivel de API 4 (v1.6) o superior. Incorpora las clases: Fragment, NotificationCompat, LocalBroadcastManager, ViewPager, PagerTitleStrip, PagerTabStrip, DrawerLayout, SlidingPaneLayout, ExploreByTouchHelper, Loader y FileProvider.

- * **appcompat: (androidx.appcompat)** Permite utilizar un IU basado en la Barra de Acciones siguiendo especificaciones de Material Design. Se añade por defecto cuando creamos un nuevo proyecto. Incorpora las clases: ActionBar AppCompatActivity, AppCompatActivityDialog y ShareActionProvider.
- * **recyclerview: (androidx.recyclerview)** Esta Incorpora la vista RecyclerView, una versión mejorada que reemplaza a ListView y GridView.
- * **constraintlayout: (androidx.constraintlayout)** Da soporte al layout ConstraintLayout.
- * **prefrence: (androidx.preference)** Incorpora la
- clases CheckBoxPreference y ListPreference usadas en preferencias.
- * **cardview: (androidx.cardview)** Incorpora la vista CardView, una forma estándar de mostrar información especialmente útil en Android Wear y TV.
- * **palette: (androidx.palette)** Esta Incorpora la vista Palette, que permite extraer los colores principales de una imagen.
- * **mediarouter: (androidx.mediarouter)** Da soporte a Google Cast.
- * **material Desing Support Library: (com.google.android.material)** libreria que incorpora Material Design.

Si tienes dudas sobre los nuevos paquetes utilizados consulta la siguiente tabla: <https://developer.android.com/jetpack/androidx/migrate>

Creación de dispositivos virtuales (AVD)

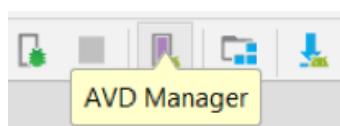
Objetivos:

- Se describe el uso de la herramienta Android Virtual Device Manager para la creación de dispositivos virtuales Android (AVD).

Un dispositivo virtual Android (AVD) te va a permitir emular en tu ordenador diferentes tipos de dispositivos basados en Android. De esta forma podrás probar tus aplicaciones en una gran variedad de teléfonos, tabletas, relojes o TV con cualquier versión de Android, tamaño de pantalla o tipo de entrada.

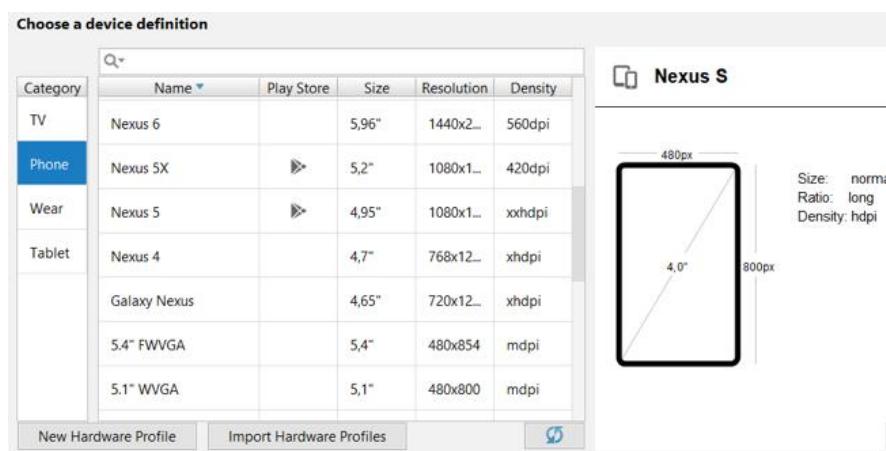
Ejercicio: Creación de un dispositivo virtual Android (AVD)

1. Pulsa el botón AVD Manager:



Aparecerá la lista con los AVD creados. La primera vez estará vacía.

2. Pulsa a continuación el botón *Create Virtual Device...* para crear un nuevo AVD. Aparecerá la siguiente ventana:

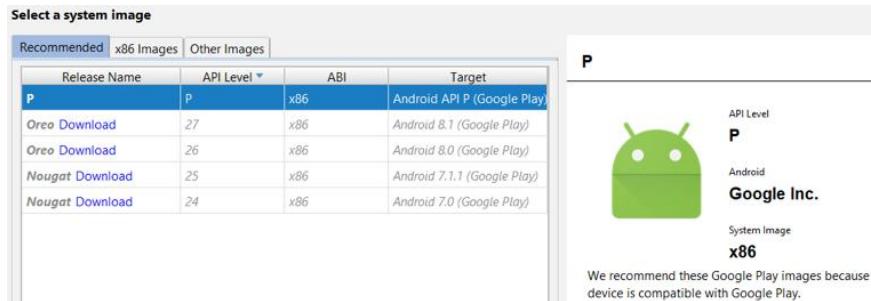


3. En la primera columna podremos seleccionar el tipo de dispositivo a emular (móvil, tableta, dispositivo *wearable* o Google TV). A la derecha, se muestran distintos dispositivos que emulan dispositivos reales de la familia Nexus y también otros genéricos. Junto al nombre de cada dispositivo, se indica si tiene la posibilidad de incorporar Google Play, el tamaño de la pantalla en pulgadas, la resolución y el tipo de densidad gráfica.

NOTA: Los tipos de pantalla se clasifican en Android según su densidad gráfica: *ldpi*, *mdpi*, *hdpi*, *xhdpi*, ... Véase sección 2.6 Recursos alternativos

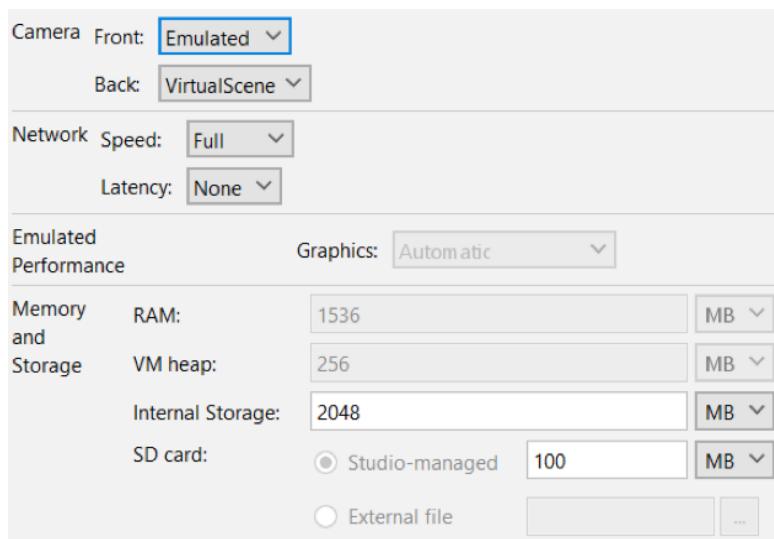
Si quisieras añadir a esta lista crear un nuevo tipo de dispositivo, puedes seleccionar *New Hardware Profile*. Podrás indicar las principales características del dispositivo y ponerle un nombre. Usando *Clone Device* podrás crear un nuevo tipo de AVD a partir del actual. Pulsando con el botón derecho sobre un tipo de dispositivo podrás eliminarlos o exportarlos a un fichero.

- Pulsa *Next* para pasar a la siguiente ventana, donde podrás seleccionar la imagen del sistema que tendrá el dispositivo y el tipo de procesador:



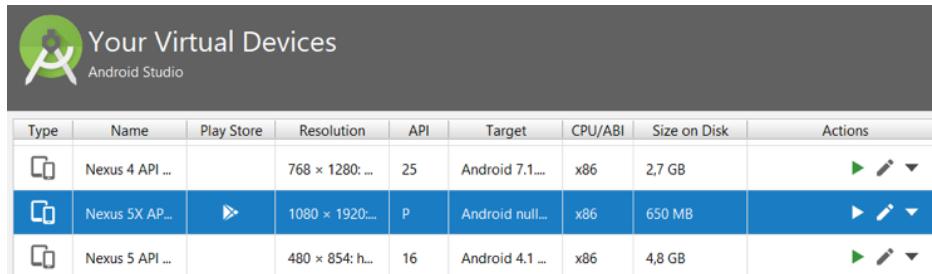
Observa como las distintas versiones de Android se pueden seleccionar, solo con el código abierto de Android, añadiendo las API de Google (para utilizar servicios como Google Maps) o incluso incorporando Google Play (Para poder instalar apps desde la tienda de Google).

- Pulsa *Next* para pasar a la última ventana. Se nos mostrará un resumen con las opciones seleccionadas, además podremos seleccionar la orientación inicial del AVD, si queremos usar el coprocesador gráfico (GPU) de nuestro ordenador o si queremos que dibuje un marco alrededor del emulador simulando un dispositivo real.
- Pulsa en el botón *Show Advanced Settings* para que se muestren algunas configuraciones adicionales:



Podemos hacer que el emulador utilice la cámara o teclado de nuestro ordenador. También podemos limitar la velocidad y latencia en el acceso a la red. Finalmente, podremos ajustar la memoria utilizada: RAM total del dispositivo, memoria dinámica usada por Java y memoria para almacenamiento, tanto interna como externa.

- Una vez introducida la configuración deseada, pulsa el botón *Finish*. Aparecerá el dispositivo creado en la lista:



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
□	Nexus 4 API ...		768 x 1280: ...	25	Android 7.1....	x86	2,7 GB	▶ ⚪ ▾
□	Nexus 5X AP...	▶	1080 x 1920:...	26	Android null...	x86	650 MB	▶ ⚪ ▾
□	Nexus 5 API ...		480 x 854: h...	16	Android 4.1 ...	x86	4,8 GB	▶ ⚪ ▾

8. Para arrancarlo, pulsa el botón con forma de triángulo verde que encontrarás en la columna de la derecha. Es posible que te pregunte por la entrada de vídeo para emular la cámara del AVD.



NOTA: Algunas características de hardware no están disponibles en el emulador; por ejemplo, el multi-touch o los sensores.

video [Creación de dispositivos virtuales Android \(AVD\)](#)

Recursos adicionales: Teclas de acceso rápido en un emulador

Inicio: Tecla Home.

F2: Tecla Menú.

Esc: Tecla de volver.

F7: Tecla On/Off

Ctrl-F5/Ctrl-F6 ó KeyPad +/-: Control de volumen de audio.

Ctrl-F11 ó KeyPad 7: Cambia la orientación entre horizontal y vertical.

Un primer programa en Android

Objetivos:

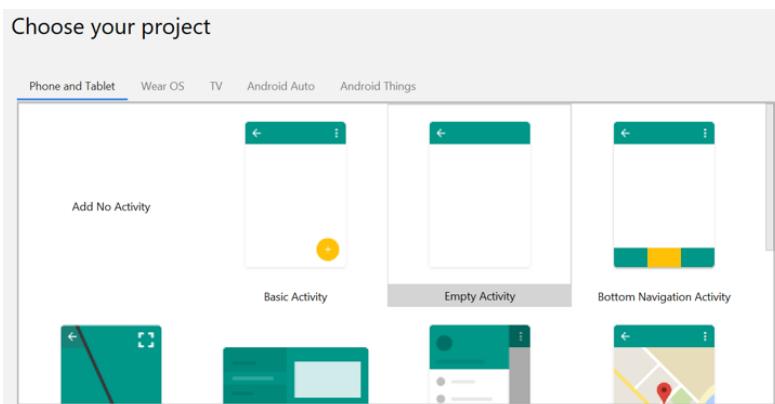
1. Describir los diferentes ficheros creados automáticamente en un nuevo proyecto Android
 2. Diferenciación entre código, contenido y diseño.
-

Utilizar un entorno de desarrollo nos facilita mucho la creación de programas. Esto es especialmente importante en Android dado que tendremos que utilizar una gran variedad de ficheros. Gracias a Android Studio, la creación y gestión de proyectos se realizará de forma muy rápida, acelerando los ciclos de desarrollo.

Ejercicio paso a paso: *Crear un primer proyecto*

Para crear un primer proyecto Android, con Android Studio sigue los siguientes pasos:

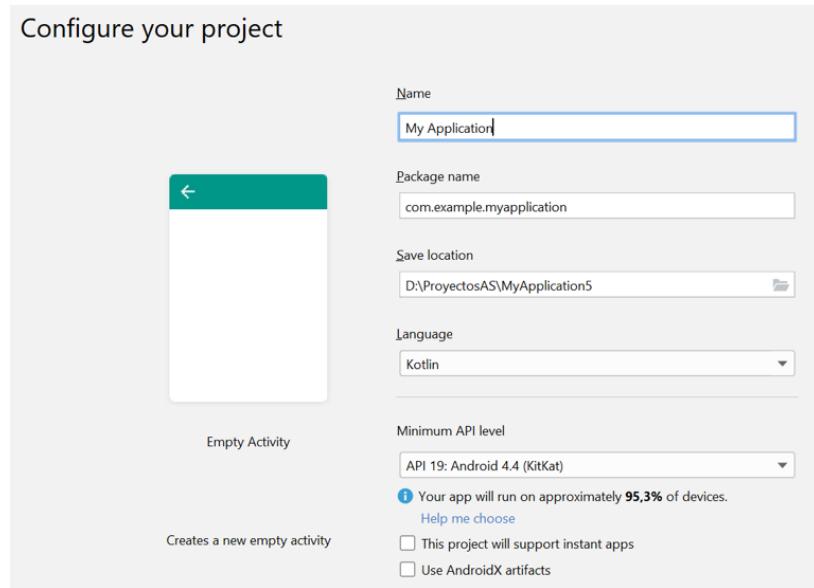
1. Selecciona **File > New Project...**
2. En primer lugar podrás indicar la plataforma para la que quieras desarrollar (Teléfonos y tabletas, Wear OS, TV, ...) y el tipo de actividad inicial quieres en tu aplicación:



En este curso nos centraremos en las aplicaciones para teléfonos y tabletas, por lo que has de seleccionar siempre la primera pestaña.

Observa cómo para este tipo de aplicación puedes elegir diferentes clases de actividades que incorporen ciertos elementos de uso habitual, como menús, botones, anuncios, etc. El concepto de actividad será explicado más adelante. Selecciona *Empty Activity* para añadir una actividad inicial.

3. Pulsa Next para pasar a la pantalla donde se rellena los detalles del proyecto. Puedes dejar los valores por defecto:



A continuación vemos una descripción para cada campo:

Name: Es el nombre de la aplicación que aparecerá en el dispositivo Android. Tanto en la barra superior, cuando esté en ejecución, como en el ícono que se instalará en el menú de programas.

Package name: Indicamos el nombre de paquete de la aplicación. **Las clases Java que creamos pertenecerán a este paquete.** Como veremos a lo largo del curso, el nombre del paquete también es utilizado por Android para múltiples funciones. Por ejemplo, para determinar en qué directorio se instala la aplicación.

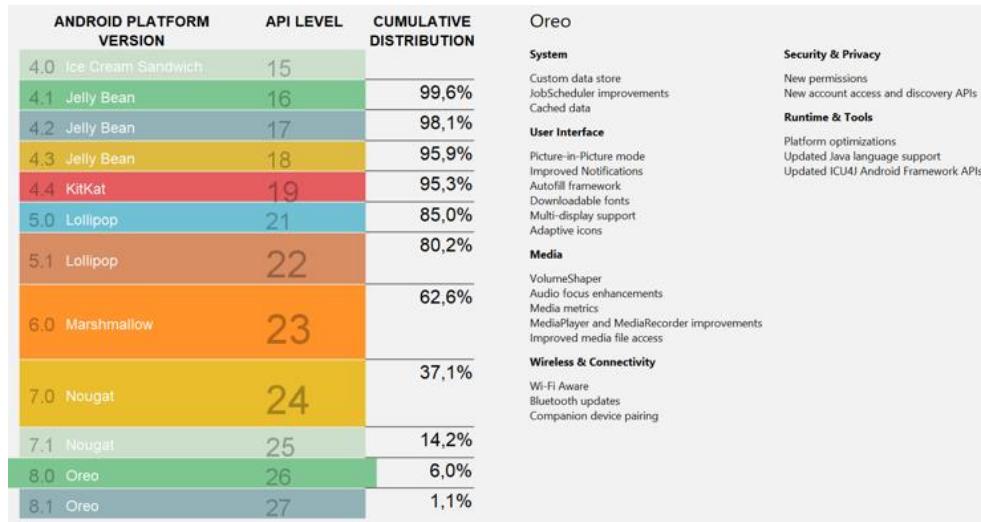
Nota sobre Java/Kotlin: *El nombre del paquete debe ser único en todos los paquetes instalados en un sistema.* Por ello, cuando quieras distribuir una aplicación, es muy importante utilizar un dominio que no puedan estar utilizando otras empresas (por ejemplo: es.upv.elgranlibroandroid.proyecto1). El espacio de nombres “com.example” está reservado para la documentación de ejemplos (como en este libro) y nunca puede ser utilizado para distribuir aplicaciones. De hecho, Google Play no permite publicar una aplicación si su paquete comienza por “com.example”.

Save location: Permite configurar la carpeta donde se almacenarán todos los ficheros del proyecto.

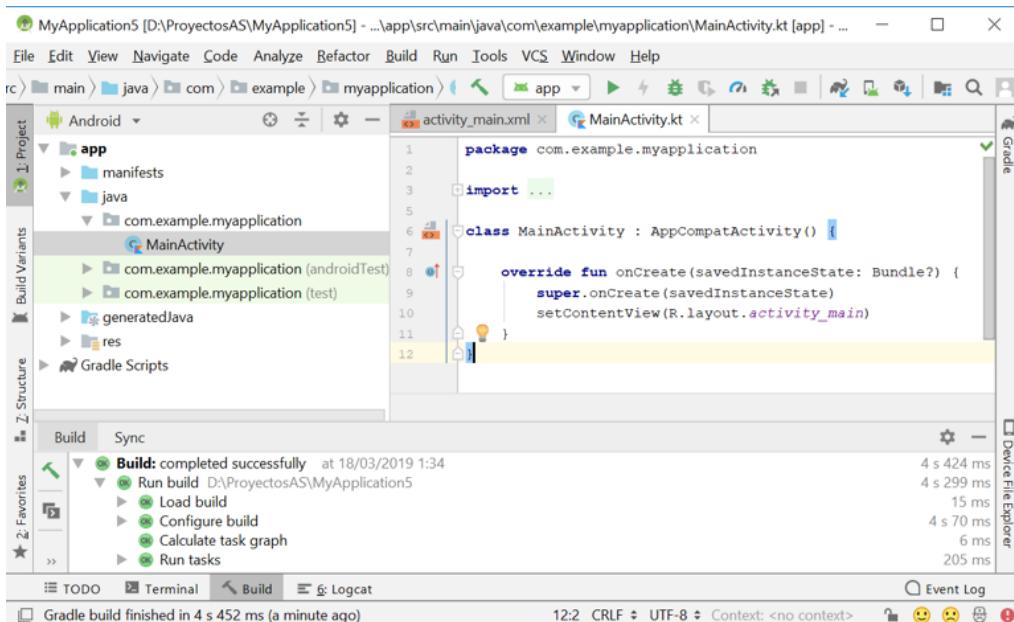
Language: Selecciona Java o Kotlin, según el lenguaje con el que quieras programar la aplicación.

Minimum API level: Este valor especifica el mínimo nivel de la API que requiere tu aplicación. Por lo tanto, la aplicación no podrá ser instalada en dispositivos con una versión inferior. Procura escoger valores pequeños para que tu aplicación pueda instalarse en la mayoría de los dispositivos. Un valor adecuado puede ser el nivel de API 16 (v4.1), dado que cubriría prácticamente el 100% de los dispositivos. O el nivel de API 19 (v4.4), que cubriría más del 95% de los dispositivos. Escoger valores pequeños para este parámetro tiene un inconveniente: no podremos utilizar ninguna de las mejoras que aparezcan en los siguientes niveles de API. Por

ejemplo, si queremos utilizar el motor de animaciones de propiedades en nuestra aplicación, tendremos que indicar en este campo la versión 3.0, dado que esta API no aparece hasta esta versión. Pero, en este caso, nuestra aplicación no se podrá instalar en la versión 2.3. Como se acaba de indicar escoger la versión mínima del SDK es un aspecto clave a la hora de crear un proyecto. Para ayudarnos a tomar esta decisión se indica en negrita el porcentaje de dispositivo donde se podrá instalar nuestra aplicación. En el apartado anterior se explica cómo se obtiene esta información. Pulsa en *Help Me choose* para visualizar una gráfica donde se muestra los diferentes niveles de API y el porcentaje de usuarios que podrán instalarla la aplicación. Además si pulsas sobre un nivel te mostrará un resumen con las nuevas características introducidas en este nivel.



- Deja el resto de valores por defecto y pulsa **Finish** para crear el proyecto. Deberás tener visible el explorador del proyecto (Project) a la izquierda. Abre el fichero **MainActivity** (situado en **app > java > com.example.myapplication**). Debe tener este aspecto:



Observa que la clase MainActivity extiende AppCompatActivity que a su vez es un descendiente de Activity. Una *actividad* es una entidad de aplicación que se utiliza para representar cada una de las pantallas de nuestra aplicación. Es decir, el usuario interactúa con solo una de estas actividades y va navegando entre ellas. El sistema llamará al método onCreate() cuando comience su ejecución. Es donde se debe realizar la inicialización y la configuración de la interfaz del usuario. Las actividades van a ser las encargadas de interactuar con el usuario.

Nota sobre Java/Kotlin: Antes de este método se ha utilizado la anotación @Override (sobrescribir). Esto indica al compilador que el método ya existe en la clase padre y queremos reemplazarlo. Es opcional, aunque conviene incluirlo para evitar errores.

Lo primero que hay que hacer al sobrescribir un método suele ser llamar al método de la clase de la que hemos heredado. Para referirnos a nuestra clase padre usaremos la palabra reservada super. El método termina indicando que la actividad va a visualizarse en una determinada vista. Esta vista está definida en los recursos. Más adelante también se describe la finalidad de cada fichero y carpeta de este proyecto

video [Un primer proyecto con Android Studio](#)

Ejecución del programa

Objetivos:

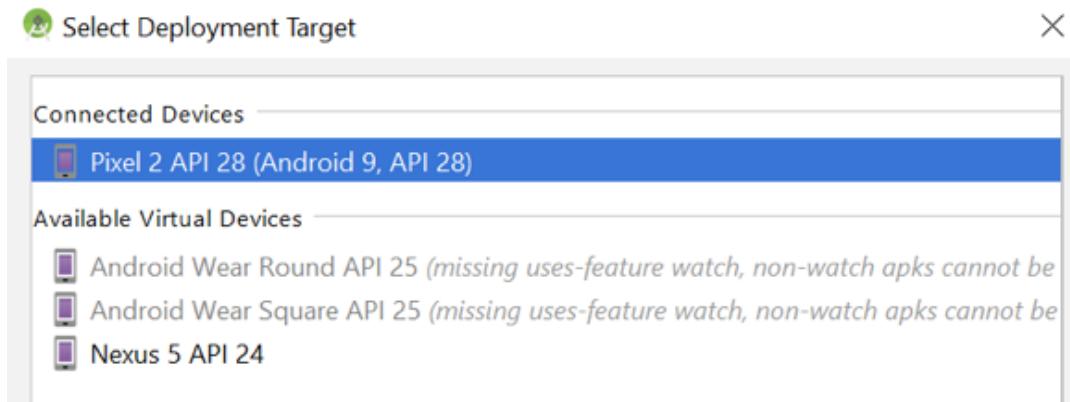
Veremos dos alternativas para ejecutarla: en un emulador y en un dispositivo real. Esta lección no tiene vídeo, pulsa en el botón "Versión texto" para acceder al contenido.

Una vez creada esta primera aplicación vamos a ver dos alternativas para ejecutarla: en un emulador y en un dispositivo real.

Ejecución en el emulador

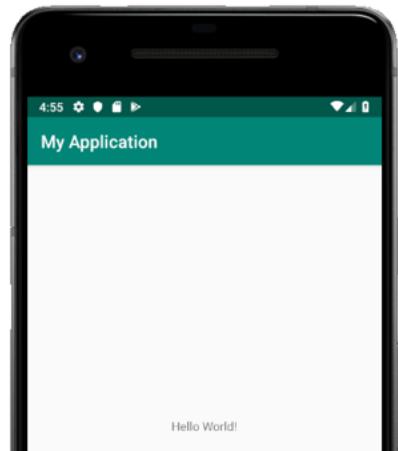
Ejercicio: Ejecución en el emulador

1. Selecciona **Run >Run 'app'** (Mayús-F10) o pulsa el icono ➤ de la barra de herramientas.
2. Te preguntará sobre que dispositivo quieres ejecutar la aplicación:



Te permite escoger entre dispositivos conectados (AVD o reales), lanzar un AVD ya creado o crear uno nuevo.

3. Una vez que el emulador esté cargado, debes ver algo así:



Ejecución en un terminal real

También es posible ejecutar y depurar tus programas en un terminal real. Incluso es una opción más rápida y fiable que utilizar un emulador. No tienes más que usar un cable USB para conectar el terminal al PC. Resulta imprescindible haber instalado un *driver* especial en el PC. Puedes encontrar un *driver* genérico que se encuentra en la carpeta de instalación del SDK `\sdk\extras\google\usb_driver`. Aunque lo más probable es que tengas que utilizar el *driver* del fabricante. También es muy probable que el driver lo instale Windows por su cuenta.

Ejercicio: Ejecución en un terminal real

1. Abre *Android SDK Manager* y asegúrate de que está instalado el paquete USB Driver. En caso contrario, instálalo.

Name	Version	Status
Android Support Repository, rev 25	25.0.0	Installed
Android Support Library, rev 23.1.1	23.1.1	Installed
Android Auto Desktop Head Unit emulator	1.1.0	Not installed
Google Play services, rev 28	28.0.0	Installed
Google Repository, rev 23	23.0.0	Installed
Google Play APK Expansion Library	3.0.0	Not installed
Google Play Billing Library	5.0.0	Not installed
Google Play Licensing Library, rev 2	2.0.0	Installed
Android Auto API Simulators	1.0.0	Not installed
Google USB Driver, rev 11	11.0.0	Installed
Google Web Driver	2.0.0	Not installed
Intel x86 Emulator Accelerator (HAXM installer), rev 5.5	5.5.0	Installed
Android NDK	1.0.0	Not installed

2. Posiblemente, este *driver* genérico no sea adecuado para tu terminal y tengas que utilizar el del fabricante. Si no dispones de él, puedes buscarlo en:

<http://developer.android.com/tools/extras/oem-usb.html>

3. A partir de Android 4.2 las opciones para desarrolladores vienen ocultas por defecto. De esta forma, un usuario sin experiencia no podrá activar estas opciones de forma accidental. Para activar las opciones de desarrollo tienes que ir a Ajustes > Información del teléfono y pulsar siete

veces sobre el número de compilación. Tras esto aparecerá el mensaje “¡Ahora eres un desarrollador!” y nos mostrará más ajustes

4. En el terminal accede al menú *Ajustes > Opciones de desarrollador* y asegúrate de que la opción *Depuración de USB* está activada.



5. Conecta el cable USB.
6. Se indicará que hay un nuevo *hardware* y te pedirá que le indiques el controlador.

NOTA: En Windows, si indicas un controlador incorrecto no funcionará. Además, la próxima vez que conectes el cable no te pedirá la instalación del controlador. Para desinstalar el controlador sigue los siguientes pasos:

1. Asegúrate de haber desinstalado el controlador incorrecto.
2. Accede al registro del sistema (Inicio > ejecutar > RegEdit). Busca la siguiente clave y bórrala: «vid_0bb4&pid_0c02».
3. Vuelve al paso 3 del ejercicio.
7. Selecciona de nuevo **Run >Run ‘app’** (Mayús-F10) o pulsa el icono ➤. Aparecerá una ventana que te permite escoger en qué dispositivo o emulador quieras ejecutar la aplicación:
8. Selecciona el dispositivo real y pulsa *OK*.

Ficheros y directorios de un proyecto Android

Objetivos:

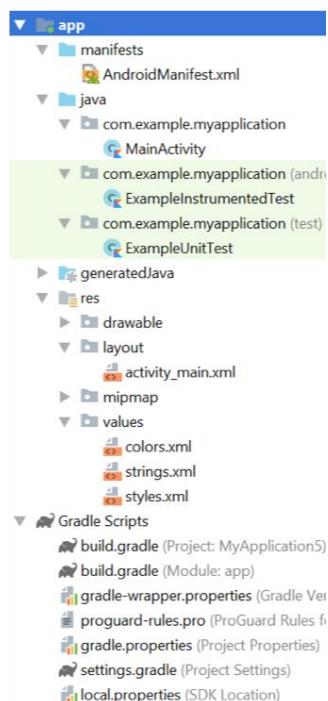
1. Describir la estructura de un proyecto Android

Video [Ficheros y directorios](#)

2. Enumerar los ficheros y carpetas donde se almacenan los proyectos

Lo primero que conviene que conozcas es que **un proyecto en Android Studio puede contener varios módulos**. Cada módulo corresponde a una aplicación o ejecutable diferente. Disponer de varios módulos en un mismo proyecto nos será muy útil cuando queramos crear varias versiones de nuestra aplicación, para dispositivo móvil, Wear, TV, Things, etc. También si queremos crear varias versiones de nuestra aplicación con nivel mínimo de SDK diferentes. En este libro solo vamos a desarrollar aplicaciones para móviles, por lo que no vamos a necesitar un módulo ¿no será “solo vamos a necesitar un m”? . Este módulo ha sido creado con el nombre *app*.

Cada módulo en Android está formado por un descriptor de la aplicación (*manifests*), el código fuente en Java (*java*), una serie de ficheros con recursos (*res*) y ficheros para construir el módulo (*Gradle Scripts*). Cada elemento se almacena en una carpeta específica, que hemos indicado entre paréntesis. Aprovecharemos el proyecto que acabamos de crear para estudiar la estructura de un proyecto en Android Studio. No te asustes con el exceso de información. Más adelante se dará más detalles sobre la finalidad de cada fichero.



AndroidManifest.xml: Este fichero describe la aplicación Android. Se define su nombre, paquete, icóno, estilos, etc. Se indican las *actividades*, las *intenciones*, los *servicios* y los *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica el paquete Java, la versión de la aplicación, etc.

java: Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en carpetas según el nombre de su paquete.

MainActivity: Clase con el código de la actividad inicial.

ExampleInstrumentTest: Clase pensada para insertar código de testeo de la aplicación.

ExampleUnitTest: Clase para insertar test unitarios sobre otras aplicaciones.

res: Carpeta que contiene los recursos usados por la aplicación.

drawable: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.

mipmap: En una carpeta guardaremos el icono de la aplicación. En el proyecto se ha incluido el fichero ic_launcher.png que será utilizado como ícono de la aplicación. Observa cómo este recurso se ha añadido en seis versiones diferentes. Como veremos en el siguiente capítulo, usaremos un sufijo especial si queremos tener varias versiones de un recurso, de forma que solo se cargue al cumplirse una determinada condición. Por ejemplo: (hdpi) significa que solo ha de cargar los recursos contenidos en esta carpeta cuando el dispositivo donde se instala la aplicación tenga una densidad gráfica alta (180- dpi); (mdpi) se utilizará con densidad gráfica alta (180- dpi). Si pulsas sobre las diferentes versiones del recurso, observarás como se trata del mismo ícono, pero con más o menos resolución de forma que, en función de la densidad gráfica del dispositivo, se ocupe un tamaño similar en la pantalla. Véase el apartado 2.6 del anexo E para más detalles. El fichero ic_launcher_round.png es similar, pero para cuando se quieren usar iconos redondos.

layout: Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web. Se tratarán en el siguiente capítulo.

menu: Ficheros XML con los menús de cada actividad. En el proyecto no hay ningún menú por lo que no se muestra esta carpeta.

values: También utilizaremos ficheros XML para indicar valores usados en la aplicación, de esta manera podremos cambiarlos desde estos ficheros sin necesidad de ir al código fuente. En *colors.xml* se definen los tres colores primarios de la aplicación. En *dimens.xml*, se pueden definir dimensiones como el margen por defecto o el ancho de los botones. En el fichero *strings.xml*, tendrás que definir todas las cadenas de caracteres de tu aplicación. Creando recursos alternativos resultará muy sencillo traducir una aplicación a otro idioma. Finalmente, en *styles.xml*, podrás definir los estilos y temas de tu aplicación. Se estudian en el siguiente capítulo.

anim: Contiene ficheros XML con animaciones de vistas (Tween). Las animaciones se describen al final del capítulo 4.

animator: Contiene ficheros XML con animaciones de propiedades.

xml: Otros ficheros XML requeridos por la aplicación.

raw: Ficheros adicionales como binarios por ejemplo que no se encuentran en formato XML.

Gradle Scripts: En esta carpeta se almacenan una serie de ficheros Gradle que permiten compilar y construir la aplicación. Observa como algunos hacen referencia al módulo app y el resto son para configurar todo el proyecto. El fichero más importante es *build.gradle (Module:app)* que es donde se configuran las opciones de compilación del módulo:

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 29
    defaultConfig {
        applicationId "com.example.myapplication"
        minSdkVersion 19
        targetSdkVersion 29
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'pr...
        }
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
}
```

El primer parámetro que podemos configurar es `compileSdkVersion` que nos permite definir la versión del sdk con la que compilamos la aplicación. Las nuevas versiones no solo añaden funcionalidades al API, también añaden mejoras en los procesos. Por ejemplo, a partir de la versión 3.0 (API 11) solo se permite el acceso a Internet desde un hilo auxiliar[\[2\]](#). `applicationId` suele coincidir con el nombre del paquete Java creado para la aplicación. Se utiliza como identificador único de la aplicación, de forma que no se permite instalar una aplicación si ya existe otra con el mismo id. `minSdkVersion` especifica el nivel mínimo de API que requiere la aplicación. Es un parámetro de gran importancia, la aplicación no podrá ser instalada en dispositivos con versiones anteriores y solo podremos usar las funcionalidades del API hasta este nivel (con excepción de las librerías de

compatibilidad). targetSdkVersion indica la versión más alta con la que se ha puesto a prueba la aplicación. Cuando salgan nuevas versiones del SDK tendrás que comprobar la aplicación con estas versiones y actualizar el valor. versionCode y versionName indica la versión de tu aplicación. Cada vez que publies una nueva versión incrementa en uno el valor de versionCode y aumenta el valor de versionName según la importancia de la actualización. Si es una actualización menor el nuevo valor podría ser "1.1" y si es mayor "2.0".

Dentro de buildTypes se añaden otras configuraciones dependiendo del tipo de compilación que queramos (relase para distribución, debug para depuración, etc.). Los comandos que aparecen configuran la ofuscación de código. Para más información leer el capítulo «Ingeniería Inversa en Android» de *El Gran Libro de Android Avanzado*.

Un apartado importante es el de dependencies. En él has de indicar todas las **librerías que han de ser incluidas en nuestro proyecto**. Si necesitas usar alguna librería de compatibilidad adicional has de incluirla aquí.

Componentes de una aplicación

Objetivos:

Describir los componentes principales usados en una aplicación Android.

Vídeo [Componentes de una aplicación en Android](#)

Existe una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del curso se describirán con más detalle las clases Java que implementan cada uno de estos componentes.

Vista (View)

Las *vistas* son los elementos que componen la interfaz de usuario de una aplicación: por ejemplo, un botón o una entrada de texto. Todas las vistas van a ser objetos descendientes de la clase View, y por tanto, pueden ser definidas utilizando código Java. Sin embargo, lo habitual será definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.

Layout

Un *layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de *layouts* para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los *layouts* también son objetos descendientes de la clase View. Igual que las vistas, los *layouts* pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

Actividad (Activity)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como actividad. Su función principal es la creación de la interfaz de usuario. Una aplicación suele necesitar varias actividades para crear la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Una actividad se define en una clase descendiente de Activity y utiliza un layout para que defina su apariencia.

Fragментos (Fragment)

La llegada de las tabletas trajo el problema de que las aplicaciones de Android ahora deben soportar pantallas más grandes. Si diseñamos una aplicación pensada para un dispositivo móvil y luego la ejecutamos en una tableta, el resultado no suele resultar satisfactorio.

Para ayudar al diseñador a resolver este problema, en la versión 3.0 de Android aparecen los *fragments*. Un *fragment* está formado por la unión de varias vistas para crear un bloque

funcional de la interfaz de usuario. Una vez creados los fragments, podemos combinar uno o varios *fragments* dentro de una actividad, según el tamaño de pantalla disponible.

Vídeo : [Los fragments en Android](#)

El uso de *fragments* puede ser algo complejo, por lo que recomendamos dominar primero conceptos como actividad, vista y layout antes de abordar su aprendizaje. No obstante, es un concepto importante en Android y todo programador en esta plataforma ha de saber utilizarlos. Véase el anexo A para aprender más sobre *fragments*.

Servicio (Service)

Un servicio es un proceso que se ejecuta “detrás”, sin la necesidad de una interacción con el usuario. Es algo parecido a un *demonio* en Unix o a un *servicio* en Windows. Se utilizan cuando queremos tener en ejecución un código de manera continua, aunque el usuario cambie de actividad. En Android disponemos de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso, y servicios remotos, que son ejecutados en procesos separados. Los servicios se estudian en el capítulo 8.

Intención (Intent)

Una intención representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- *Lanzar una actividad
- *Lanzar un servicio
- *Enviar un anuncio de tipo broadcast
- *Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las intenciones para el intercambio de información entre estos componentes.

Receptor de anuncios (Broadcast Receiver)

Un *receptor de anuncios* recibe anuncios *broadcast* y reacciona ante ellos. Los anuncios *broadcast* pueden ser originados por el sistema (por ejemplo: *Batería baja*, *Llamada entrante*) o por las aplicaciones. Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios *broadcast*. Los receptores de anuncios no disponen de interfaz de usuario, aunque pueden iniciar una actividad si lo estiman oportuno. Los receptores de anuncios se estudian en el CAPÍTULO 8.

Proveedores de Contenido (Content Provider)

En muchas ocasiones las aplicaciones instaladas en un terminal Android necesitan compartir información. Android define un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones. Los *Content Provider* son estudiados en el 9.

Documentación y ApiDemos

Objetivos:

Donde encontrar documentación para desarrollar aplicaciones Android.

Aunque en este libro vas a aprender mucho, resultaría imposible tocar todos los aspectos de Android y con un elevado nivel de profundidad. Por lo tanto, resulta imprescindible que dispongas de fuentes de información para consultar los aspectos que vayas necesitando. En este apartado te proponemos dos alternativas: el acceso a documentación sobre Android y el estudio de ejemplos.

Donde encontrar documentación

Puedes encontrarla una completa documentación del SDK localmente en:

...\\sdk\\docs\\index.html

Se incluye la descripción de todas las clases (*Develop > Reference*), conceptos clave y otro tipo de recursos. Esta documentación también está disponible en línea a través de Internet:

<http://developer.android.com/>

Muchos de los recursos utilizados en este libro puedes encontrarlos en:

<http://www.androidcurso.com/>

Para resolver dudas puntuales sobre programación te recomendamos la Web de preguntas y respuestas:

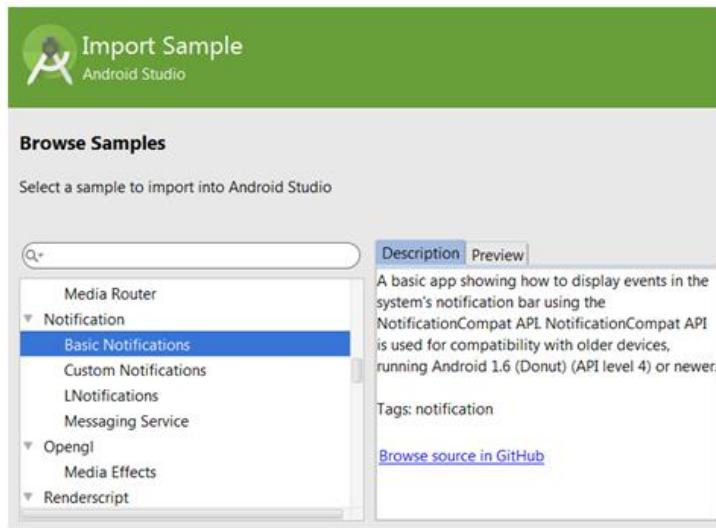
<http://stackoverflow.com/>

Repositorio de ejemplos de GitHub

Otra opción muy interesante para aprender nuevos aspectos de programación consiste en estudiar ejemplos. Google ha preparado un repositorio de ejemplos en GitHub que pueden ser instalados desde **Android Studio**.

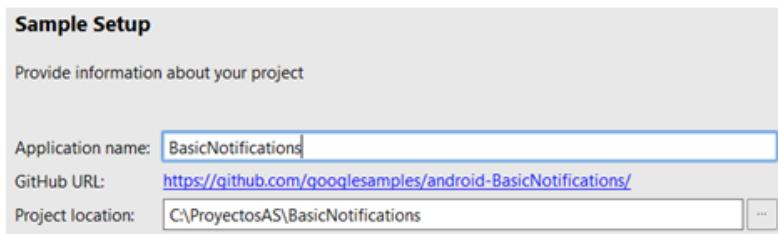
Ejercicio: *Instalación de un ejemplo desde GitHub*

1. Selecciona **File > New > Import Sample...**. Aparecerá la siguiente ventana:



Los proyectos se encuentran clasificados en categorías: *Admin*, *Background*, *Connectivity*, *Content*, *Input*, *Media*, *Notification*, ... Selecciona un proyecto de alguna de estas categorías. A la derecha podrás leer una breve descripción o ver una vista previa.

2. Pulsa *Next* para pasar a la siguiente ventana. Podrás configurar el nombre de la aplicación, explorar el proyecto accediendo a su sitio web en GitHub e indicar la carpeta donde quieras descargarlo:



3. Pulsa *Finish* y a continuación ejecuta el proyecto seleccionado.

La aplicación ApiDemos

La aplicación ApiDemos suele estar instalada en mucho de los AVD. Está formada por cientos de ejemplos, donde no solo podrás ver las funcionalidades disponibles en la API de Android, sino que además podrás estudiar su código[1].

Ejercicio: Instalación de ApiDemos en Android Studio

Para crear un proyecto con la aplicación ApiDemos, sigue los siguientes pasos :

1. Desde un dispositivo AVD abre API Demos. Los dispositivos con versión 7.0 suelen tener preinstalada esta aplicación.
2. Prueba algunos de los cientos de ejemplos que incorpora.
3. Descarga el fichero siguiente y descomprime en una carpeta:

<http://androidcurso.com/images/dcomg/ficheros/ApiDemos22.zip>

4. Selecciona **File > New > Import Project...** y selecciona la carpeta.
5. Una vez corregido algún pequeño error, ya puedes ejecutar ApiDemos. Verás como los diferentes ejemplos se organizan por carpetas. En el nivel superior tenemos:

Accessibility: Aspectos de accesibilidad, como *trackball*, *touch* o texto a voz.

Animation: Una gran variedad de efectos y animaciones.

App: Trabajando a nivel de aplicación con *Activity*, *Alarm*, *Dialog*, *Service*, etc.

Content: Describe cómo leer datos desde ficheros, recursos y archivos XML.

Graphics: Una gran cantidad de ejemplos gráficos tanto en 2D como en 3D.

Media: Reproducción de audio y vídeo con las clases *MediaPlayer* y *VideoView*.

NFC: Ejemplos de uso de *Near Field Communication*.

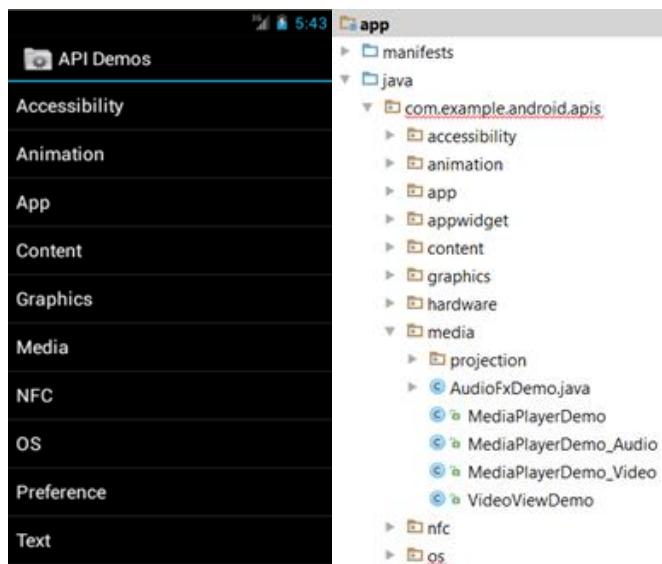
OS: Servicios del sistema operativo. Incluye sensores, vibrador o envío de SMS.

Preference: Varios ejemplos de uso de preferencias.

Security: Contiene un ejemplo sobre encriptación.

Text: Diferentes ejemplos de manipulación y visualización de texto.

Views: Android utiliza como elemento básico de representación la clase *View* (vista). Tenemos a nuestra disposición una gran cantidad de descendientes de esta clase para representar una interfaz gráfica (botones, cajas de texto, entradas, etc.). Visualiza estos ejemplos para mostrar las diferentes posibilidades.



6. Prueba alguna de la demo que se incluye. Luego resulta sencillo localizar la clase Java que la implementa, buscando en el explorador del proyecto.

[1] https://github.com/android/platform_development/tree/master/samples/ApiDemos

Depurar aplicaciones con el entorno de desarrollo

Objetivos:

Se describe el uso de perspectivas, la ejecución paso a paso del código, la visualización y modificación de variables y la introducción de puntos de ruptura.

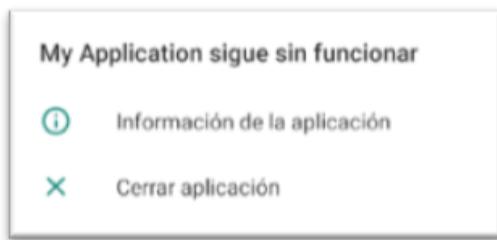
Programación y errores de código son un binomio inseparable. Por lo tanto, resulta fundamental sacar el máximo provecho a las herramientas de depuración.

Android Studio integra excelentes herramientas para la depuración de código. Para probarlos, introduce un error en tu código modificando MainActivity de forma que en método onCreate() tenga este código:

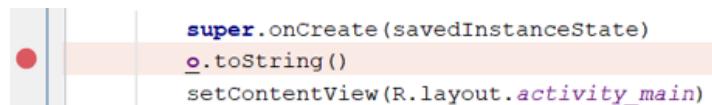
```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    Object o = null;  
    o.toString();  
    setContentView(R.layout.activity_main);  
}
```

Este cambio introduce en **Java** un NullPointerException

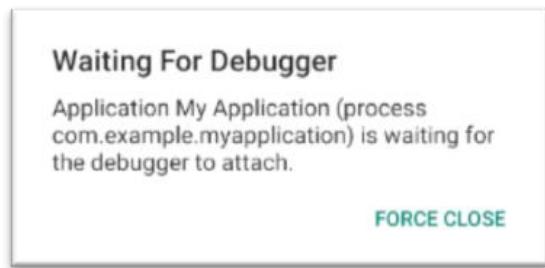
En **Kotlin** un UninitializedPropertyAccessException. Si ahora ejecutas tu aplicación, te aparecerá algo similar a:



Pulsa Cerrar para finalizar la aplicación. Para averiguar más sobre el error, inserta un punto de ruptura (breakpoint) en el código fuente en la línea o.toString() (el breakpoint se introduce haciendo clic en la barra de la izquierda).



Entonces selecciona *Run > Debug 'app'* (Mayús+F9) o pulse en para ejecutarlo en modo *Debug*. Tu aplicación se reiniciará en el emulador mostrando el siguiente mensaje:



Pero esta vez quedará suspendida cuando alcance el punto de ruptura que has introducido. Entonces puedes recorrer el código en modo *Debug*, igual que se haría en cualquier otro entorno de programación. Pulsa en *Run > Step Over (F8)* para ir ejecutando las líneas una a una.

Video [Depurar con Android Studio](#)

Depurar aplicaciones con LogCat

Marcar esta página

Objetivos:

Se muestra el uso de LogCat en Android para depurar aplicaciones dentro del entorno de desarrollo

El sistema Android utiliza el fichero *LogCat* para registrar todos los problemas y eventos principales que ocurren en el sistema. Ante cualquier error resulta muy interesante consultarlos para tratar de encontrar su origen.

La clase Log proporciona un mecanismo para introducir mensajes desde nuestro código en este fichero. Puede ser muy útil para depurar nuestros programas o para verificar el funcionamiento del código. Disponemos de varios métodos para generar distintos tipos de mensajes:

Log.e(): Errors
Log.w(): Warnings
Log.i(): Information
Log.d(): Debugging
Log.v(): Verbose

Ejercicio paso a paso: Depurar con mensajes Log

1. Modifica la clase MainActivity introduciendo la línea que aparece en subrayada:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    Log.d("HolaMundo","Entramos en onCreate");
    super.onCreate(savedInstanceState);
    Object o = null;
    o.toString();
    setContentView(R.layout.activity_main);
}
```

Nota sobre Java/Kotlin: Para poder utilizar la clase Log has de importar un nuevo paquete. Para ello añade al principio **import android.util.Log;** Otra alternativa es pulsar **Alt-Intro** para que se añadan automáticamente los paquetes que faltan. En algunos casos, el sistema puede encontrar dos paquetes con la clase Log, y puede tener dudas sobre cual importar. En estos casos te preguntará.

2. Ejecuta la aplicación. Aparecerá un error.
3. En **Android Studio** aparecerá automáticamente en la parte inferior:

The screenshot shows the Android Studio Logcat window. The title bar says "Logcat". Below it are several dropdown menus: "Emulator Pixel_2", "com.example.myap", "Verbose", "Q", "Regex", and "Show only selected". The main area displays a log message from March 18, 2019, at 19:15:30. It starts with some initial logs and then a fatal exception:

```

2019-03-18 19:15:30.261 25705-25705/com.example.myapplication D/HolaMundo: Entramos en onCreate
2019-03-18 19:15:30.428 25705-25705/com.example.myapplication D/AndroidRuntime: Shutting down VM
2019-03-18 19:15:30.437 25705-25705/com.example.myapplication E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.example.myapplication, PID: 25705
    java.lang.RuntimeException: Unable to start activity ComponentInfo{com.example.myapplication/c
        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2913)
        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:3048)
        at android.app.servertransaction.LaunchActivityItem.execute(LaunchActivityItem.java:78)
        at android.app.servertransaction.TransactionExecutor.executeCallbacks(TransactionExecutor.java:68)
        at android.app.servertransaction.TransactionExecutor.execute(TransactionExecutor.java:60)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1808)
        at android.os.Handler.dispatchMessage(Handler.java:106)
        at android.os.Looper.loop(Looper.java:193)
        at android.app.ActivityThread.main(ActivityThread.java:6669) <1 internal call>
        at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:493)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:858)
Caused by: kotlin.UninitializedPropertyAccessException: lateinit property o has not been init
    at com.example.myapplication.MainActivity.onCreate(MainActivity.kt:14)
    at android.app.Activity.performCreate(Activity.java:7136)
    at android.app.Activity.performCreate(Activity.java:7127)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1271)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2893) <8 more...>
2019-03-18 19:15:30.526 25705-25705/com.example.myapplication I/Process: Sending signal. PID: 25705

```

Below the log, there are tabs for "TODO", "Terminal", "Build", "Logcat" (which is selected), "Profiler", "Run", "Debug", and "Event Log".

En la primera línea de la captura anterior, comprobamos que se pudo entrar dentro de `onCreate()`. Dos líneas más abajo se indica una excepción. La información mostrada suele ser excesiva. Te recomendamos que busques las palabras “Caused by” para ver el tipo de excepción y la primera referencia a un paquete escrito por nosotros,

“`com.example.jtomas.myapplication`”. En este ejemplo, las líneas clave son: **en Java “Cased by: `java.lang.NullPointerException` at**

`com.example.jtomas.myapplication.MainActivity.onCreate(MainActivity.java:17)`.”.

4. Haz clic en `(MainActivity.java:17)` o `(MainActivity.kt:14)`. Te abrirá la actividad `MainActivity` y te situará en la línea donde se ha producido el error.

video [LogCat con Android Studio](#)

Código completo de la aplicación de ejemplo:

Clase Principal

```
package com.example.mislugares;
class Principal
{
    public static void main(String[] main)
    {
        LugaresLista lugares = new LugaresLista();
        lugares.anyadeEjemplos();
        System.out.print(lugares.tamanyo());
        for (int i=0; i<lugares.tamanyo(); i++)
        {
            System.out.println(lugares.elemento(i).toString());
        }
    }
}//class Principal
```

Clase Lugar

```
package com.example.mislugares;

public class Lugar //Datos de un Lugar del mundo
{
    private String nombre;
    private String direccion;
    private GeoPunto posicion;
    private String foto;
    private int telefono;
    private String url;
    private String comentario;
    private long fecha;
    private float valoracion;
    private TipoLugar tipo;

    /** Crea un Lugar vacío
     */
    public Lugar()
    {
        fecha = System.currentTimeMillis();
        posicion = GeoPunto.SIN_POSICION;
        tipo = TipoLugar.OTROS;
    }
    /** Conjunto de datos de un Lugar del mundo
     * El campo fecha se actualiza con la fecha de instanciación del objeto
     * @param nombre
     * @param direccion
     * @param Longitud Longitud y Latitud. Objeto GeoPunto
     * @param Latitud
     * @param telefono
     * @param url
     * @param comentario
     * @param valoracion
     */
    public Lugar(String nombre, String direccion, double longitud,
                double latitud,TipoLugar tipo, int telefono,
                String url, String comentario,int valoracion)
    {
        fecha = System.currentTimeMillis();
        posicion = new GeoPunto(longitud, latitud);
        this.nombre = nombre;
        this.direccion = direccion;
```

```

        this.tipo = tipo;
        this.telefono = telefono;
        this.url = url;
        this.comentario = comentario;
        this.valoracion = valoracion;
    }
    // Getters, setters y toString
    public String getNombre() {
        return nombre;
    }
    public String getDireccion() {
        return direccion;
    }
    public GeoPunto getPosicion() {
        return posicion;
    }
    public String getFoto() {
        return foto;
    }
    public int getTelefono() {
        return telefono;
    }
    public String getUrl() {
        return url;
    }
    public String getComentario() {
        return comentario;
    }
    public long getFecha() {
        return fecha;
    }
    public float getValoracion() {
        return valoracion;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }
    public void setPosicion(GeoPunto posicion) {
        this.posicion = posicion;
    }
    public void setFoto(String foto) {
        this.foto = foto;
    }
    public void setTelefono(int telefono) {
        this.telefono = telefono;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public void setComentario(String comentario) {
        this.comentario = comentario;
    }
    public void setFecha(long fecha) {
        this.fecha = fecha;
    }
    public void setValoracion(float valoracion) {
        this.valoracion = valoracion;
    }

    @Override
    public String toString() {
        return "Lugar{" +
            "nombre=" + nombre + '\'' +

```

```

        ", direccion='" + direccion + '\'' +
        ", posicion='" + posicion +
        ", foto='" + foto + '\'' +
        ", telefono='" + telefono +
        ", url='" + url + '\'' +
        ", comentario='" + comentario + '\'' +
        ", fecha='" + fecha +
        ", valoracion='" + valoracion +
        ", tipo='" + tipo +
        '}';
    }

    public TipoLugar getTipo() {
        return tipo;
    }

    public void setTipo(TipoLugar tipo) {
        this.tipo = tipo;
    }
}

```

Clase GeoPunto

```

package com.example.mislugares;

import java.util.Objects;

public class GeoPunto
{
    private double longitud, latitud;

    /**SIN_POSICION Es un objeto GeoPunto = null
     *
     */
    static public GeoPunto SIN_POSICION = new GeoPunto(0.0,0.0);

    public GeoPunto(double longitud, double latitud) {
        this.longitud= longitud;
        this.latitud= latitud;
    }
    public String toString() {
        return new String("longitud:" + longitud + ", latitud:" + latitud);
    }
    /**Calcula la diferencia entre la posición del objeto y el introducido como
    parámetro
     *
     * @param punto otro punto con el que calcular la diferencia
     * @return distancia entre puntos tipo double
     */
    public double distancia(GeoPunto punto) {
        final double RADIO_TIERRA = 6371000; // en metros
        double dLat = Math.toRadians(latitud - punto.latitud);
        double dLon = Math.toRadians(longitud - punto.longitud);
        double lat1 = Math.toRadians(punto.latitud);
        double lat2 = Math.toRadians(latitud);
        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
            Math.sin(dLon/2) * Math.sin(dLon/2) *
            Math.cos(lat1) * Math.cos(lat2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
        return c * RADIO_TIERRA;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;

```

```

    if (o == null || getClass() != o.getClass()) return false;
    GeoPunto geoPunto = (GeoPunto) o;
    return Double.compare(geoPunto.longitud, longitud) == 0 &&
           Double.compare(geoPunto.latitud, latitud) == 0;
}

@Override
public int hashCode() {
    return Objects.hash(longitud, latitud);
}

public double getLongitud() {
    return longitud;
}

public void setLongitud(double longitud) {
    this.longitud = longitud;
}

public double getLatitud() {
    return latitud;
}

public void setLatitud(double latitud) {
    this.latitud = latitud;
}
}

```

Clase LugaresLista

```

package com.example.mislugares;
import java.util.ArrayList;
import java.util.List;
public class LugaresLista implements RepositorioLugares
{
    protected List listaLugares;
    public LugaresLista()
    {
        this.listaLugares = new ArrayList<Lugar>();
    }
    @Override
    public Lugar elemento(int id)
    {
        return (Lugar) listaLugares.get(id);
    }
    @Override
    public void anyade(Lugar lugar)
    {
        listaLugares.add(lugar);
    }
    @Override
    public int nuevo()
    {
        Lugar lugar = new Lugar();
        listaLugares.add(lugar);
        return listaLugares.size()-1;
    }
    @Override
    public void borrar(int id)
    {
        listaLugares.remove(id);
    }
    @Override
    public int tamanyo()
    {
        return listaLugares.size();
    }
}

```

```

        }

    @Override
    public void actualiza(int id, Lugar lugar)
    {
        listaLugares.set(id, lugar);
    }

    public void anyadeEjemplos() {
        anyade(new Lugar("Escuela Politécnica Superior de Gandía",
            "C/ Paranimf, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
            TipoLugar.EDUCACION, 962849300, "http://www.epsg.upv.es",
            "Uno de los mejores lugares para formarse.", 3));
        anyade(new Lugar("Al de siempre",
            "P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)",
            -0.190642, 38.925857, TipoLugar.BAR, 636472405, "",
            "No te pierdas el arroz en calabaza.", 3));
        anyade(new Lugar("androidcurso.com",
            "ciberespacio", 0.0, 0.0, TipoLugar.EDUCACION,
            962849300, "http://androidcurso.com",
            "Amplia tus conocimientos sobre Android.", 5));
        anyade(new Lugar("Barranco del Infierno",
            "Vía Verde del río Serpis. Villalonga (Valencia)",
            -0.295058, 38.867180, TipoLugar.NATURALEZA, 0,
            "http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-"+
            "verde-del-rio.html","Espectacular ruta para bici o andar",
            4));
        anyade(new Lugar("La Vital",
            "Avda. de La Vital, 0 46701 Gandía (Valencia)", -0.1720092,
            38.9705949, TipoLugar.COMPRAS, 962881070,
            "http://www.lavital.es/","El típico centro comercial", 2));
    }
}

```

Interface RepositorioLugares

```

package com.example.mislugares;
public interface RepositorioLugares {
    /**
     *
     * @param id
     * @return Objeto de la clase Lugar
     */
    Lugar elemento(int id); //Devuelve el elemento dado su id

    /** Añade un nuevo objeto de la clase Lugar al repositorio
     *
     * @param Lugar tipo Lugar
     */
    void anyade(Lugar lugar); //Añade el elemento indicado

    /** Añade un objeto en blanco de la clase Lugar al repositorio
     *
     * @return
     */
    int nuevo(); //Añade un elemento en blanco y devuelve su id

    /** Borra el objeto Lugar indizado por id
     *
     * @param id tipo int
     */
    void borrar(int id); //Elimina el elemento con el id indicado

    /**Devuelve el tamaño del repositorio
     *
     * @return número de elementos en el repositorio tipo int
     */
}

```

```
/*
int tamanyo(); //Devuelve el número de elementos
/**Reemplaza un Lugar por otro
 *
 * @param id
 * @param Lugar
 */
void actualiza(int id, Lugar lugar); //Reemplaza un elemento
}

Enum TipoLugar
package com.example.mislugares;
public enum TipoLugar
{
    OTROS ("Otros", 5),
    RESTAURANTE ("Restaurante", 2),
    BAR ("Bar", 6),
    COPAS ("Copas", 0),
    ESPECTACULO ("Espectáculo", 0),
    HOTEL ("Hotel", 0),
    COMPRAS ("Compras", 0),
    EDUCACION ("Educación", 0),
    DEPORTE ("Deporte", 0),
    NATURALEZA ("Naturaleza", 0),
    GASOLINERA ("Gasolinera", 0);

    private final String texto;
    private final int recurso;

    TipoLugar(String texto, int recurso) {
        this.texto = texto;
        this.recurso = recurso;
    }

    public String getTexto() { return texto; }
    public int getRecurso() { return recurso; }
}
```

Unidad 3 Diseño del interfaz del usuario- Vistas y Layouts

Introducción a la unidad

El diseño de la interfaz de usuario cobra cada día más importancia en el desarrollo de una aplicación. La calidad de la interfaz de usuario puede ser uno de los factores que conduzca al éxito o al fracaso de todo el proyecto.

Si has realizado alguna aplicación utilizando otras plataformas, advertirás que el diseño de la interfaz de usuario en Android sigue una filosofía muy diferente. En **Android la interfaz de usuario no se diseña en código, sino utilizando un lenguaje de marcado en XML** similar al HTML.

A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán entender el diseño de la interfaz de usuario en Android. Aunque no será la forma habitual de trabajar, comenzaremos creando la interfaz de usuario mediante código. De esta forma comprobaremos que cada uno de los elementos de la interfaz de usuario (las vistas) realmente son objetos Java. Continuaremos mostrando cómo se define la interfaz de usuario utilizando código XML. Pasaremos luego a ver las herramientas de diseño integradas en Android Studio. Se describirá el uso de **layouts**, que nos permitirá una correcta organización de las vistas, y el uso de recursos alternativos nos permitirá adaptar nuestra interfaz a diferentes circunstancias y tipos de dispositivos.

En este capítulo también comenzaremos creando la aplicación de ejemplo desarrollada a lo largo del curso, Asteroides. Crearemos la actividad principal, donde simplemente mostraremos cuatro botones, con los que se podrán arrancar diferentes actividades. A continuación aprenderemos a crear estilos y temas y los aplicaremos a estas actividades. Para terminar el capítulo propondremos varias prácticas para aprender a utilizar diferentes tipos de vistas y *layouts*.

Objetivos

- Entender cómo se realiza el diseño del interfaz de usuario en una aplicación Android.
- Aprender a trabajar con vistas y mostrar sus atributos más importantes.
- Enumerar los tipos de Layouts que nos permitirán organizar las vistas.
- Mostrar cómo se utilizan los recursos alternativos.
- Aprenderemos a crear estilos y temas para personalizar nuestras aplicaciones.
- Mostrar cómo interactuar con las vistas desde el código Java o Kotlin.
- Describir el uso de layouts basados en pestañas (*tabs*).

Creación de una interfaz de usuario por código

Objetivos:

Mostrar cómo crear una interfaz de usuario usando exclusivamente código Java.

Veamos un primer ejemplo de cómo crear una interfaz de usuario utilizando exclusivamente código Java. Aunque esta no es la forma recomendable de trabajar con Android, resulta interesante para resaltar algunos conceptos.

Ejercicio: Creación del interfaz de usuario por código

1. Abre el proyecto creado en el capítulo anterior y visualiza *MainActivity.java*. (El proyecto era el resultado de crear un nuevo proyecto con “Empty Activity” sin más)
2. Comenta la última sentencia del método *onCreate()* añade las tres subrayadas

```
@Override public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    //setContentView(R.layout.activity_main);  
    TextView texto = new TextView(this);  
    texto.setText("Hello, Android");  
    setContentView(texto);  
}
```

Nota sobre Java / Kotlin: Para poder utilizar el objeto *TextView* has de importar un nuevo paquete. Para ello añade al principio “import android.widget.TextView;”. Otra alternativa es pulsar *Alt-Intro*, para que se añadan automáticamente los paquetes que faltan.

La interfaz de usuario de Android está basada en una jerarquía de clases descendientes de la clase *View* (vista). Una vista es un objeto que se puede dibujar y se utiliza como un elemento en el diseño de la interfaz de usuario (un botón, una imagen, una etiqueta de texto como en el utilizado en el ejemplo, etc.). Cada uno de estos elementos se define como una subclase de la clase *View*; la subclase para representar un texto es *TextView*.

El ejemplo comienza creando un objeto de la clase *TextView*. El constructor de la clase acepta como parámetro una instancia de la clase *Context* (contexto). Un contexto es un manejador del sistema que proporciona servicios como la resolución de recursos, obtención de acceso a bases de datos o preferencias. La clase *Activity* es una subclase de *Context*, y como la clase *MainActivity* es una subclase de *Activity*, también es de tipo *Context*. Por ello, puedes pasar *this* (el objeto actual de la clase *MainActivity*) como contexto del *TextView*.

3. Después se define el texto que se visualizará en el *TextView* mediante *setText()*. Finalmente, mediante *setContentView()* se indica la vista utilizada por la actividad.
4. Ejecuta el proyecto para verificar que funciona.

Código del ejemplo

```
package com.example.myapplication;  
import androidx.appcompat.app.AppCompatActivity;  
import android.os.Bundle;
```

```
import android.util.Log;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        //Log.d("HolaMundo", "Entramos en onCreate");
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        TextView texto = new TextView(this);
        texto.setText("Hello, Android");
        setContentView(texto);
    }
}
```

Creación de una interfaz de usuario usando XML

Objetivos:

Mostrar cómo crear una interfaz de usuario usando XML.

En el ejemplo anterior hemos creado la interfaz de usuario directamente en el código. A veces puede ser muy complicado programar interfaces de usuario, ya que pequeños cambios en el diseño pueden corresponder a complicadas modificaciones en el código. Un principio importante en el diseño de *software* es que conviene separar todo lo posible el diseño, los datos y la lógica de la aplicación.

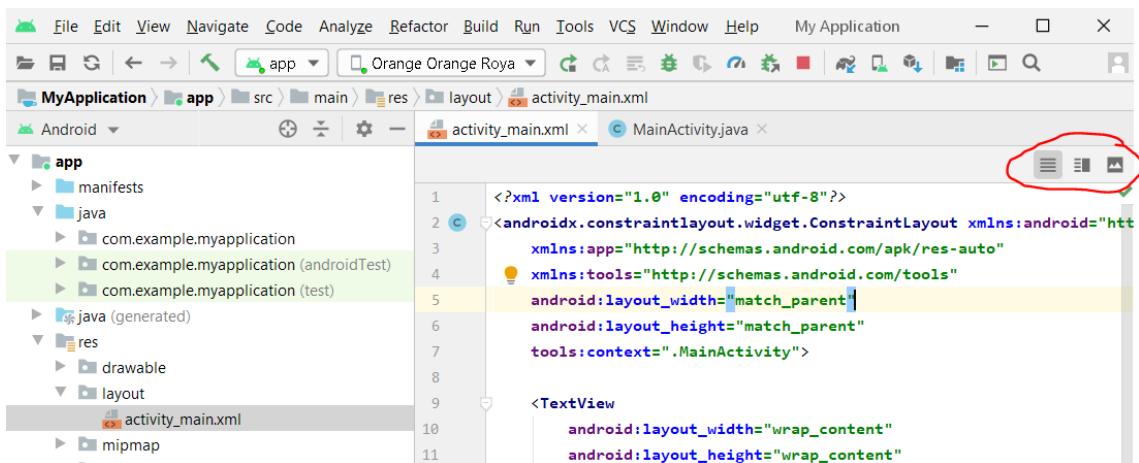
Android proporciona una alternativa para el diseño de interfaces de usuario: los ficheros de diseño basados en XML. Veamos uno de estos ficheros. Para ello accede al fichero *res/layout/activity_main.xml* de nuestro proyecto. Se muestra a continuación. Este *layout* o fichero de diseño proporciona un resultado similar al del ejemplo de diseño por código anterior:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

NOTA: Cuando haces doble clic en el explorador del proyecto sobre *activity_main.xml*, probablemente lo abra en modo gráfico. Para verlo en modo texto selecciona la pestaña *Text*(ver la siguiente figura)

Resulta sencillo interpretar su significado. Se introduce un elemento de tipo *ConstraintLayout*, cuya función como se estudiará más adelante es contener otros elementos de tipo *View*. Este *ConstraintLayout* tiene seis atributos. Los tres primeros, *xmlns:android*, *xmlns:app* y *xmlns:tools* son declaraciones de espacios de nombres de XML que utilizaremos en este fichero (este tipo de parámetro solo es necesario especificarlo en el primer elemento). Los dos siguientes permiten definir el ancho y alto de la vista. En el ejemplo se ocupará todo el espacio disponible. El último atributo indica la actividad asociada a este *Layout*.



Dentro del ConstraintLayout solo tenemos un elemento de tipo TextView. Este dispone de varios atributos. Los dos primeros definen el ancho y alto (se ajustará al texto contenido). El siguiente indica el texto a mostrar. Los cuatro siguientes indican la posición de la vista dentro del ConstraintLayout.



Ejercicio paso a paso: Creación del Interfaz de usuario con XML

1. Para utilizar el diseño en XML regresa al fichero *MainActivity.java* y deshaz los cambios que hicimos antes (elimina las tres últimas líneas y quita el comentario).
2. Ejecuta la aplicación y verifica el resultado. Ha de ser muy similar al anterior.
3. Modifica el valor de hello_world en el fichero *res/values/strings.xml*.
4. Vuelve a ejecutar la aplicación y visualiza el resultado.

Analicemos ahora la línea en la que acabas de quitar el comentario:

```
setContentView(R.layout.activity_main);
```

Aquí, *R.layout.main* corresponde a un objeto View que será creado en tiempo de ejecución a partir del recurso *activity_main.xml*. Trabajar de esta forma, en comparación con el diseño basado en código, no quita velocidad y requiere menos memoria. Este identificador es creado automáticamente en la clase R del proyecto a partir de los elementos de la carpeta *res*. La definición de la clase R puede ser similar a:

```

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f070000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int main=0x7f060000;
    }
}

```

```
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    ...
}
```

NOTA: Este fichero se genera automáticamente. Nunca debes editarlo.

Has de tener claro que los identificadores de la clase R son meros números que informan al gestor de recursos, que datos ha de cargar. Por lo tanto no se trata de verdaderos objetos, estos serán creados en tiempo de ejecución solo cuando sea necesario usarlos.



Ejercicio: El fichero *R.java*.

1. En **Android Studio**, el fichero *R.java* no es accesible desde el explorador de proyecto. No obstante, puedes acceder a él si pulsas con el botón derecho sobre *app* y seleccionas *Show in Explorer* (*o Show in Dolfin*). Desde esta carpeta abre el fichero:

app\build\generated\source\r\debug\nombre\del\paquete\R.java

Donde nombre\del\paquete has de reemplazarlo por el que corresponda al paquete de tu aplicación.

2. Comparalo con el fichero mostrado previamente. ¿Qué diferencias encuentras? (*RESPUESTA: cambia los valores numéricos en hexadecimal y contiene muchos más identificadores*).
3. Abre el fichero *MainActivity.java* y reemplaza *R.layout.activity_main* por el valor numérico al que corresponde en *R.java*.
4. Ejecuta de nuevo el proyecto. ¿Funciona? ¿Crees que sería adecuado dejar este valor numérico?
5. Aunque haya funcionado, este valor puede cambiar en un futuro. Por lo tanto para evitar problemas futuros vuelve a reemplazarlo por *R.layout.activity_main*.

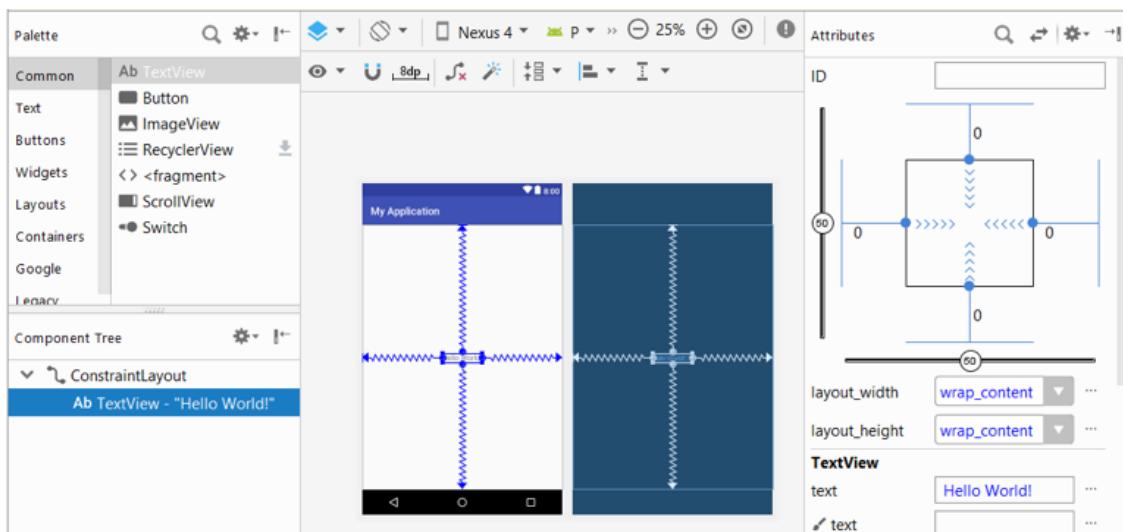
Diseño visual de vistas

Video Diseño visual de vistas

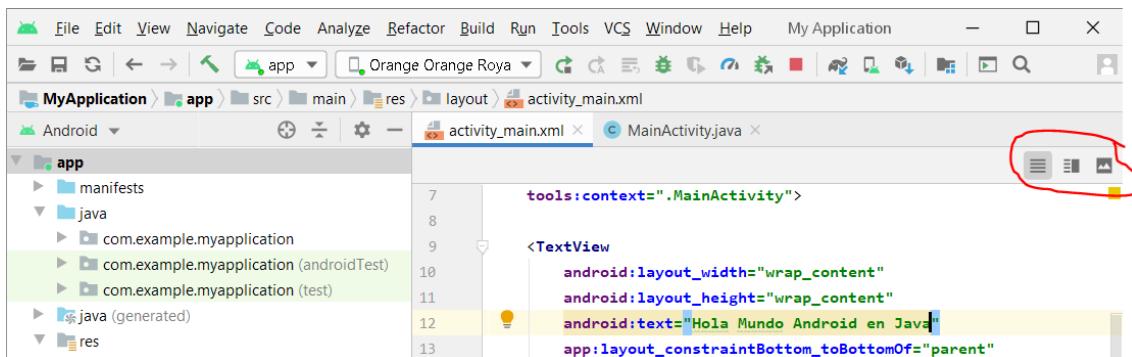
Objetivos:

Aprender a utilizar las herramientas integradas para la edición visual de Vistas y Layouts.

Veamos ahora como editar los *layouts* o ficheros de diseño en XML. En el explorador del proyecto abre el fichero *res/layout/activity_main.xml*. Verás que en la parte inferior de la ventana central aparecen dos lengüetas: *Design* y *Text*. Podrás usar dos tipos de diseño: editar directamente el código XML (segunda lengüeta) o realizar este diseño de forma visual (primera lengüeta). Veamos cómo se realizaría el diseño visual. La herramienta de edición de layouts se muestra a continuación:



(Para cambiar entre modo gráfico y texto, en la versión 3.6.3 de Android Studio son iconos a la derecha)



Nota: Si aparece un error con problemas de renderizado prueba otros niveles de API en el desplegable que aparece junto al pequeño robot verde, o con otro tema, en el botón con forma de círculo.

En la parte inferior izquierda encontramos el marco *Component Tree* con una lista con todos los elementos del *layout*. Este layout tiene solo dos vistas: un *ConstraintLayout* que contiene un *TextView*. En el marco central aparece una representación de cómo se verá el resultado y a

su derecha, con fondo azul, una representación con los nombres de cada vista y su tamaño. En la parte superior aparecen varios controles para representar este layout en diferentes configuraciones. Cuando diseñamos una vista en Android, hay que tener en cuenta que desconocemos el dispositivo final donde se visualizará y la configuración específica elegida por el usuario. Por esta razón, resulta importante que verifiques que el layout se ve de forma adecuada en cualquier configuración.

En la parte superior, de izquierda a derecha, encontramos los siguientes botones: El primero permite mostrar solo la visualización de diseño, solo la visualización esquemática o ambas. El botón muestra la orientación horizontal (*landscape*), vertical (*portrait*) y también podemos escoger el tipo de interfaz de usuario (coche, TV, reloj,...), escogemos el tipo de dispositivo (tamaño y resolución de la pantalla), con la versión de Android, con cómo se verá nuestra vista tras aplicar un tema y con Default(en-us) editar traducciones.

Nota: El siguiente vídeo corresponde a una versión anterior de la herramienta. Aunque cambian algunos iconos el funcionamiento continúa siendo similar. Para crear un nuevo layout pulsa con el botón derecho en el explorador de proyecto sobre app y selecciona la opción: New > Android resource file

Video [Diseño visual de Layouts: Visión general](#)

Ejercicio: Creación visual de Vistas

1. Crea un nuevo proyecto con los siguientes datos:

Phone and Tablet / Empty Activity

Name: Primeras Vistas

Minimum API level: API 19 Android 4.4 (KitKat)

Use androidx.* artifacts

Deja el resto de los parámetros con los valores por defecto.

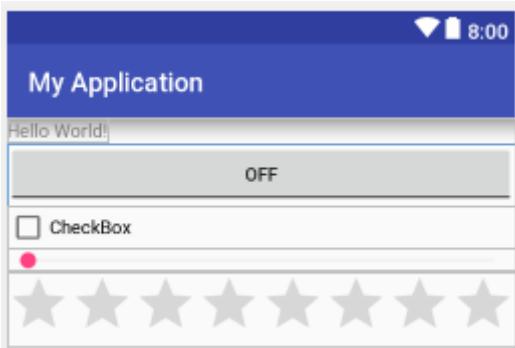
2. Abre el fichero `res/layout/activity_main.xml`.

3. Vamos a hacer que la raíz del *layout* se base en un *LinearLayout* vertical. Este tipo de *layout* es uno de los más sencillos de utilizar. Te permite representar las vistas una debajo de la otra. En el marco *Component Tree* pulsa con el botón derecho sobre *ConstraintLayout* y selecciona *Convert View...* Indica que quieres usar el layout, *LinearLayout*. El layout que ha añadido es de horizontal. En nuestro caso lo queremos de tipo vertical, para cambiarlo pulsa con el botón derecho sobre *LinearLayout*. y selecciona *LinearLayout/Convert orientation to vertical*. Todas las operaciones que hacemos en modo diseño visual (lengüeta *Design*) también las podemos hacer con el editor de texto. Para probarlo deshaz el trabajo anterior, pulsando el botón *Undo* `Ctrl+Z`). Selecciona la lengüeta *Text* y cambiar la etiqueta *ConstraintLayout* por *LinearLayout*. Añade el atributo *orientation* a *LinearLayout* para que la orientación sea vertical. Elimina los atributos innecesarios del *TextView* que utiliza con *ConstraintLayout*. El resultado ha de ser similar a:

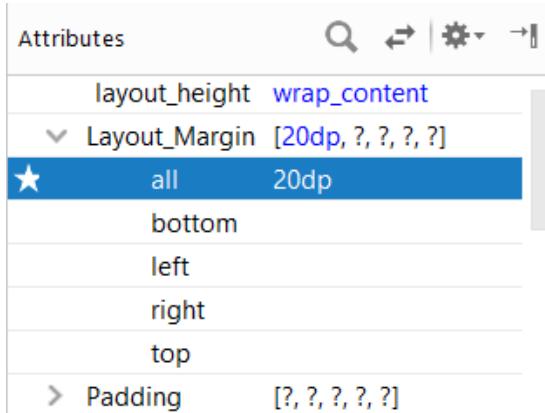
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</LinearLayout>
```

Regresa a la lengüeta *Design*.

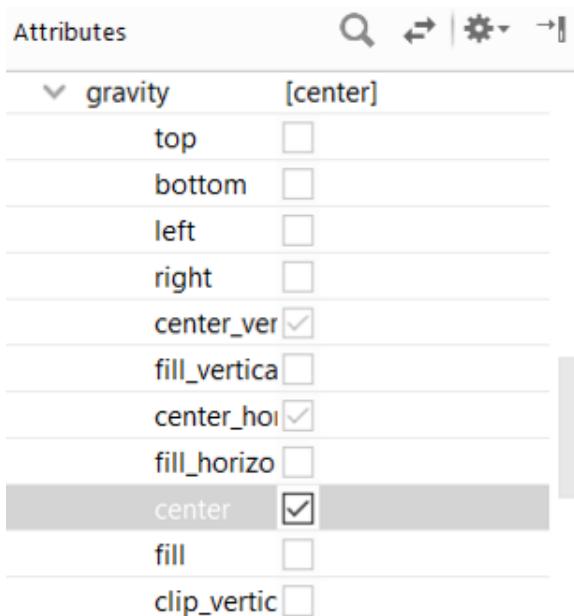
4. Desde la paleta de izquierda arrastra, al área de diseño, los siguientes elementos: ToggleButton, CheckBox,SeekBar y RatingBar.



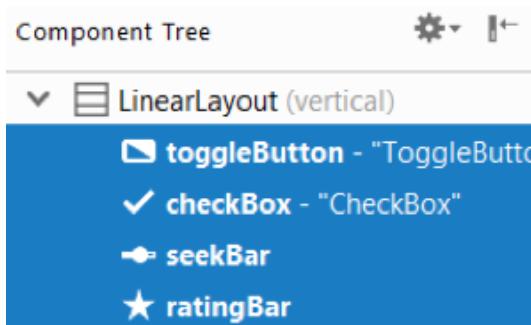
5. Selecciona la primera vista que estaba ya creada (TextView) y pulsa el botón <Supr> para eliminarla.
6. Selecciona la vista ToggleButton. Pulsa ahora (Set layout width to wrap_content). Conseguirás que la anchura del botón se ajuste a la anchura de su texto. Pulsa el botón (Set layout width to match_parent) para que el ancho del botón se ajuste a su contenedor. Observa en el marco Atributes como cambia la propiedad layout_width. Si el botón anterior no funciona cámbialo desde este marco. Deja el valor wrap_content.
7. Pulsa el botón (Convert orientation to horizontal), para conseguir que el LinearLayout donde están las diferentes vistas tenga una orientación horizontal. Comprobarás que no caben todos los elementos. Pulsa el botón (Convert orientation to vertical), para volver a una orientación vertical.
8. Con la vista ToggleButton seleccionada. Pulsa el botón (Set layout height to match_parent); Conseguirás que la altura del botón se ajuste al a altura de su contenedor. El problema es que el resto de elementos dejan de verse. Vuelve a pulsar este botón para regresar a la configuración anterior (También puedes pulsar Ctrl-z).
9. Selecciona la vista CheckBox. Ve al marco Atributes y en la parte inferior pulsa en All attributes. Busca la propiedad layout_margin en el campo all introduce "20dp". Se añadirá un margen arrededor de la vista.



10. Busca la propiedad gravity y selecciona center.



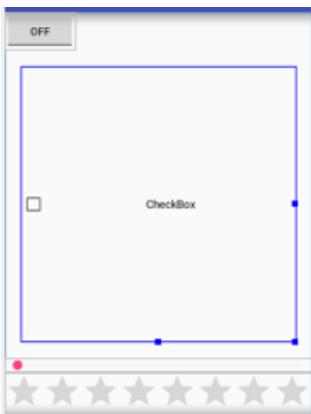
11. Observa que hay un espacio sin usar en la parte inferior del layout. Vamos a distribuir este espacio entre las vistas. Desde el marco Component Tree selecciona las cuatro vistas que has introducido dentro del LinearLayout. Para una selección múltiple mantén pulsada la tecla *Ctrl*.



12. Aparecerá un nuevo botón: (*Distribute Weights Evenly*). Púlsalo y la altura de las vistas se ajustará para que ocupen la totalidad del layout. Realmente, lo que hace es dividir el espacio sin usar de forma proporcional entre las vistas. Es equivalente a poner layout_weight = 1 y layout_height = 0dp para todas las vistas de este layout. Esta propiedad se modificará en un siguiente punto.

13. Selecciona las cuatro vistas y pulsa el botón  (*Clear All Weight*) para eliminar los pesos introducidos.

14. Selecciona la vista CheckBox. Asigna en el marco *Attributes*, *layout_weight = 1* y *layout_weight = 0dp* en esta vista. Observa como toda la altura restante es asignada a la vista seleccionada.



15. Para asignar un peso diferente a cada vista, repite los pasos anteriores donde asignábamos peso 1 a todas las vistas (botón). Pulsa la lengüeta *Text* y modifica manualmente el atributo *layout_weight* para que el ToggleButton tenga valor 2; CheckBox tenga valor 0.5; SeekBar valor 4 y RatingBar valor 1. Pulsa la lengüeta *Design*. Como puedes observar, estos pesos permiten repartir la altura sobrante entre las vistas.

16. Utiliza los siguientes botones  para ajustar el zum.

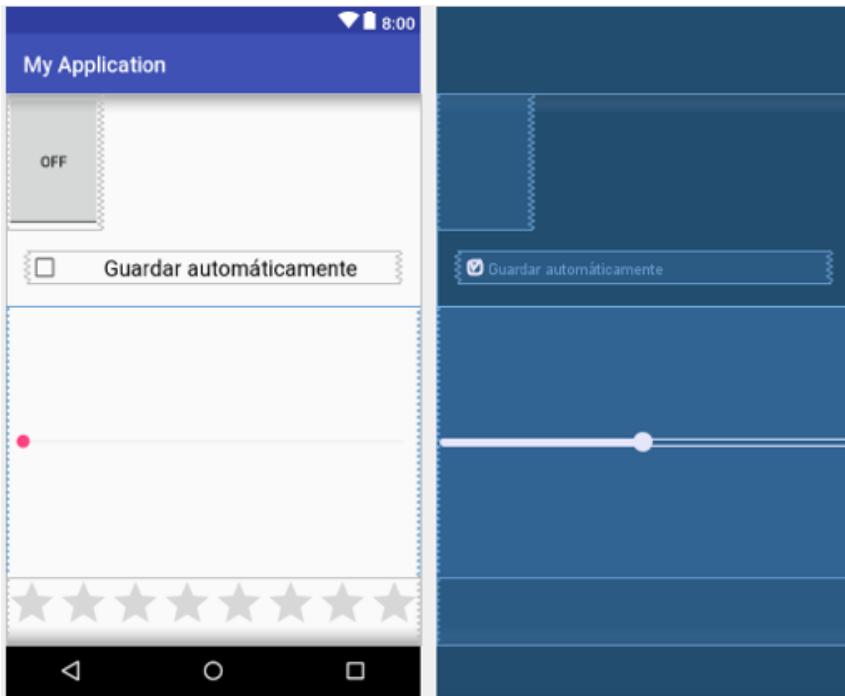
17. Utiliza los botones de la barra superior para observar cómo se representará el *layout* en diferentes situaciones y tipos de dispositivos:



18. Selecciona la vista CheckBox y observa las diferentes propiedades que podemos definir en el marco *Attributes*. Algunas ya han sido definidas por medio de la barra de botones. En concreto y siguiendo el mismo orden que en los botones hemos modificado: *Layout margin = 20dp*, *gravity = center* y *Layout weight = 0.5*.

19. Busca la propiedad *Text* y sustituye el valor “CheckBox” por “Guardar automáticamente” y *Text size* por “9pt”.

20. Pulsa el botón  para mostrar la visualización de diseño junto a la esquemática. A continuación se muestra el resultado obtenido:



21. Pulsa sobre la lengüeta *Text*. Pulsa las teclas *Ctrl-Alt-L* para que formatee adecuadamente el código XML. A continuación se muestra este código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="match_parent"
        android:layout_height="68dp"
        android:layout_weight="2"
        android:text="ToggleButton" />
    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="match_parent"
        android:layout_height="58dp"
        android:layout_margin="20dp"
        android:layout_marginStart="20dp"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="20dp"
        android:layout_marginEnd="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginBottom="20dp"
        android:layout_weight="0.5"
        android:gravity="center"
        android:text="CheckBox" />
    <SeekBar
        android:id="@+id/seekBar"
        android:layout_width="match_parent"
```

```
        android:layout_height="53dp"
        android:layout_weight="1" />
<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="387dp"
    android:layout_height="wrap_content"
    android:layout_weight="1" />
</LinearLayout>
```

- 22.** Ejecuta el proyecto para ver el resultado en el dispositivo.

Ejercicio: Vistas de entrada de texto

1. Añade en la parte superior del *Layout* anterior una vista de tipo entrada de texto *EditText*, de tipo normal (Plain Text). Lo encontrarás dentro del grupo *Text Fieds (EditText)*. Debajo de esta, una de tipo correo electrónico (*E-mail*) seguida de una de tipo palabra secreta (*Password*). Continua así con otros tipos de entradas de texto.

2. Ejecuta la aplicación.
3. Observa, como al introducir el texto de una entrada se mostrará un tipo de teclado diferente.

Los atributos de las vistas

video [Atributo de la clase View en Android](#)

video [Atributo de la clase TextView en Android](#)

Recursos adicionales: *Atributos de dimensión*

En muchas ocasiones tenemos que indicar la anchura o altura de una vista, un margen, el tamaño de un texto o unas coordenadas. Este tipo de atributos se conocen como atributos de dimensión. Dado que nuestra aplicación podrá ejecutarse en una gran variedad de dispositivos con resoluciones muy diversas, Android nos permite indicar estas dimensiones de varias formas. En la siguiente tabla se muestran las diferentes posibilidades:

px (píxeles): Estas dimensiones representan los píxeles en la pantalla.

mm (milímetros): Distancia real medida sobre la pantalla.

in (pulgadas): Distancia real medida sobre la pantalla.

pt (puntos): Equivale a 1/72 pulgadas.

dp (píxeles independientes de la densidad): Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densidad, se realizará el correspondiente ajuste. A diferencia de otras medidas como mm, in y pt este ajuste se hace de forma aproximada dado que no se utiliza la verdadera densidad gráfica, sino el grupo de densidad en que se ha clasificado el dispositivo (ldpi, mdpi, hdpi...). Esta medida presenta varias ventajas cuando se utilizan recursos gráficos en diferentes densidades. Por esta razón, Google insiste en que se utilice

siempre esta medida. **Desde un punto de vista práctico un dp equivale aproximadamente a 1/160 pulgadas. Y en dispositivos con densidad gráfica mdpi un dp es siempre un pixel.**

sp (píxeles escalados): Similar a dp, pero también se escala en función del tamaño de fuente que el usuario ha escogido en las preferencias. Indicado cuando se trabaja con fuentes.

Recursos adicionales: *Tipos de vista y sus atributos*

Consulta el siguiente [link](#), para conocer una lista con todos los descendientes de la clase View y sus atributos.

Los Layouts en Android

Video [Los Layouts en Android](#)

Objetivos:

Mostrar como podemos combinar varias vistas usando diferentes tipos de layouts.

Si queremos combinar varios elementos de tipo vista tendremos que utilizar un objeto de tipo *Layout*. Un *Layout* es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un *Layout* puede contener a otro *Layout* y que es un descendiente de la clase *View*.

La siguiente lista describe los *Layout* más utilizados en Android:

LinearLayout: Dispone los elementos en una fila o en una columna.

TableLayout: Distribuye los elementos de forma tabular.

RelativeLayout: Dispone los elementos en relación a otro o al padre.

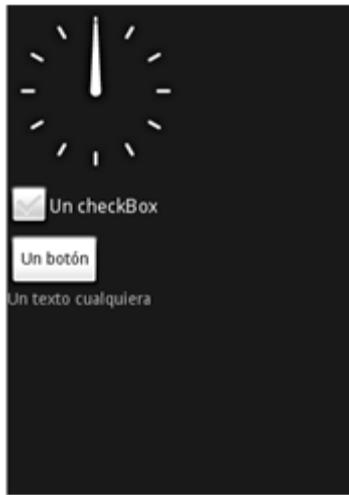
ConstraintLayout: Versión mejorada de RelativeLayout, que permite una edición visual desde el editor.

FrameLayout: Permite el cambio dinámico de los elementos que contiene.

AbsoluteLayout: Posiciona los elementos de forma absoluta.

Dado que un ejemplo vale más que mil palabras, pasemos a mostrar cada uno de estos *layouts* en acción:

LinearLayout es uno de los *Layout* más utilizado en la práctica. Distribuye los elementos uno detrás de otro, bien de forma horizontal o vertical.



```
<LinearLayout xmlns:android="http://...
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation ="vertical">
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkBox"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un texto cualquiera"/>
</LinearLayout>
```

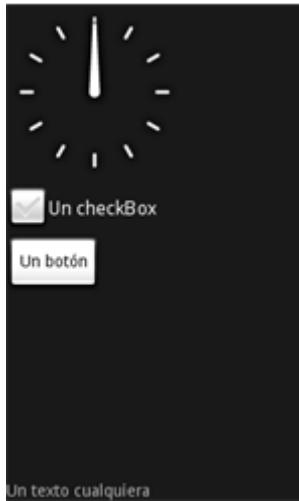
TableLayout distribuye los elementos de forma tabular. Se utiliza la etiqueta **<TableRow>** cada vez que queremos insertar una nueva línea.



```
<TableLayout xmlns:android="http://...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
```

```
<TableRow>
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkBox"/>
</TableRow>
<TableRow>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un texto cualquiera"/>
</TableRow>
</TableLayout>
```

RelativeLayout permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor e ir añadiendo nuevos elementos pegados a estos.



```
<RelativeLayout
    xmlns:android="http://schemas...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
    <AnalogClock
        android:id="@+id/AnalogClock01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"/>
    <CheckBox
        android:id="@+id/CheckBox01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/AnalogClock01"
        android:text="Un checkBox"/>
    <Button
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"
```

```
    android:layout_below="@+id/CheckBox01"/>
<TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="Un texto cualquiera"/>
</RelativeLayout>
```

ConstraintLayout Versión más flexible y eficiente de *RelativeLayout*.



```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
<AnalogClock
    android:id="@+id/AnalogClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
<CheckBox
    android:id="@+id/CheckBox01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"
    app:layout_constraintTop_toBottomOf=
        "@+id/AnalogClock01"
    app:layout_constraintTop_toTopOf="parent"/>
<Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    app:layout_constraintTop_toBottomOf=
        "@+id/CheckBox01"
    app:layout_constraintLeft_toLeftOf=
        "@+id/CheckBox01"/>
<TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:text="Un texto cualquiera"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

FrameLayout posiciona las vistas usando todo el contenedor, sin distribuirlas espacialmente. Este *Layout* suele utilizarse cuando queremos que varias vistas ocupen un mismo lugar. Podemos hacer que solo una sea visible, o superponerlas. Para modificar la visibilidad de un elemento utilizaremos la propiedad *visibility*.



```
<FrameLayout xmlns:android="http://schemas...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
    <AnalogClock
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un checkBox"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un botón"
        android:visibility="invisible"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Un texto cualquiera"
        android:visibility="invisible"/>
</FrameLayout>
```

AbsoluteLayout permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento. No es recomendable utilizar este tipo de *Layout*. La aplicación que estamos diseñando tiene que visualizarse correctamente en dispositivos con cualquier tamaño de pantalla. Para conseguir esto, no es una buena idea trabajar con coordenadas absolutas. De hecho, este tipo de *Layout* ha sido marcado como obsoleto.



```
<AbsoluteLayout xmlns:android="http://schemas.  
    android:layout_height="match_parent"  
    android:layout_width="match_parent">  
    <AnalogClock  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_x="50px"  
        android:layout_y="50px"/>  
    <CheckBox  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Un checkBox"  
        android:layout_x="150px"  
        android:layout_y="50px"/>  
    <Button  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Un botón"  
        android:layout_x="50px"  
        android:layout_y="250px"/>  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="Un texto cualquiera"  
        android:layout_x="150px"  
        android:layout_y="200px"/>  
</AbsoluteLayout>
```

Si tienes dudas sobre cuando usar cada layout usa la siguiente tabla:

- * **LinearLayout:** Diseños muy sencillos.
- * **RelativeLayout:** Nunca, hay una nueva alternativa
- * **ConstraintLayout:** Usar por defecto.
- * **FrameLayout:** Varias vistas superpuestas.
- * **AbsoluteLayout:** Nunca. Aunque está bien conocerlo por si acaso.

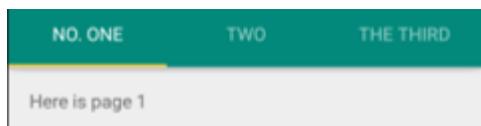
También podemos utilizar otras clases de *Layouts*, que son descritas a continuación:

ScrollView: Visualiza una vista en su interior; cuando estos no caben en pantalla se permite un deslizamiento vertical.

HorizontalScrollView: Visualiza una vista en su interior;; cuando esta no cabe en pantalla se permite un deslizamiento horizontal.



TabLayout , FragmentTabHost, TabLayout ó TabHost: Proporciona una lista de pestañas que pueden ser pulsadas por el usuario para seleccionar el contenido a visualizar. Se estudia al final del capítulo.



ListView: Visualiza una lista deslizable verticalmente de varios elementos. Su utilización es algo compleja. Se verá un ejemplo en el capítulo siguiente.



GridView: Visualiza una cuadrícula deslizable de varias filas y varias columnas.



RecyclerView: Versión actualizada que realiza las mismas funciones que ListView o GridView. Se verá en el siguiente capítulo.

ViewPager: Permite visualizar una serie de páginas, donde el usuario puede navegar arrastrando a derecha o izquierda. Cada página ha de ser almacenada en un fragment.

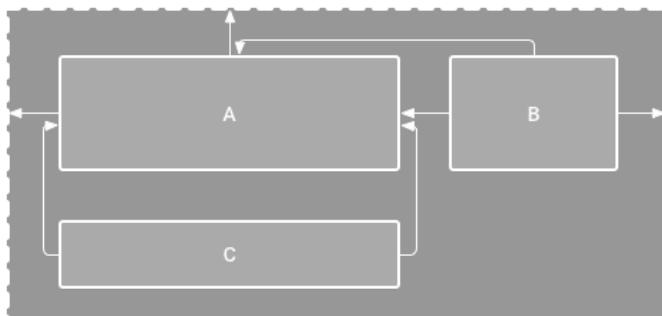
Uso de ConstraintLayout

Videos [Uso del constraintLayout1](#) [Uso del constraintLayout2](#)

Este nuevo layout ha sido añadido en una librería de compatibilidad, por lo que se nos anima a usarlo de forma predeterminada. Nos permite crear complejos diseños sin la necesidad de usar layout anidados. El hecho de realizar diseños donde un layout se introduce dentro de otro y así repetidas veces, ocasionaba problemas de memoria y eficiencia en dispositivos de pocas prestaciones.

Es muy parecido a RelativeLayout, pero más flexible y fácil de usar desde el editor visual de Android Studio (disponible desde la versión 2.3). De hecho, se recomienda crear tu layout con las herramientas drag-and-drop, en lugar de editar el fichero XML. El resto de layouts, son más fáciles de crear desde XML.

Las posiciones de las diferentes vistas dentro de este layout se definen usando constraint (en castellano restricciones). Un constraint puede definirse en relación al contenedor (parent), a otra vista o respecto a una línea de guía (guideline). Es necesario definir para cada vista al menos un constraint horizontal y uno vertical. Pero también podemos definir más de un constraint en el mismo eje. Veamos un ejemplo:



Observa como la vista A está posicionada con respecto al contenedor. La vista B está posicionada verticalmente con respecto a la vista A, aunque se ha definido un segundo constraint con respecto al lado derecho del contenedor. Veremos más adelante cómo se maneja esta circunstancia.

También podemos posicionar las vistas usando alineamientos. Observa como el borde superior de B está alineado con el borde superior de A. **La vista C define dos alineamientos horizontales simultáneos pero ninguno vertical, lo que ocasionará un error de compilación.** También podemos realizar alineamientos usando la línea base de texto y líneas de guía, como se muestra a continuación:



Ejercicio: Creación de un layout con ConstraintLayout

1. Abre el proyecto PrimerasVistas o crea uno nuevo.
2. En Gradle Scripts/Bulid.gradle (Module:app) ha de estar la dependencia:

```
dependencies
{
    ...
    implementation 'androidx.constraintlayout:constraintlayout:1.1.2'
}
```

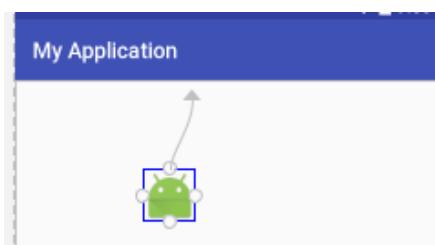
3. Abre el layout *activity_main.xml* creado en el ejercicio anterior. Pulsa con el botón derecho en *Component Tree* y seleccionamos la opción *Convert LinearLayout to ConstraintLayout*. Esta herramienta nos permite convertir nuestros viejos diseños que se basaban en *LinearLayout* y *RelativeLayout* en este nuevo tipo de layouts. Por desgracia no siempre funciona todo lo bien que

4. Crea un nuevo layout. Para ello, pulsa con el botón derecho sobre *app/res/layout* y selecciona *New/Layout resource file*. Como nombre introduce “constraint” y en *Root element*: *androidx.constraintlayout.widget.ConstraintLayout*
5. Vamos a desactivar la opción de *Autoconnect* de la barra de acciones del *ConstraintLayout*. Es el segundo ícono con forma de imán:



Tener activa esta opción es útil para diseñar más rápido los layouts. No obstante, a la hora de aprender a usar los constraint, es mejor ir haciéndolos de uno en uno.

8. Dentro del área *Palette*, selecciona *Common* y arrastra una vista de tipo *ImageView* al área de diseño. Se abrirá una ventana con diferentes recursos *Drawable*. Selecciona en *Project*, *ic_launcher*.
9. Para definir el primer constraint, pulsa sobre el punto de anclaje que aparece en la parte superior del *ImageView* y arrástralo hasta el borde superior del contenedor:

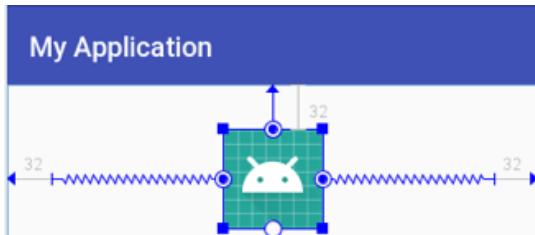


10. En la parte superior derecha nos aparece un editor visual para los constraint. Por defecto la distancia seleccionada ha sido 8dp. Pulsa sobre este número y cámbialo a 32dp:

NOTA:Según las recomendaciones de Material Design los márgenes y tamaños han de ser un múltiplo de 8dp.

11. Realiza la misma operación con el punto de anclaje izquierdo, arrastrándolo al borde izquierdo. Ya tenemos la restricción horizontal y vertical por lo que la vista está perfectamente ubicada en el layout.

12. Arrastra el punto de anclaje derecho al borde derecho, introduciendo una distancia de 32dp:



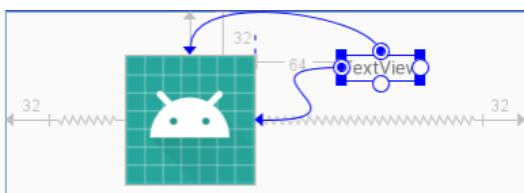
Observa como en este caso, al tener que cumplir simultáneamente dos constraint horizontales la imagen es centrada horizontalmente. Esto se representa con la línea en zigzag, representando un muelle, que estira de la vista desde los dos lados. El pequeño botón con una cruz roja que aparece, nos permite borrar todos los constraint de la vista.

13. Si en lugar de querer la imagen centrada la queremos en otra posición, podemos ir al editor de constraint y usar la barra deslizante (Horizontal Bias). Desplazarla a la posición 25.

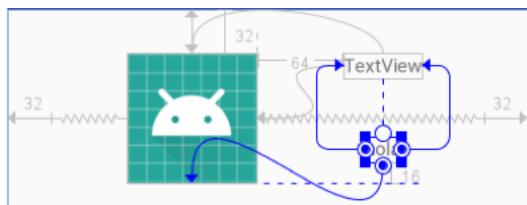
Observa en el área de trabajo, como la longitud del muelle de la izquierda es un 25%, frente al 75% del muelle de la derecha.

14. Seleccionando el ImageView en el área de trabajo, arrastra el cuadrado azul de la esquina inferior derecha, hasta aumentar su tamaño hasta 96x96 dp (múltiplos de 8). Otra alternativa es modificar los valores layout_width y layout_height en el área Properties.

15. Desde la paleta de widgets, añade un TextView a la derecha del ImageView. Arrastra el punto de anclaje de la izquierda hasta el punto de la derecha de la imagen y establece un margen de 64 dp. Arrastra el punto de anclaje superior del TextView hasta el punto superior de la imagen:

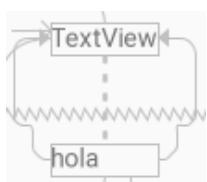


16. Añade un nuevo TextView bajo el anterior, con texto "hola". Introduce tres constraint, usando los puntos de anclaje inferior, izquierdo y derecho, tal y como se muestra en la siguiente figura:



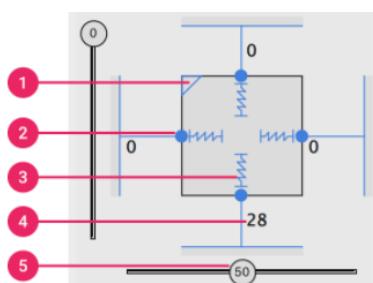
El margen inferior ha de ser 16dp y el izquierdo y derecho 0. De esta forma hemos centrado horizontalmente los dos TextView.

17. También podemos conseguir que el ancho del nuevo TextView coincida con el superior. Para ello selecciona la vista y en el campo layout_width introduce match_constraint ó 0dp. Con esto, hacemos que el ancho se calcule según las restricciones de los constraint. Quita los márgenes laterales para que los anchos de las dos vistas coincidan.



18. Haz que desde MainActivity se visualice este layout y ejecuta el proyecto en un dispositivo. Una vez familiarizados con los conceptos básicos de los constraint vamos a ver con más detalle las herramientas disponibles.

Una vez familiarizados con los conceptos básicos de los constraint vamos a ver con más detalle las herramientas disponibles. Veamos en editor de constraint:



(1) relación de tamaño: Puede establecer el tamaño de la vista en una proporción, por ejemplo 16:9, si al menos una de las dimensiones de la vista está configurada como "ajustar a constraint" (0dp). Para activar la relación de tamaño, haz clic donde señala el número 1.
(2) eliminar constraint: Se elimina la restricción para este punto de anclaje.
(3) establecer alto/ancho: Para cambiar la forma en la que se calcula las dimensiones de la vista, pulsa en este elemento. Existen tres posibilidades:

>>> ajustar a contenido: equivale al valor warp_content. (Ej. 1er TextView)

ajustar a constraint: equivale a poner 0dp. (Ej. 2º TextView)

tamaño fijo: equivale a poner un valor concreto de dp. (Ej. ImageView)

Aunque se representan 4 segmentos, realmente podemos cambiar 2, los horizontales para el ancho y los verticales al alto.

(4) establecer margen: Podemos cambiar los márgenes de la vista.

(5) Sesgo del constraint: Ajustamos como se reparte la dimensión sobrante.

También es importante repasar las acciones disponibles cuando trabajamos con ConstraintLayout:



Ocultar constraint: Elimina las marcas que muestra las restricciones y los márgenes existentes.



Autoconectar: Al añadir una nueva vista se establecen unos constraint con elementos cercanos de forma automática.

8dp **Definir márgenes por defecto**



Borrar todos los constraint: Se eliminan todas las restricciones del layout.



Crear automáticamente constraint: Dada una vista seleccionada, se establecen unos constraint con elementos cercanos de forma automática.



Empaquetar / expandir: Se agrupan o se separan los elementos.



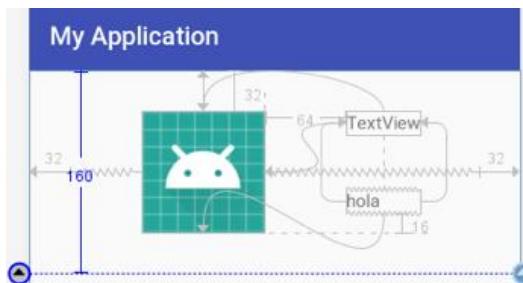
Alinear: Centra o justifica los elementos seleccionados.



Añadir línea de guía: Se crea una nueva línea de referencia.

Ejercicio: Líneas guía y cadenas en ConstraintLayout

1. Siguiendo con el Layout del ejercicio anterior. Pulsa en la acción Guideline , y selecciona Add Horizontal GuideLine. Aparecerá un circulo gris con un pequeño triángulo pegado al borde izquierdo, desde donde sale una línea guía horizontal. Arrástrala hacia abajo, hasta separarla una distancia de 160dp.



2. Esta guía nos permite dividir el layout en dos áreas, la superior donde ya hemos realizado una especie de cabecera y la inferior. A partir de ahora los elementos de la parte inferior los colocaremos en relación a esta línea guía.
3. Selecciona el ImageView, cópialo (Ctrl+C) y pégalo tres veces (Ctrl+V). Acabamos de hacer tres copias de la imagen, pero no son visibles al estar en la misma posición. En el área *Component Tree*, selecciona imageView2 y arrastrarlo bajo la línea guía, pegado a la izquierda. Coloca imageView3 bajo la línea guía, en el centro e imageView4, a la derecha de este
4. Selecciona imageView2, con el botón derecho y borra todos sus constraint seleccionando Clear Constraint of Selección. Repite esta operación para imageView3 y imageView4.
5. Las tres imágenes han de tener un constraint desde el punto de anclaje superior a la línea guía con un margen de 32dp. Desde el punto de anclaje izquierdo de imageView2, establece un constraint con el borde izquierdo y en las otras dos imágenes, al punto de anclaje derecho de la vista de la derecha. El punto de anclaje derecho de la tercera vista únelo al borde derecho. El resultado ha de ser el siguiente:
6. Para conseguir que estas tres vistas formen una cadena, seleccionalas y utiliza la acción Align / *Horizontalaly* o el botón derecho *Chains / Create Horizontal Chain*:

Observa cómo las vistas ahora están unidas por medio de un conector de cadena. Si abres la lengüeta Text, para estudiar el XML, puedes comprobar que para establecer la cadena se han añadido dos constraint, desde el punto de anclaje derecho de las dos primeras vista hacia la vista de su izquierda. Es decir, una restricción mutua, de A → B y de B → A. **Nota:** En la versión actual del editor visual, parece que establecer la cadena indicando estos constraint individualmente no parece posible. Hay que usar la acción Align / *Center Horizontalaly* o *Chains/Create Horizontal Chain*.

7. También es posible otras distribuciones de cadena. Podemos hacer que los márgenes se distribuyan solo entre las vistas o solo en los extremos izquierdo y derecho:

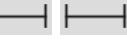
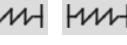


Si abres la lengüeta Text puedes comprobar que estas dos nuevas configuraciones de cadena se consiguen añadiendo en la primera vista el atributo:

app:layout_constraintHorizontal_chainStyle="spread_inside"
para la primera distribución y para la segunda:

app:layout_constraintHorizontal_chainStyle="packed"
Para la distribución del punto anterior el valor es "spread" o no indicar nada.

ta

8. Existe otra distribución en la que los márgenes desaparecen y se ajusta el ancho de las vistas hasta cubrir todo el espacio disponible. Selecciona la vista central y en el editor de constraint pulsa sobre el icono  hasta que aparezca . Recuerda que esta acción es equivalente a poner 0 en layout_width. El resultado se muestra a la izquierda:



Para conseguir el resultado de la derecha, hemos repetido la operación con las otras dos imágenes.

Si en lugar de repartir los anchos por igual, quieres otra configuración, puedes usar el atributo layout_constraintHorizontal_weight (funciona igual que layout_weight en un LinearLayout)

9. Ejecuta el proyecto en un dispositivo.

Al crear constraint, recuerde las siguientes reglas:

- Cada vista debe tener al menos dos restricciones: una horizontal y otra vertical.
- Sólo puede crear restricciones que comparten el mismo plano. Así, el plano vertical (los lados izquierdo y derecho) de una vista puede anclarse sólo a otro plano vertical.
- Cada manejador de restricción puede utilizarse para una sola restricción, pero puede crear varias restricciones (desde vistas diferentes) al mismo punto de anclaje.

Práctica: Uso de layouts

1. Utiliza un ConstraintLayout para realizar un diseño similar al siguiente:

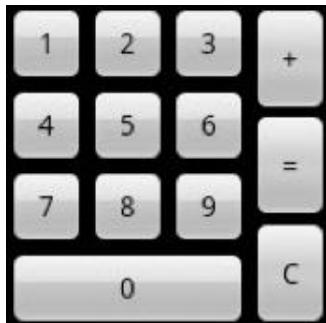
Escribe tu nombre

Nombre

2. Utiliza un TableLayout para realizar un diseño similar al siguiente:



3. Utiliza un LinearLayout horizontal que contenga en su interior otros LinearLayout para realizar un diseño similar al siguiente. Ha de acaparar toda la pantalla:

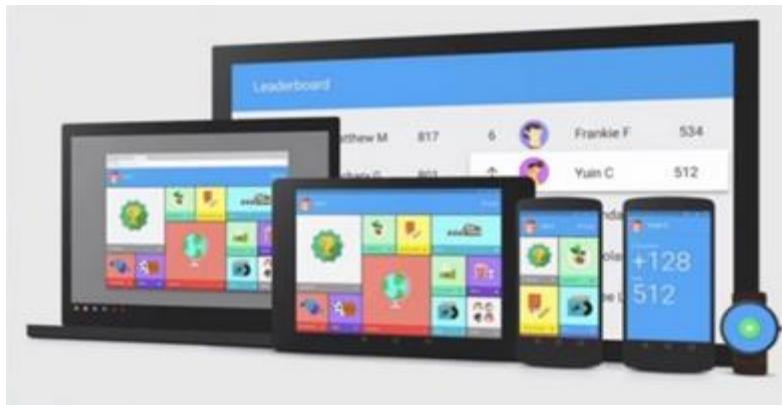


4. Visualiza el resultado obtenido en diferentes tamaños de pantalla. ¿Se visualiza correctamente?
5. Realiza el ejercicio de la calculadora usando un ConstraintLayout.

Material Design

Video [Material Design Curso de Material Design](#)

A partir de la versión 5.0 de Android (API 21), se introduce Material Design. Se trata de una guía para el diseño visual de las aplicaciones, que Google no quiere aplicar exclusivamente a dispositivos móviles, sino que pretende utilizar Material Design en todo tipo de contenidos digitales (páginas Web, aplicaciones para ordenadores, vídeos,...).



Se basa en diseños y colores planos. Uno de sus principios es dar peso o materialidad a los elementos del interfaz de usuario. Para ello va a tratar de darle volumen o profundidad utilizando sombras, capas y animaciones. Observa como los botones flotantes (ver botón con estrella en el siguiente ejercicio) se visualizan con una sombra para que parezcan que están en una capa superior y suelen visualizarse con animaciones. La idea es que parezcan que están construidos de material físico. Para más información puedes consultar la especificación de Material Design que se incluye en enlaces de interés.



Desde el punto de vista de la programación destacamos que se incorporan nuevas vistas: RecyclerView, Toolbar, FloatingActionButton, ...

Enlaces de interés:

Especificaciones de diseño de Material Design:

<http://www.google.com/design/spec/material-design/introduction.html>

Crear aplicaciones con Material Design:

<http://developer.android.com/intl/es/training/material/index.html>

Aplicación de ejemplo con diseño Material Design (Web / Android):

<https://polymer-topeka.appspot.com/>

<https://play.google.com/store/apps/details?id=com.chromecordova.Topeka>

Ejercicio paso a paso: Una aplicación basada en Material Design

Cuando creas un nuevo proyecto con Android Studio, si escoges una actividad del tipo Basic Activity, tendrá un diseño inicial basado en Material Design. Esta actividad, utilizará un tema que hereda de android:Theme.Material. Además, se incorporan varios widgets basados en este diseño, como: Toolbar o FloatingActionButton, y se incluirán las dependencias adecuadas para poder usar estos widwets:

1. Crea un nuevo proyecto con los siguientes datos:

Phone and Tablet / Basic Activity

Name: Material Design

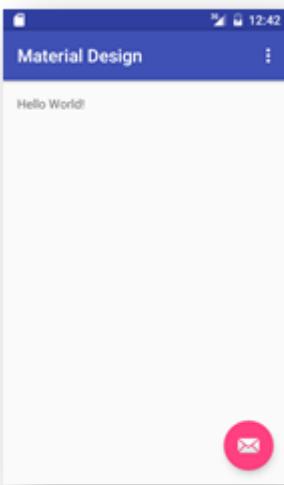
Package name: com.example.materialdesign

Language: Java ó Kotlin

Minimum API level: API 19 Android 4.4 (KitKat)

Al seleccionar una actividad de tipo Basic Activity se utilizará un diseño basado en Material Design. Dispondrá de una barra de acciones y un botón flotante.

2. Ejecuta el proyecto. El resultado ha de ser similar al siguiente



3. Pasemos a estudiar algunos elementos del proyecto. En el explorador del proyecto abre el fichero *Gradle Scripts > build.gradle (Module:app)*. Observa como en la sección dependencies se han incluido las siguientes librerías:

```
dependencies {
```

```
...
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.android.support:design:28.0.0'
...
}

Yo veo esto:
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.navigation:navigation-fragment:2.2.2'
    implementation 'androidx.navigation:navigation-ui:2.2.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

La primera es la librería de compatibilidad v7, que incorpora las clases más importantes como: AppCompatActivity, Toolbar o CardView. La segunda es para incluir el layout ConstraintLayout. La última es la librería de compatibilidad de diseño. Incorpora otras clases como: FloatingActionButton, AppBarLayout, TabLayout, NavigationView o Snackbar. Gracias al uso de estas librerías podremos utilizar estas clases con un nivel mínimo de API 7, a pesar de que la mayoría han sido introducidas en la versión 5.0 de Android.

Hay una diferencia entre estas librerías: Las clases definidas en la librería de compatibilidad appcompat-v7 son del API de Android. Cuando en una aplicación la versión mínima de API sea mayor o igual que 21 (v5.0) ya no tiene sentido usar esta librería. Por el contrario, las clases definidas en la librería de diseño son solo de esta librería. Has de usarla siempre que necesites una de sus clases.

4. Pasemos a estudiar los layouts creados en el proyecto. En el explorador del proyecto abre el fichero *app > res > layout > activity_main.xml*. En la vista *Design* se mostrará una previsualización del layout. Selecciona la vista *Text* pulsando sobre la lengüeta de la parte inferior.

5. Estudia la estructura de este layout. Observa como las etiquetas de las vistas son muy largas. Esto es debido a que no están definidas en el API de Android si no en una librería (en concreto a las design). La función de cada una de vistas se estudia en el siguiente apartado. Puede ser un buen momento para leerlo.

6. Abre el fichero *app > res > layout > content_main.xml*. Selecciona la vista *Text* pulsando sobre la lengüeta de la parte inferior. Se trata de un layout muy simple: un <ConstraintLayout> con un <TextView> dentro.

7. Abre ahora *app > java > com.example.mislugares > MainActivity.java*. En el método *onCreate()* se inicializan algunas de las vistas introducidas en el layout. Este código se explica en el siguiente apartado.

8. Si al ejecutar la aplicación pulsas en los tres puntos de la esquina superior derecha, se mostrará un menú con la opción *Settings*. Como se ha definido este menú y otras opciones sobre la barra de acciones se explicarán más adelante.

Vistas de Material Design: CoordinatorLayout, AppBarLayout, FloatingActionButtony Snackbar

Las animaciones son muy importantes en *Material Design*. Google quiere que los elementos del interfaz de usuario se muevan de forma coordinada, de forma que, al moverse un elemento, este puede desplazar a otro. Observa como al ejecutar el proyecto anterior, cuando pulsas el botón flotante se mueve hacia arriba mientras aparece un texto.

Para poder realizar esta animación, y alguna otra más, se utiliza CoordinatorLayout. Es un descendiente de LinearLayout que nos proporciona un nivel adicional de control sobre las vistas que contiene para controlar su comportamiento y permite crear interdependencias. Este contenedor se suele utilizar como raíz del layout.

Para entender cómo se utiliza abre el fichero *activity_main.xml*. Observa cómo tras eliminar los atributos la estructura resultante es la siguiente:



El CoordinatorLayout actúa de contenedor y va a permitir que los elementos que contiene puedan realizar animaciones coordinadas. Mediante el atributo `fitsSystemWindows` podemos conseguir modificar la barra de estado del sistema. En concreto aplicando a esta barra el color `colorPrimaryDark` (azul oscuro en el ejemplo anterior), para que combine con el color de la aplicación `colorPrimary` (azul más claro en el ejemplo anterior). El uso de colores en Material Design se explica en el siguiente apartado.

El AppBarLayout también es un LinearLayout vertical y ha sido diseñado para usarlo dentro de un CoordinatorLayout. A su vez es habitual que contenga un elemento de tipo Toolbar y opcionalmente pestañas (TabLayout). Su finalidad es contener los elementos de la barra de acciones de la aplicación (conocida según la nueva nomenclatura de Material Design como *AppBar*). Gracias al atributo `layout_scrollFlags` podemos conseguir que los elementos de dentro de esta barra puedan desplazarse hasta que sean ocultados.

Dentro de AppBarLayout tenemos un Toolbar. Como veremos en breve es un nuevo widget que tiene la misma función que un ActionBar.

El contenido del layout que ha de visualizar la aplicación se incluye en un fichero aparte `content_main`. De esta forma queda más estructurado y podemos crear contenidos específicos según el tipo de pantalla, versión, ... Observa como en el proyecto se ha creado un `content_main` que simplemente muestra el texto “Hello World!”.

Tras el `<include>` se ha insertado un `FloatingActionButton`. Se trata de un tipo de botón característico introducido en *Material Design* para las acciones principales.

Otro aspecto que no hay que olvidar es que ahora hemos de inicializar desde código algunos elementos. Esto se realiza en el método `onCreate()` de la actividad:

```
public class MainActivity extends AppCompatActivity
{
    @Override protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener()
        {
            @Override public void onClick(View view)
            {
                Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG).setAction("Action", null).show();
            } //onClick
        });
    } //onCreate
```

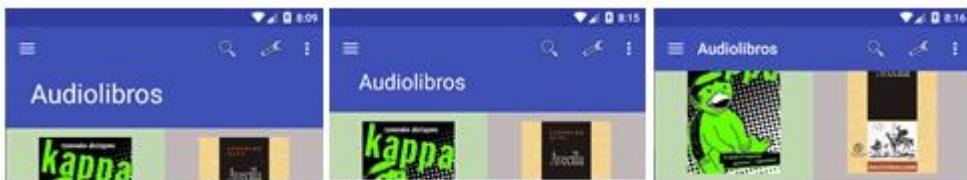
Este código resultará más sencillo de entender, tras realizar el apartado *Uso práctico de Vistas*. Las dos primeras líneas han sido explicadas en la unidad anterior. El método `setSupportActionBar(..)` es utilizado para indicar a la actividad la barra de acciones que ha de utilizar. Esta barra de acciones es almacenada en el objeto `toolbar`, que es creado en el layout XML. **En Java** para encontrar el objeto se utiliza `findViewById(R.id.toolbar)`. **En Kotlin** este proceso es realizado automáticamente. Lo único que tienes que hacer es asegurarte que se ha importado el paquete `kotlinx.android.synthetic.main.activity_main.*`.

Con respecto al botón flotante simplemente se carga en el objeto `fab` y se le asigna un escuchador `onClick`. Más adelante se describe el uso de escuchadores para personalizar la acción asociada. Actualmente, cuando se pulsa muestra un mensaje de texto utilizando un `Snackbar`. Se trata de una nueva forma de mostrar cuadros de dialogo, introducida en Material Desing, similar a `Toast` o `AlertDialog`. La diferencia es que se muestran en la parte inferior de la pantalla y que se visualizan con una animación de desplazamiento.

Ejercicio paso a paso: Creación de la aplicación Mis Lugares con barra extendida

En este curso vamos a crear una aplicación de ejemplo. Tendrá por nombre *Mis Lugares* y permitirá que los usuarios guarden información relevante sobre los sitios que suelen visitar (restaurantes, tiendas, etc.). En este apartado comenzaremos creando la aplicación Android que solo contendrá una actividad con una barra de acciones extendida.

La barra de acciones extendida es un patrón de diseño propuesto en *Material Design* que habrás observado en algunas aplicaciones actuales. Al entrar en la actividad la barra de acciones se muestra en modo extendido. Al desplazar el contenido la barra de acciones pasará al tamaño habitual.



Para conseguir este efecto has de utilizar el layout CollapsingToolbarLayout. En este ejercicio se describe cómo hacerlo:

1. Crea un nuevo proyecto con los siguientes datos:

Phone and Tablet / Scrolling Activity

Name: Mis Lugares

Package name: com.example.mislugares

Language: Java ó Kotlin

Minimum API level: API 19 Android 4.4 (KitKat)

De esta forma se creará una actividad similar a la que hemos creado en el ejercicio anterior. Igualmente dispondrá de una barra de acciones y un botón flotante. Pero ahora el contenido principal podrá desplazarse, a la vez que la barra de acciones reduce su tamaño.

2. En el navegador de proyecto pulsa con el botón derecho sobre la clase ScrollingActivity y selecciona *Refactor > Rename*. Introduce como nuevo nombre MainActivity. En la carpeta res/layout renombra el fichero activity_scrolling.xml por activity_main.xml. En la misma carpeta renombra content_scrolling.xml por content_main.xml. Finalmente en la carpeta res/menu renombra menu_scrolling.xml por menu_main.xml.

3. Ejecuta el proyecto y verifica su comportamiento:



4. Compara los layouts creados para *activity_main.xml* de este proyecto y el anterior. En la siguiente figura se muestra un resumen con las nuevas etiquetas y atributos XML que han sido añadidos. No ha de preocuparte si conoces el significado exacto de cada atributo.

Blank Activity	Scrolling Activity
<CoordinatorLayout...>	<CoordinatorLayout...>
<AppBarLayout ...>	<AppBarLayout ...>
<Toolbar.../>	layout_height="@dimen/app_bar_height"
	fitsSystemWindows="true">
	<CollapsingToolbarLayout.../>
	<Toolbar.../>
	</CollapsingToolbarLayout> </AppBarLayout>
</AppBarLayout>	
<include layout=	<include layout=
"@layout/content_main"/>	"@layout/content_main"/>
<FloatingActionButton.../>	<FloatingActionButton
	layout_anchor="@+id/app_bar"/>
</CoordinatorLayout>	</CoordinatorLayout>

Nota: De momento quédate con los conceptos principales. Si en un futuro quieres modificar este comportamiento básico, puedes consultar la información necesaria.

5. Compara los layouts creados para *content_main.xml*. Observa como ahora se utiliza el layout <NestedScrollView> para conseguir que el contenido sea desplazable. Además, el campo de texto es muy largo para conseguir que no quepa completo en pantalla.

Una aplicación de ejemplo: Mis Lugares

Objetivos:

Crear el proyecto de la aplicación Mis Lugares, que será desarrollada a lo largo del curso.

En el apartado anterior creamos la aplicación Mis Lugares que permitirá a los usuarios almacenar información sobre los sitios que han visitado.

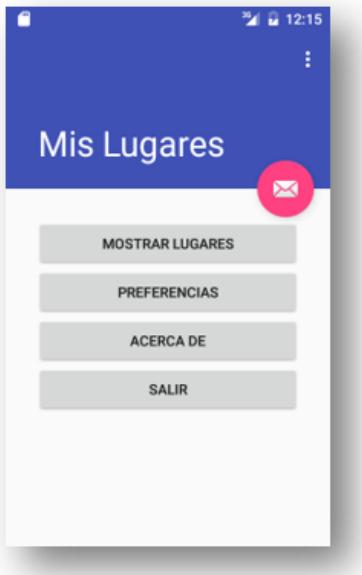
En este apartado crearemos un par de actividades para esta aplicación. La primera actividad contendrá simplemente cuatro botones. La segunda contendrá un formulario para dar de alta y editar un lugar. Esta primera versión de la aplicación no almacenará los datos introducidos.

Práctica: Creación de una primera actividad en Mis Lugares

En esta práctica crearemos una primera actividad que contendrá simplemente cuatro botones

1. Edita el fichero `res > layout > content_main.xml` y trata de crear una vista similar a la que ves a continuación. Has de dejar el `NestedScrollView` que ya tenías y reemplazar el `TextView` por un `ConstraintLayout` o `LinearLayout` que contenga cuatro `Button`.

Un `NestedScrollView` solo puede contener dentro un elemento, por lo que no puedes introducir directamente los cuatro botones. Usando un `layout` que los contenga se resuelve el problema.



Solución:

1. El fichero `content_main.xml` ha de ser similar al siguiente:

```
<android.support.v4.widget.NestedScrollView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

```
tools:showIn="@layout/activity_scrolling" android:layout_width="match_parent"
    android:layout_height="match_parent" tools:context=".MainActivity">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="30dp">
    <Button
        android:id="@+id/button01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/accion_mostrar" />
    <Button
        android:id="@+id/button02"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/accion_preferencias" />
    <Button
        android:id="@+id/button03"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/accion_acerca_de" />
    <Button
        android:id="@+id/button04"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/accion_salir" />
</LinearLayout>
</android.support.v4.widget.NestedScrollView>
```

2. El fichero `res > values > strings.xml` ha de tener el siguiente contenido:

```
<resources>
    <string name="accion_mostrar">Mostrar_Lugares</string>
    <string name="accion_preferencias">Preferencias</string>
    <string name="accion_acerca_de">Acerca de </string>
    <string name="accion_salir">Salir</string>
    <string name="app_name">Mis_Lugares</string>
    ...
</resources>
```

Práctica: Un formulario para introducir nuevos lugares.

El objetivo de esta práctica es crear un *layout* que permita introducir y editar lugares en la aplicación *Mis Lugares*.

- 1.** Crea un nuevo *layout* con nombre `edicion_lugar.xml`.
- 2.** Ha de parecerse al siguiente formulario. Puedes basarte en un *LinearLayout* o *ConstraintLayout* para distribuir los elementos. Pero es importante que este *layout*, se encuentre dentro de un *NestedScrollView* para que cuando el formulario no quepa en pantalla se pueda desplazar verticalmente.



3. Introduce a la derecha del TextView con texto “Tipo:” un Spinner con *id* tipo. Más adelante configuraremos esta vista para que muestre un desplegable con los tipos de lugares.

4. Las vistas EditText han de definir el atributo *id* con los valores: nombre, direccion, telefono, url y comentario. Utiliza también el atributo hint para dar indicaciones sobre el valor a introducir. Utiliza el atributo inputType para indicar qué tipo de entrada esperamos. De esta manera se mostrará un teclado adecuado (por ejemplo si introducimos un correo electrónico aparecerá la tecla @).

Nota: El atributo inputType admite los siguientes valores (en negrita los que has de utilizar en este ejercicio): none, text, textCapCharacters, textCapWords, textCapSentences, textAutoCorrect, textAutoComplete, textMultiLine, textImeMultiLine, textNoSuggestions, textUri, textEmailAddress, textEmailSubject, textShortMessage, textLongMessage, textPersonName, textPostalAddress, textPassword, textVisiblePassword, textWebEditText, textFilter, textPhonetic, textWebEmailAddress, textWebPassword, number, numberSigned, numberDecimal, numberPassword, phone, datetime, date y time.

5. Abre la clase MainActivity y en el método onCreate() reemplaza:

```
setContentView(R.layout.activity_main.edicion_lugar);
```

6. Comenta todas la líneas de este método que hay debajo usando /* ... */. Como ya no se crea el layout activity_main los id de vista a los que se accede ya no existen.

7. Ejecuta la aplicación y verifica como cambia el tipo de teclado en cada EditText

8. Deshaz el cambio realizado en el punto 5 y 6.

Solución:

```
<android.support.v4.widget.NestedScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">
<android.support.constraint.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/relativeLayout">
    <TextView
        android:id="@+id/t_nombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nombre:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="algo que identifique el lugar"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_nombre"
        android:layout_marginStart="8dp">
        <requestFocus/>
    </EditText>
    <TextView
        android:id="@+id/t_tipo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tipo:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nombre"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <Spinner
        android:id="@+id/tipo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="@+id/t_tipo"
        app:layout_constraintLeft_toRightOf="@+id/t_tipo"/>
    <TextView
        android:id="@+id/t_direccion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dirección:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_tipo"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/direccion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="dirección del lugar"
```

```

        android:inputType="textPostalAddress"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_direccion"
        android:layout_marginStart="8dp"/>
<TextView
        android:id="@+id/t_telefono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Telefono:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/direccion"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
<EditText
        android:id="@+id/telefono"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="teléfono para contactar"
        android:inputType="phone"
        app:layout_constraintLeft_toRightOf="@+id/t_telefono"
app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        app:layout_constraintStart_toEndOf="@+id/t_telefono"
        android:layout_marginStart="8dp"
        app:layout_constraintBaseline_toBaselineOf="@+id/t_telefono"/>
<TextView
        android:id="@+id/t_url"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Url:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/telefono"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
<EditText
        android:id="@+id/url"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="página web"
        android:inputType="textUri"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_url"
        android:layout_marginStart="8dp"/>
<TextView
        android:id="@+id/t_comentario"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comentario:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/url"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
<EditText
        android:id="@+id/comentario"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="introduce tus notas"
        android:inputType="textMultiLine"

```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/t_comentario"
    android:layout_marginStart="8dp"/>
</android.support.constraint.ConstraintLayout>
</android.support.v4.widget.NestedScrollView>
```

Uso de recursos alternativos en Android

video [Uso de recursos alternativos en Android](#)

Objetivos:

Demostrar por medio de varios ejemplos la utilidad de los recursos alternativos.

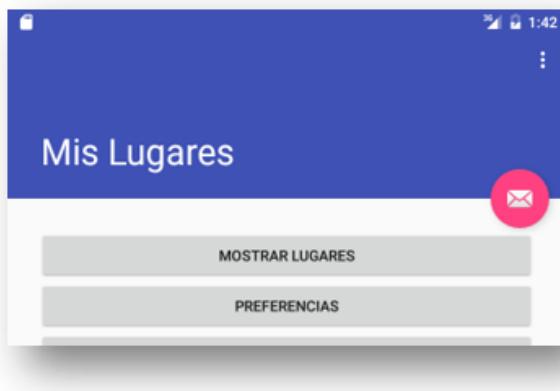
Una aplicación Android va a poder ser ejecutada en una gran variedad de dispositivos. El tamaño de pantalla, la resolución o el tipo de entradas puede variar mucho de un dispositivo a otro. Por otra parte, nuestra aplicación ha de estar preparada para diferentes modos de funcionamiento, como el modo “automóvil” o el modo “noche”, y para poder ejecutarse en diferentes idiomas.

A la hora de crear la interfaz de usuario, hemos de tener en cuenta todas estas circunstancias. Afortunadamente, la plataforma Android nos proporciona una herramienta de gran potencia para resolver este problema: el uso de los recursos alternativos.

Práctica: Recursos alternativos en Mis Lugares

1. Ejecuta la aplicación Mis Lugares.

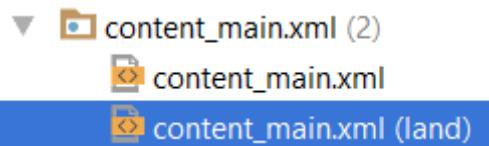
2. Los teléfonos móviles basados en Android permiten cambiar la configuración en apaisado y en vertical. Para conseguir este efecto con el emulador pulsa el botón . Si, usas un dispositivo con pantalla pequeña, puedes observar que el resultado de la vista que acabas de diseñar en vertical, no queda todo lo bien que deseáramos.



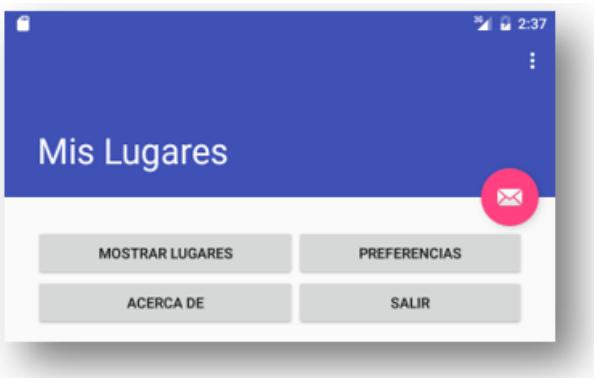
Para resolver este problema Android te permite diseñar una vista diferente para la configuración horizontal y otra para vertical.

3. Pulsa con el botón derecho sobre la carpeta `res/layout` y selecciona *New > Layout resource file*. Aparecerá una ventana donde has de rellenar en *File name*: `content_main`, en *Available qualifiers*: selecciona *Orientation* y pulsa en el botón `>>`. En el desplegable *Screen orientation*: selecciona *Landscape*. Pulsa en *OK*. Observa como ahora hay dos recursos para el

fichero *content_main.xml*. El primero es el recurso por defecto mientras que el segundo es el que se usará cuando el dispositivo esté en orientación *Landscape*.



4. Crea en el nuevo layout (*land*) una vista similar a la que ves a continuación: formada por un *TableLayout* con dos *Button* por columna.



5. Ejecuta de nuevo la aplicación y observa como la vista se ve de forma más adecuada en las dos orientaciones.

Solución:

Has de obtener un código XML similar al siguiente:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:padding="30dp"
    tools:context=".MainActivity" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/tituloAplicacion"
        android:gravity="center"
        android:textSize="25sp"
        android:layout_marginBottom="20dp"/>
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:stretchColumns="*">
        <TableRow>
            <Button android:id="@+id/button01"
                android:layout_height="wrap_content"
```

```
        android:layout_width="match_parent"
        android:text="@string/Arrancar"/>
    <Button android:id="@+id/button02"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/Configurar"/>
</TableRow>
<TableRow>
    <Button android:id="@+id/button03"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/Acercade"/>
    <Button android:id="@+id/button04"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/Salir"/>
</TableRow>
</TableLayout>
</LinearLayout>
```

NOTA: Para conseguir que en un *TableLayout*, las columnas se ajusten a todo el ancho de la tabla poner *stretchColumns="*"*. *stretchColumns="0"* significa que asigne el ancho sobrante a la primera columna. *stretchColumns="1"* significa que asigne el ancho sobrante a la segunda columna. *stretchColumns="*"* significa que asigne el ancho sobrante entre todas las columnas.

Android utiliza una lista de sufijos para expresar recursos alternativos. Estos sufijos pueden hacer referencia a la orientación del dispositivo, al lenguaje, la región, la densidad de píxeles, la resolución, el método de entrada, etc.

Por ejemplo, si queremos traducir nuestra aplicación al inglés, español y francés. Siendo el primer idioma el usado por defecto, crearíamos tres versiones del fichero *strings.xml* y lo guardaríamos en los siguientes tres directorios:

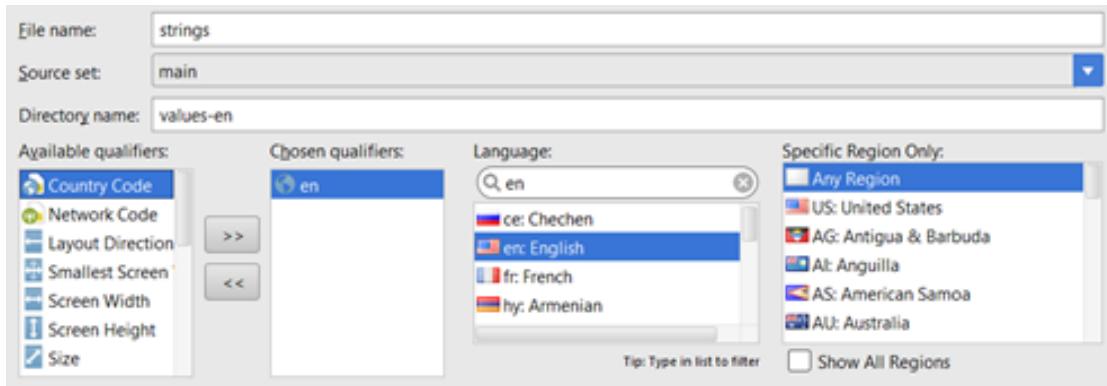
```
res/values/strings.xml
res/values-es/strings.xml
res/values-fr/strings.xml
```

Aunque internamente el SDK de Android utiliza la estructura de carpetas anterior, en Android Studio el explorador del proyecto muestra los recursos alternativos de la siguiente manera:

```
res/values/
    strings.xml(3)
    strings.xml
    strings.xml(es)
    strings.xml(fr)
```

Ejercicio: Traducción de Mis Lugares

1. Crea un nuevo recurso alternativo para *strings.xml* (en): Pulsa con el botón derecho en *res/values*, selecciona *New > Values resource file* e introduce *strings.xml* como nombre de fichero. Mueve el cualificador *Locale* desde el marco de la izquierda a la derecha, pulsando el botón *>>*. En *Language* selecciona *en: English*, en *Specific Region Only* deja el valor *Any Region Only* y pulsa *OK*.



NOTA: Observa cómo además del idioma también permite seleccionar la región. De esta forma podremos diferenciar entre inglés americano, británico, australiano, ...

2. Copia el contenido del recurso por defecto, *strings.xml*, al recurso para inglés, *strings.xml(en)*. Traduce los textos al inglés. No has de traducir los nombres de los identificadores de recursos (accion_mostrar, app_name, ...) estos han de ser igual en todos los idiomas.

3. Ejecuta la aplicación.

5. Vamos a cambiar la configuración de idioma de un dispositivo Android. Para ello, accede a *Ajustes del dispositivo (Settings)* y selecciona la opción *Mi dispositivo > Idioma e introducción*. Dentro de esta opción selecciona como idioma Español o Inglés.

NOTA: Observa que en otros idiomas permite seleccionar tanto el idioma como la región. Por desgracia, para el español solo permite dos regiones: España y Estados Unidos.

6. Observa cómo el texto aparece traducido al idioma seleccionado.

Otro ejemplo de utilización de recursos diferenciados lo podemos ver con el icono que se utiliza para lanzar la aplicación. Observa como, al crear una aplicación, este icono se crea en cinco carpetas *drawable* diferentes, para utilizar un ícono distinto según la densidad de píxeles del dispositivo:

```
res/mipmap-mdpi/ic_launcher.png
res/mipmap-hdpi/ic_launcher.png
res/mipmap-xhdpi/ic_launcher.png
res/mipmap-xxhdpi/ic_launcher.png
res/mipmap-xxxhdpi/ic_launcher.png
```

NOTA: En el siguiente capítulo se describe porque se actua de esta manera.

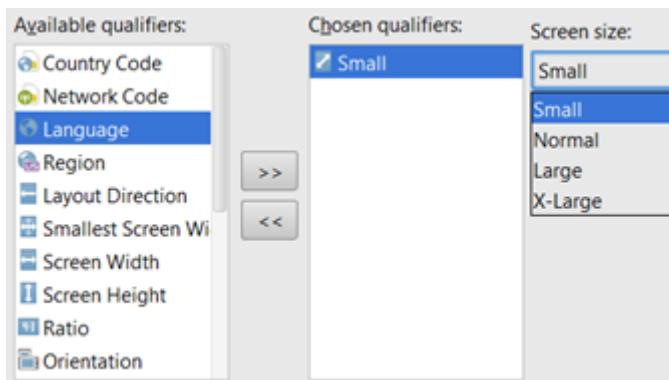
Resulta posible indicar varios sufijos concatenados; por ejemplo:

```
res/values-en-rUS/strings.xml
res/values-en-rUK/strings.xml
```

Pero cuidado, Android establece un orden a la hora de encadenar sufijos. Puedes encontrar una lista de estos sufijos en la *Referencia recursos alternativos* y en este enlace:

<http://developer.android.com/guide/topics/resources/providing-resources.html>

Para ver los sufijos disponibles, también puedes pulsar con el botón derecho sobre una carpeta de recursos y seleccionar *New > Android resource file*. Esta opción te permite crear un nuevo recurso y poner el sufijo deseado de forma y orden correctos.



Ejercicio paso a paso: Creando un Layout para tabletas en Mis Lugares

Si ejecutas la aplicación Mis Lugares en una tableta observarás que los botones son demasiado alargados:



Queremos que en este caso la apariencia sea similar a:



1. Crea un recurso alternativo a *res/values/dimens.xml*, que sea utilizado en pantallas de tamaño (size) *xlarge* (7-10,5 pulgadas) en orientación (orientation) *land* (apaisado). En este fichero define el siguiente valor:

1. `<resources>`
2. `<dimen name="margen_botones">150dp</dimen>`

3. </resources>
2. Añade el mismo valor al recurso por defecto, pero esta vez con 30dp.
3. Modifica los ficheros *res/layout/content_main.xml* y *content_main.xml (lan)*, reemplazando `android:padding="30dp"` por `android:padding="@dimen/margen_botones"`.
4. Verifica que la aplicación se visualiza correctamente en todos los tipos de pantalla, tanto en horizontal como en vertical.

Tipos de recursos

video [Tipos de recursos en Android](#)

Objetivos:

Enumerar los tipos de recursos que podemos utilizar en Android.

La definición de los recursos en Android es un aspecto muy importante en el diseño de una aplicación. Una de sus principales ventajas es que facilita a los diseñadores gráficos e introductores de contenido trabajar en paralelo con los programadores.

Añadir un recurso a nuestra aplicación es muy sencillo, no tenemos más que añadir un fichero dentro de una carpeta determinada de nuestro proyecto. Para cada uno de los recursos que añadamos el sistema crea, de forma automática, un id de recurso dentro de la clase R.

Tipos de recursos

Según la carpeta que utilicemos el recurso creado será de un tipo específico. Pasamos a enumerar las carpetas y tipos posibles:

Tabla: Tipos de recursos según carpeta en Android.

Veamos los tipos de recursos que encontramos dentro de la carpeta *values*:

Carpeta identificador	Descripción
res/drawable/ R.drawable	Ficheros en bitmap (.png, .jpg o .gif). Ficheros PNG en formato Nine-patch (.9.png). Ficheros XML con descriptores gráficos (ver clase Drawable)
res/mipmap/ R.mipmap	Ficheros en <i>bitmap</i> (.png, .jpg o .gif). Estos gráficos no son rescalados para adaptarlos a la densidad gráfica del dispositivo, sino que se buscará en las subcarpetas el gráfico con la densidad más parecida y se utilizará directamente.

res/layout/ R.layout	Ficheros XML con los Layouts usados en la aplicación.
res/menu/ R.menu	Ficheros XML con la definición de menús. Podemos asignar una actividad o una vista.
res/anim/ R.anim	Fichero XML que permiten definir una animaciones Tween también conocidas como animaciones de vista.
res/animator R.animator	Ficheros XML que permiten modificar las propiedades de un objeto a lo largo del tiempo. (Véase apartado "Animación de propiedades"). Solo desde la versión 3.0.
res/xml/ R.xml	Otros ficheros XML, como los ficheros de preferencias
res/raw/ R.raw	Ficheros que se encuentran en formato binario. Por ejemplo ficheros de audio o vídeo.
res/values/	Ficheros XML que definen un determinado valor para definir un color, estilo, cadena de caracteres, etc. Se describen en la siguiente tabla.

Fichero por defecto identificado r	Descripción

strings.xml R.string	<p>Identifica cadenas de caracteres</p> <pre><string name="saludo">¡Hola Mundo!</string></pre>
colors.xml R.color	<p>Un color definido en formato ARGB (alfa, rojo, verde y azul). Los valores se indican en hexadecimal en uno de los siguientes formatos: #RGB, #ARGB, #RRGGBB ó #AARRGGBB</p> <pre><color name="verde_opaco">#0f0</color> <color name="red_translucido">#80ff0000</color></pre>
dimensions.xml R.dimen	<p>Un número seguido de una unidad de medida.</p> <p>px - pixeles, mm - milímetros, in – pulgadas, pt – puntos (=1/72 pulgadas), dp – píxeles independientes de la densidad (=1/160 pulgadas), sp – igual que dp pero cambia según las preferencias de tamaño de fuente.</p> <pre><dimen name="alto">2.2mm</dimen> <dimen name="tamano_fuente">16sp</dimen></pre>
styles.xml R.style	<p>Definen una serie de atributos que pueden ser aplicados a una vista o a una actividad. Si se aplican a una actividad se conocen como temas.</p> <pre><style name="TextoGrande" parent="@style/Text"> <item name="android:textSize">20pt</item> <item name="android:textColor">#000080</item> </style></pre>
R.int	<p>Define un valor entero.</p> <pre><integer name="max_asteroides">5</integer></pre>
R.bool	<p>Define un valor booleano.</p> <pre><bool name="misiles_ilimitados">true</bool></pre>
R.id	<p>Define un recurso de id único. La forma habitual de asignar id a los recursos es utilizando el atributo id="@+id/nombre". Aunque en algunos casos puede ser interesante disponer de id previamente creado, para que los elementos así nombrados tengan una determinada función. Este tipo de id se utiliza en las vistas TabHost y ListView.</p> <pre><item type="id" name="button_ok"/> <item type="id" name="dialog_exit"/></pre>

R.array	Una serie ordenada de elementos. Pueden ser de strings, de enteros o de recursos (TypedArray) <string- array name="dias_semana"><item>lunes</item><item>martes</item></stri ng-array> <integer-array name="primos"> <item>2</item><item>3</item><item>5</item> </integer-array> <array name="asteroides"><item>@drawable/asteroide1</item><item>@dra wable/asteroide2</item> </array>
---------	--

Tabla: Tipos de recursos en carpeta values.

Aunque el sistema crea ficheros que aparecen en la columna de la izquierda de la tabla anterior y se recomienda definir los recursos de cadena dentro de *strings.xml*, hay que resaltar que no es más que una sugerencia de organización. Sería posible mezclar cualquier tipo de recurso de esta tabla dentro de un mismo fichero y poner a este fichero cualquier nombre.

Acceso a los recursos

Una vez definido un recurso este puede ser utilizado desde un fichero XML o desde Java. A continuación se muestra un ejemplo desde XML:

```
<ImageView
    android:layout_height="@dimen/alto"
    android:layout_width="match_parent"
    android:background="@drawable/asteroide"
    android:text="@string/saludo"
    android:text_color="@color/verde_opaco"/>
```

Para acceder a un recurso definido en los ejemplos anteriores desde Java usaremos el siguiente código:

```
Resources res = getResources();
Drawable drawable = ContextCompat.getDrawable(R.drawable.asteroide);
String saludo = res.getString(R.string.saludo);
int color = ContextCompat.getColor(R.color.verde_opaco);
float tamanoFuente = res.getDimension(R.dimen.tamano_fuente);
int maxAsteroides = res.getInteger(R.integer.max_asteroides);
boolean ilimitados = res.getBoolean(R.bool.misiles_ilimitados);
String[] diasSemana = res.getStringArray(R.array.dias_semana);
int[] primos = res.getIntArray(R.array.primos);
TypedArray asteroides = res.obtainTypedArray(R.array.asteroides);
Drawable asteroide1 = asteroides.getDrawable(0);
val drawable = ContextCompat.getDrawable(R.drawable.asteroide)
val saludo = resources.getString(R.string.saludo)
val color = ContextCompat.getColor(R.color.verde_opaco)
val tamanoFuente = resources.getDimension(R.dimen.tamano_fuente)
val maxAsteroides = resources.getInteger(R.integer.max_asteroides)
val ilimitados = resources.getBoolean(R.bool.misiles_ilimitados)
val diassSemana = resources.getStringArray(R.array.dias_semana)
val primos = resources.getIntArray(R.array.primos)
val asteroides = resources.obtainTypedArray(R.array.asteroides)
val asteroide1 = asteroides.getDrawable(0)
```

Recursos del sistema

video Recursos del sistema en Android

Objetivos:

Mostrar cómo podemos utilizar recursos del sistema en nuestras aplicaciones.

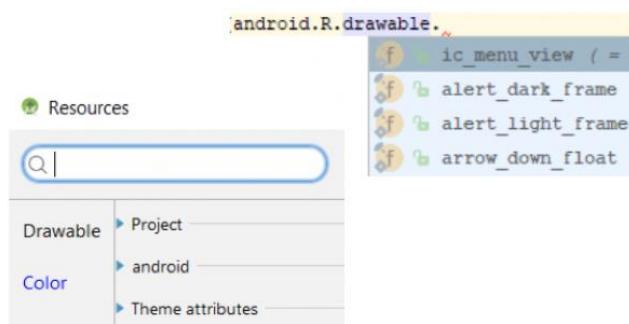
Además de los recursos que podamos añadir a nuestra aplicación, también podemos utilizar una serie de recursos que han sido incluidos en el sistema.

Usar recursos del sistema tiene muchas ventajas. No consumen memoria en nuestra aplicación, al estar ya incorporados al sistema. Además los usuarios están familiarizados con ellos. Por ejemplo, si utilizamos el recurso android.R.drawable.ic_menu_edit se mostrará al usuario el icono . Muy posiblemente el usuario ya está familiarizado con este ícono y lo asocie a la acción de editar. Otra ventaja es que los recursos del sistema se adaptan a las diferentes versiones de Android. Si se utiliza el tema android.R.style.Theme_Panel este es bastante diferente en cada una de las versiones, pero seguro que estará en consonancia con el resto de estilos para esta versión. Lo mismo ocurre con el ícono anterior. Este ícono es diferente en algunas versiones, pero al usar un recurso del sistema nos aseguramos que se mostrará el adecuado a la versión del usuario. Finalmente, estos recursos se adaptan siempre a las configuraciones locales. Si yo utilizo el recurso android.R.string.cancel este será “Cancelar”, “Cancel”, “取消”,... según el idioma escogido por el usuario.

Para acceder a los recursos del sistema desde código usaremos la clase android.R. Se utiliza la misma estructura de jerárquica de clases. Por ejemplo android.R.drawable.ic_menu_edit. Para acceder desde XML utiliza la sintaxis habitual pero comenzando con @android:. Por ejemplo @android:drawable/ic_menu_edit.

Para buscar recursos del sistema tienes varias alternativas:

- *Usa la opción de autocompletar de Android Studio.
- *En el editor de layouts se incluye un buscador de recursos.
- *Usa la aplicación android.R para explorar los recursos del sistema.



Estilos y Temas en Android

video Estilos y Temas en Android

Objetivos:

Mostrar cómo podemos aplicar estilos a una vista.

Mostrar cómo podemos aplicar temas a una actividad.

Describir cómo podemos crear nuevos estilos y temas a partir de otros ya creados

Si tienes experiencia con el diseño de páginas web, habrás advertido grandes similitudes entre HTML y el diseño de layouts. En los dos casos se utiliza un lenguaje de marcado y se trata de crear diseños independientes del tamaño de la pantalla donde se visualizarán. En el diseño web resultan clave las hojas de estilo en cascada CSS, que permiten crear un patrón de diseño y aplicarlo a varias páginas. Cuando diseñes los layouts de tu aplicación, vas a poder utilizar unas herramientas similares conocidas como estilos y temas. Te permitirán crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación. Estas herramientas te ahorrarán mucho trabajo y te permitirán conseguir un diseño homogéneo en toda tu aplicación.

Los estilos

Un estilo es una colección de propiedades que definen el formato y apariencia que tendrá una vista. Podemos especificar cosas como tamaño, márgenes, color, fuentes, etc. Un estilo se define en ficheros XML, diferente al fichero XML Layout que lo utiliza.

Veamos un ejemplo. El siguiente código:

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="Un texto" />
```

Es equivalente a escribir:

```
<TextView  
    style="@style/MiEstilo"  
    android:text="Un texto" />
```

Habiendo creado en el fichero res/values/styles.xml con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <style name="MiEstilo"  
        parent="@android:style/TextAppearance.Medium">  
        <item name="android:layout_width">match_parent</item>  
        <item name="android:layout_height">wrap_content</item>  
        <item name="android:textColor">#00FF00</item>  
        <item name="android:typeface">monospace</item>  
    </style>  
</resources>
```

Observa como un estilo puede heredar todas las propiedades de un parent (parámetro parent) y a partir de estas propiedades realizar modificaciones.

Heredar de un estilo propio

Si vas a heredar de un estilo definido por ti no es necesario utilizar el atributo parent. Por el contrario, puedes utilizar el mismo nombre de un estilo ya creado y completar el nombre con un punto más un sufijo. Por ejemplo:

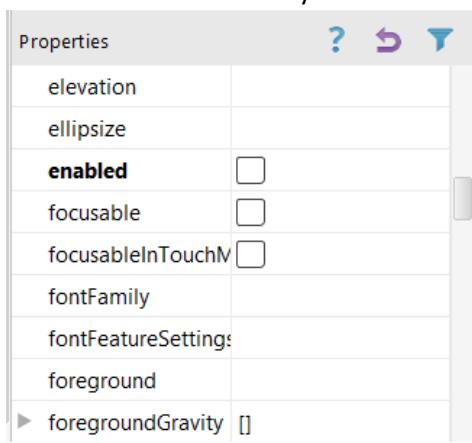
```
<style name="MiEstilo.grande">
    <item name="android:textSize">18pt</item>
</style>
```

Crearía un nuevo estilo que sería igual a MiEstilo más la nueva propiedad indicada. A su vez puedes definir otro estilo a partir de este:

```
<style name="MiEstilo.grande.negrita">
    <item name="android:textStyle">bold</item>
</style>
```

Práctica: Creando un estilo

1. Abre el Mis Lugares .
2. Crea un nuevo estilo con nombre EstiloBoton. Para ver las propiedades que puedes modificar te recomendamos que consultes la "Referencia de la clase View". Para un botón puedes definir los atributos de View, TextView y Button. Otra alternativa consiste en seleccionar un botón en el editor visual de vistas y en la ventana Properties buscar las propiedades disponibles:



3. Apícalo al primer botón del layout.
4. Crea un nuevo estilo con nombre EstiloBoton.Alternativo. Este ha de modificar alguno de los atributos anteriores y añadir otros, como padding.
5. Apícalo al segundo botón del Layout.
6. Visualiza el resultado.

Los temas

Un tema es un estilo aplicado a toda una actividad o aplicación, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible. Por ejemplo, CodeFont solo afectará al texto.

Para aplicar un tema a toda una aplicación edita el fichero AndroidManifest.xml y añade el parámetro android:theme en la etiqueta <application>:

```
<application android:theme="@style/MiTema">
```

También puedes aplicar un tema a una actividad en concreto:

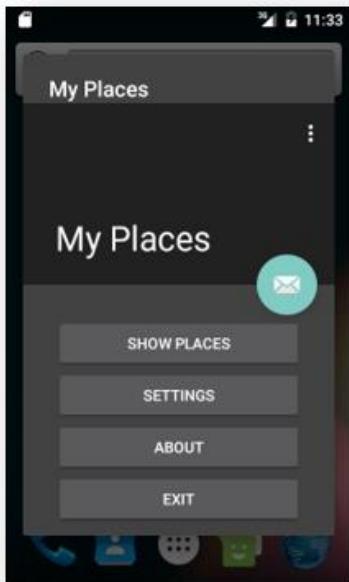
```
<activity android:theme="@style/MiTema">
```

Además de crear tus propios temas vas a poder utilizar algunos disponibles en el sistema. Puedes encontrar una lista de todos los estilos y temas disponibles en Android en:<http://developer.android.com/reference/android/R.style.html>

Ejercicio paso a paso: Aplicando un tema del sistema

1. Abre el proyecto Mis Lugares

2. Aplica a la actividad principal el tema @style/Theme.AppCompat.Dialog tal y como se acaba de mostrar.
3. Visualiza el resultado. Este tema es utilizado en cuadros de dialogo. No parece muy adecuado para nuestra actividad.



4. Deshaz el cambio realizado en este ejercicio.

Práctica: Modificando el tema por defecto de la aplicación

1. Abre el Mis Lugares.
2. Abre el fichero res/values/styles.xml (recurso por defecto).
3. Observa como se define el estilo AppTheme que será usado como estilo por defecto en la aplicación. Hereda de Theme.AppCompat.Light.DarkActionBar y solo define los colores principales usados en la aplicación. Añade las líneas subrayadas para personalizar un par de aspectos:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="android:typeface">serif</item>
    <item name="android:textColor">#0000FF</item>
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

4. Ejecuta la aplicación para visualizar el resultado.
5. Modifica otros atributos y comprueba el resultado.

Uso práctico de Vistas

Objetivos:

Mostrar como podemos interaccionar con los elementos de una vista desde el código Java.

En este apartado vamos a aprender a usar varios tipos de vistas y layouts desde un punto de vista práctico. También empezaremos a escribir código que será ejecutado cuando ocurran ciertos eventos:

Ejercicio: Un botón con gráficos personalizados

1. Crea un nuevo proyecto con nombre: Mas Vistas. En la tercera ventana selecciona *Empty Activity*. Puedes dejar el resto de parámetros con los valores por defecto.
2. Crea el fichero boton.xml en la carpeta res/drawable/. Para ello puedes utilizar el menú *File > New > Drawable Resource File*. Introduce en el campo *File name: <>boton>* Reemplaza el código por el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/boton_pulsado"
          android:state_pressed="true" />
    <item android:drawable="@drawable/boton_con_foco"
          android:state_focused="true" />
    <item android:drawable="@drawable/boton_normal" />
</selector>
```

Este XML define un recurso único gráfico (drawable) que cambiará en función del estado del botón. El primer ítem define la imagen usada cuando se pulsa el botón, el segundo ítem define la imagen usada cuando el botón tiene el foco (cuando el botón está seleccionado con la rueda de desplazamiento o las teclas de dirección) y el tercero, la imagen en estado normal. Los gráficos, y en concreto los *drawables*, se estudiarán en el capítulo 4.

NOTA: *El orden de los elementos <item> es importante. Cuando se va a dibujar se recorren los ítems en orden hasta que se cumpla una condición. Debido a que "boton_normal" es el último, sólo se aplica cuando las condiciones state_pressed y state_focused no se cumplen.*

3. Descarga de <http://www.androidcurso.com/index.php/119> las tres imágenes que aparecen a continuación. Para bajar cada imagen, pulsa sobre los nombres. Guardalos con el nombre de fichero se indica a continuación:

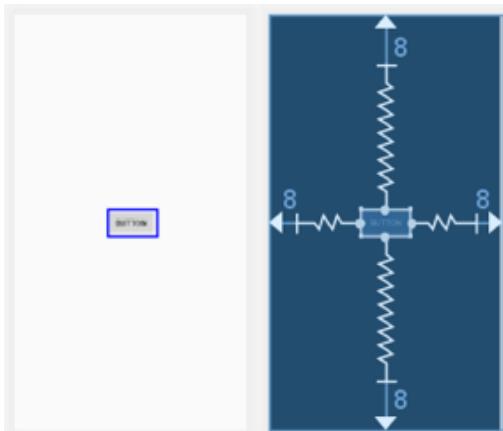


[boton_normal.jpg](#)

[boton_con_foco.jpg](#)

[boton_pulsado.jpg](#)

4. Selecciona los tres ficheros y cópialos en el portapapeles (*Ctrl-C*), selecciona la carpeta *res/drawable/* del proyecto y pega los ficheros (*Ctrl-V*). Te preguntará si quieres copiarlos a la carpeta de recursos por defecto a alguna de recursos alternativos. Selecciona la primera opción.
5. Abre el fichero *res/layout/activity_main.xml*.
6. En la ventana *Component Tree*, elimina el *TextView* que encontrarás dentro del *ConstraintLayout*.
7. Selecciona el *ConstraintLayout*. En la ventana de *Attributes* busca el atributo *Background*. Pulsal en el ícono de la herramienta y selecciona el recurso *Color/android/white*.
8. Arrastra una vista de tipo *Button* dentro del *ConstraintLayout*.
9. Sitúa el botón en el centro. Para ello selecciona cada uno de sus puntos de anclaje arrastrando hasta el borde al que mira. El resultado ha de ser:



10. Selecciona el atributo *Background* y pulsa en el botón selector de recurso (con puntos suspensivos). Selecciona *Drawable/boton*.
11. Modifica el atributo *Text* para que no tenga ningún valor.
12. Introduce en el atributo *onClick* el valor *sePulsa*.

A continuación se muestra el código resultante para *activity_main.xml*:

```
<android.support.constraint.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity"  
    android:background="@android:color/white">  
    <Button android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:id="@+id/button"  
        android:background="@drawable/boton"  
        android:onClick="sePulsa"  
        app:layout_constraintStart_toStartOf="parent"  
        android:layout_marginStart="8dp"  
        android:layout_marginTop="8dp"
```

```
    app:layout_constraintTop_toTopOf="parent"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    android:layout_marginEnd="8dp"
    app:layout_constraintEnd_toEndOf="parent"/>
</android.support.constraint.ConstraintLayout>
```

- 13.** Abre el fichero *MainActivity.java* e introduce al final, antes de la última llave, el código:

```
public void sePulsa(View view){
    Toast.makeText(this, "Pulsado", Toast.LENGTH_SHORT).show();
}
```

NOTA: Pulta **Alt-Intro** en **Android Studio** para que se añadan automáticamente los paquetes que faltan en la sección import.

El método anterior se ejecutará cuando se pulse el botón. A este tipo de métodos se los conoce como escuchadores de eventos (*listeners*). Este método se limita a lanzar un *toast*, es decir, un aviso que permanece cierto tiempo sobre la pantalla y luego desaparece. Los tres parámetros son: el contexto utilizado, que coincide con la actividad, el texto a mostrar y el tiempo que permanecerá este texto. Los conceptos de *actividad* y *contexto* se desarrollarán en el siguiente capítulo.

- 14.** Ejecuta el proyecto y verifica el resultado.

Acceder y modificar las propiedades de las vistas por código

Ejercicio: Acceder y modificar las propiedades de las vistas por código

1. Abre el *Layout activity_main.xml* creado en el ejercicio anterior.
2. En la paleta de vistas, dentro de *Text*, busca *Number (Decimal)* y arrástralolo arriba del botón rojo.
3. Modifica algunos atributos de esta vista: *hint* = “Introduce un número”, *ID* = “entrada”
4. En la paleta de vistas, dentro de *Widgets*, busca *Button* y arrástralolo arriba del botón rojo.
5. Modifica algunos atributos de esta vista: Haz que su anchura ocupe toda la pantalla, que su texto sea “0” y que su *id* sea “boton0”.
6. En la paleta de vistas, dentro de *Widgets*, busca *TextView* y arrástralolo debajo del botón rojo.
7. Modifica algunos atributos de esta vista: *TextColor* = #0000FF, *Text* = “”, *Hint* = “Resultado”, *id*= “salida”.
8. Ajusta las restricciones de las vistas introducidas.

9. Abre el fichero *MainActivity*. **En Java** vamos a añadir dos nuevas propiedades a la clase. Para ello copia el siguiente código al principio de la clase (antes del método *onCreate()*):

```
private EditText entrada;
private TextView salida;
```

NOTA: Recuerda pulsar **Alt-Intro** para que se añadan los paquetes de las dos nuevas clases utilizadas.

10. **En Java** copia al final del método *onCreate()* las siguientes dos líneas:

```
entrada = findViewById(R.id.entrada);
salida = findViewById(R.id.salida);
```

Como se explicó al principio del capítulo, las diferentes vistas definidas en *activity_main.xml*, son creadas como objetos Java cuando se ejecuta `setContentView(R.layout.main)`. Si queremos manipular algunos de estos objetos hemos de declarar las variables (paso 9) y asignarles la referencia al objeto correspondiente (paso 10). Para ello, hay que introducir el atributo `id` en XML y utilizar el método `findViewById(R.id.valor_en_atributo_id)`. Este método devuelve un objeto de la clase `View`.

11. Introduce en el atributo `onClick` del botón con id `boton0` el valor “`sePulsa0`”. De esta manera, cuando se pulse sobre el botón se ejecutará el método `sePulsa0()`. Según la jerga de Java, diremos que este método es un escuchador del evento *click* que puede generar el objeto `boton0`.

12. Añade el siguiente método al final de la clase `MainActivity`.

```
public void sePulsa0(View view){
    entrada.setText(entrada.getText()+"0");
}
```

Lo que hace es asignar como texto de entrada el resultado de concatenar al texto de entrada el carácter “0”.

13. Añade al botón con texto “0” el atributo `tag = “0”`.

14. Modifica el método `sePulsa0()` de la siguiente forma:

Para seleccionar entre código Java y Kotlin

```
entrada.setText(entrada.getText()+(String)view.getTag());
```

El resultado obtenido es equivalente al anterior. En algunos casos será interesante utilizar un mismo método como escuchador de eventos de varias vistas. Podrás averiguar la vista que causó el evento, dado que esta es pasada como parámetro del método. En el ejemplo sabemos que en el atributo `tag` guardamos el carácter a insertar. El atributo `tag` puede ser usado libremente por el programador para almacenar un objeto de la clase `Object` ([mas info](#)). (en la práctica podemos usar cualquier tipo de clase, dado que `Object` es la clase raíz de la que heredan todas las clases en Java). En nuestro caso hemos almacenado un objeto `String`, por lo que necesitamos una conversión de tipo. [Polimorfismo](#).

NOTA: Utiliza esta forma de trabajar en la práctica para no tener que crear un método `onClick` para cada botón.

15. Modifica el código de `sePulsa()` con el siguiente código:

Para seleccionar entre código Java y Kotlin

```
salida.setText(String.valueOf(Float.parseFloat(
    entrada.getText().toString()) * 2.0));
```

En este código el valor de entrada es convertido en `Float`, multiplicado por dos y convertido en `String` para ser asignado a salida.

16. Ejecuta el proyecto y verifica el resultado.

NOTA: *En este ejercicio no se ha realizado la verificación de que los datos introducidos por el usuario. Has de tener introducir datos válidos y en el orden adecuado.*

Unidad 4 Activities

Introducción a la unidad

En esta unidad seguimos trabajando con el diseño del interfaz de usuario. En lugar de tratar aspectos de diseño visual, como hicimos en la unidad anterior, vamos a tratar temas más relacionados con el código. En concreto nos centraremos en las *Actividades*. Estudiaremos también dos herramientas de gran utilidad para cualquier aplicación: la barra de acciones y la definición de las preferencias de configuración.

Nos vamos a centrar en el ejemplo de aplicación que estamos desarrollando, Mis Lugares, para añadirle diferentes actividades.

Objetivos:

- Describir cómo el interfaz de usuario en una aplicación Android estará formado por un conjunto de actividades.
- Mostrar cómo desde una actividad podemos invocar a otras y cómo podemos comunicarnos con ellas.
- Incorporar a nuestras aplicaciones ciertos elementos prácticos como son los menús o las preferencias.
- Describir cómo podemos utilizar y crear iconos en nuestras aplicaciones.

Actividades en Android

Objetivos:

Mostrar cómo podemos añadir nuevas actividades a nuestra aplicaciones.

El concepto de actividad en Android representa una unidad de interacción con el usuario. Corresponde a lo que coloquialmente llamamos una pantalla de la aplicación. Una aplicación suele estar formada por una serie de actividades, de forma que el usuario puede ir navegando entre actividades. En concreto, Android suele disponer de un botón (físico o en pantalla) que nos permite volver a la actividad anterior.

video [Actividades en Android](#)

Layout aunque no es imprescindible, como se verá en el siguiente ejemplo.

Toda actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Esta vista suele ser de tipo layout, aunque también puede ser una vista simple, como se verá en el siguiente ejemplo.

Una aplicación estará formada por un conjunto de actividades independientes; es decir, se trata de **clases independientes que no comparten variables**, aunque todas trabajan para un objetivo común. Otro aspecto importante es que **toda actividad ha de ser una subclase de Activity**.

Las aplicaciones creadas hasta ahora disponían de una única actividad. Esta era creada automáticamente y se le asignaba la vista definida en *res/layout/activity_main.xml*. Esta actividad era arrancada al comenzar la aplicación. A medida que nuestra aplicación crezca va a ser imprescindible crear nuevas actividades. En este apartado describiremos como hacerlo. Este proceso se puede resumir en cuatro pasos:

- *Crear un nuevo Layout para la actividad.
- *Crear una nueva clase descendiente de Activity. En esta clase tendrás que indicar que el Layout a visualizar es el desarrollado en el punto anterior.
- *Para que nuestra aplicación sea visible será necesario activarla desde otra actividad.
- ***De forma obligatoria tendremos que registrar toda nueva actividad en *AndroidManifest.xml***

Veamos un primer ejemplo de cómo crear una nueva actividad en la aplicación que estamos desarrollando.

Ejercicio: Implementación de una caja Acerca de

Vamos a crear una caja Acerca de... y visualizarla cuando se pulse el botón adecuado.

1. En primer lugar crea el fichero *res/layout/acercade.xml*. Para ello pulsa con el botón derecho sobre el explorador del proyecto en la carpeta *res/layout* y selecciona *New > Layout resource file*. Indica en *File name: acercade*.

2. Selecciona la lengüeta *Text* y copia el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="@dimen/text_margin"
    android:text="Este programa ha sido desarrollado como ejemplo en el curso de Android
para demostrar cómo se pueden lanzar nuevas actividades desde la actividad principal.">
</TextView>
```

3. Creamos ahora una nueva actividad, que será la responsable de visualizar esta vista. Para ello crea el fichero *AcercaDeActivity.java*, pulsando con el botón derecho sobre el nombre del paquete de la aplicación y seleccionando *New > Java Class*. En el campo *Name* introduce *AcercaDeActivity* y pulsa *Finish*. Reemplaza el código por:

```
public class AcercaDeActivity extends Activity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);
    }
}
```

}

Nota sobre Java/Kotlin: Puedes pulsar **Alt-Intro** en las dos clases modificadas para que automáticamente se añadan los paquetes que faltan.

- 4.** Pasemos ahora a crear un método en la actividad principal que será ejecutado cuando sea pulsado el botón Acerca de.

```
public void lanzarAcercaDe(View view)
{
    Intent i = new Intent(this, AcercaDeActivity.class);
    startActivity(i);
}
```

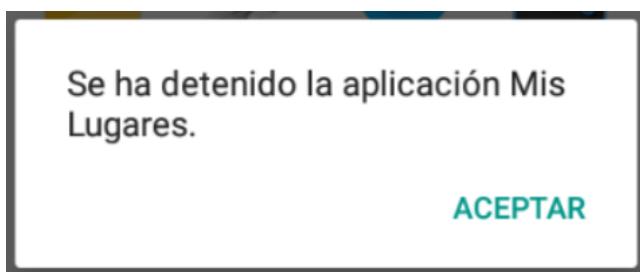
- 5.** Para asociar este método al botón edita el o *content_main.xml* en Mis Lugares. Selecciona la vista *Design* y pulsa sobre el botón Acerca de... y en la vista *Properties* busca el atributo *onClick* e introduce el valor *lanzarAcercaDe*.

- 6.** Selecciona la lengüeta *Text* y observa como, en la etiqueta <Button> correspondiente, se ha añadido el atributo:

```
android:onClick="lanzarAcercaDe"
```

NOTA: En caso de que exista algún recurso alternativo para el layout repite el mismo proceso.

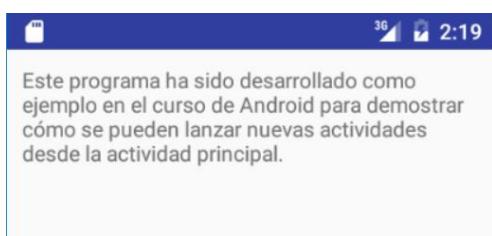
- 7.** Ejecuta ahora la aplicación y pulsa en el botón Acerca de. Observarás que el resultado no es satisfactorio ¿Qué ha ocurrido?



El problema es que toda actividad que ha de ser lanzada por una aplicación ha de ser registrada en el fichero *AndroidManifest.xml*. Para registrar la actividad, abre *AndroidManifest.xml*. Añade el siguiente texto dentro de la etiqueta <application ...></application>:

```
<activity android:name=".AcercaDeActivity"
    android:label="Acerca de ..."/>
```

- 8.** Ejecuta de nuevo el programa. El resultado ha de ser similar al mostrado a continuación:

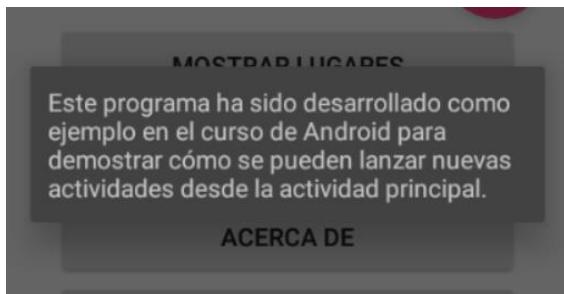


La vista mostrada en el ejemplo anterior no parece muy atractiva. Tratemos de mejorarla aplicando un tema. Como vimos en el capítulo anterior, un tema es una colección de estilos que define el aspecto de una actividad o aplicación. Puedes utilizar alguno de los temas disponibles en Android o crear el tuyo propio.

9. En este caso utilizaremos uno de los de Android. Para ello abre *AndroidManifest.xml* e introduce la línea subrayada:

```
<activity android:name=".AcercaDeActivity"
    android:label="Acerca de ..."
    android:theme="@style/Theme.AppCompat.Light.Dialog"/>
```

10. Ejecuta de nuevo el programa y observa cómo la apariencia mejora:



Ejercicio: Un escuchador de evento por código

Como acabamos de ver en un *layout* podemos definir el atributo XML `android:onClick` que nos permite indicar un método que será ejecutado al hacer *click* en una vista. A este método se le conoce como escuchador de evento. Resulta muy sencillo y además está disponible en cualquier descendiente de la clase *View*. Sin embargo esta técnica presenta dos inconvenientes. Solo está disponible para el evento `onClick()`. La clase *View* tiene otros eventos (`onLongClick()`, `onFocusChange()`, `onKey()`,...) para los que no se han definido un atributo xml. Entonces, ¿qué hacemos si queremos definir un evento distinto de `onClick()`? La respuesta la encontrarás este ejercicio:

1. Abre la clase *MainActivity*, y añade las líneas que aparecen subrayadas:

```
public class MainActivity extends Activity {
    private Button bAcercaDe;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        bAcercaDe = findViewById(R.id.button03);
        bAcercaDe.setOnClickListener(new OnClickListener() {
                public void onClick(View view) {
                        lanzarAcercaDe(null);
                }
        });
    }
    ...
}
```

N.D.E Me ha quedado:

```
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState); //Constructor del padre
    ...
    //Creamos el Listener del botón "acerca de"
    private Button bAcercaDe; //Declaración de un botón para "LanzarAcercaDe"
    ...
    bAcercaDe = findViewById(R.id.button3); //Creamos el objeto Button de
    activity_main
    bAcercaDe.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View view)
        {
            lanzarAcercaDe(null);
        }
    });
    ...
}
```

Nota sobre Java: Puedes pulsar **Alt+Intro** para que automáticamente se añadan los imports que faltan.
Para la clase OnClickListener selecciona android.view.View.OnClickListener.

2. Elimina el atributo añadido al botón:

1. android:onClick="lanzarAcercaDe"
3. Ejecuta la aplicación. El resultado ha de ser identico al anterior.

NOTA: Más adelante se estudiará con más detalle los escuchadores de evento.

Práctica: El botón salir

En el layout *activity_main.xml* (o *content_main.xml* en Mis Lugares) hemos introducido un botón con texto “Salir”. Queremos que cuando se pulse este botón se cierre la actividad. Para cerrar una actividad puedes llamar al método `finish()`; Llamar a este método es equivalente a cuando un usuario pulsa la tecla «retorno».

1. Realiza este trabajo utilizando un escuchador de evento por código.
2. Hazlo ahora con el atributo xml `android:onClick`.
3. Verifica que el resultado es el mismo en ambos casos.

NOTA: No es conveniente que en tus actividades incluyas un botón para cerrarlas. Un dispositivo Android siempre dispone de la tecla «retorno», que tiene la misma función.

Solución:

1. Para resolverlo mediante un escuchador por código, añade en el método `onCreate()` de la clase `MainActivity` el siguiente código:

```
public class MainActivity extends AppCompatActivity
{
    private Button bSalir; //Declaración de un botón para salir de la app
    ...
}
```

```
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState); //Constructor del padre
    ...
    //Listener para el botón salir
    bSalir = findViewById(R.id.button4);
    bSalir.setOnClickListener(new View.OnClickListener()
    {
        public void onClick(View view)
        {
            lanzarSalir();
        }
    });
    ...
}
//Salir de la app
public void lanzarSalir(){finish();}
...
}
```

2. Para resolverlo con el atributo onClick, añade en MainActivity el método:

```
public void salir(View view){
    finish();
}
```

Y añade el siguiente atributo al botón “Salir” en el *layout activity_main.xml*:

```
android:onClick="salir"
```

Intercambio de datos entre actividades

video [Intercambio de datos entre actividades](#)

Objetivos:

Aprender los mecanismos de comunicación entre actividades.

Cuando una actividad ha de lanzar a otra actividad en muchos casos necesita enviarle cierta información.

Android nos permite este intercambio de datos entre la activity llamada y la activity llamadora utilizando el mecanismo que es descrito a continuación:

Cuando lances una actividad usa el siguiente código:

```
//en ActividadLanzadora
Intent intent = new Intent(this, MiActividad.class);
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
startActivity(intent);
```

En la actividad lanzada podemos recoger los datos de la siguiente forma:

```
//En MiActividad
Bundle extras = getIntent().getExtras();
String s = extras.getString("usuario");
int i = extras.getInt("edad");
//imagino que se pueden añadir más extras
```

Cuando la actividad lanzada termina también podrá devolver datos que podrán ser recogidos por la actividad lanzadora de la siguiente manera.

```
//en ActividadLanzadora
//Para lanzar la actividad MiActividad
Intent intent = new Intent(this, MiActividad.class);
startActivityForResult(intent, 1234);
//1234 es "requestCode" un código para reconocer a la activity
//Que devuelver resultados
...
...
```

En la actividad llamada has de escribir:

```
//En MiActividad
Intent intent = new Intent();
intent.putExtra("resultado", "valor");
//imagino que aquí se pueden poner más puts
setResult(RESULT_OK, intent);
finish();

//en ActividadLanzadora
//Para recibir resultados de la actividad MiActividad lanzada desde esta otra
//actividad hay que sobreescribir el método...
```

```
@Override protected void onActivityResult(int requestCode, int resultCode,
Intent data)
{//También se puede utilizar un switch
    if (requestCode==1234 && resultCode==RESULT_OK)
    {
        String res = data.getExtras().getString("resultado");
        //Imagino que aquí se pueden recoger más resultados
    }
}
```

Práctica: Comunicación entre actividades.

1. Crea un nuevo proyecto con nombre *ComunicacionActividades* y tipo *Empty Activity*.
2. El *Layout* de la actividad inicial ha de ser similar al que se muestra abajo a la izquierda.
3. Introduce el código para que cuando se pulse el botón “Verificar” se arranque una segunda actividad. A esta actividad se le pasará como parámetro el nombre introducido en el *EditText*.
4. El *Layout* correspondiente a la segunda actividad se muestra a la derecha.
5. Al arrancar la actividad el texto del primer *TextView* ha de modificarse para que ponga “Hola ”+nombre recibido+”¿Aceptas las condiciones?”
6. En esta actividad se podrán pulsar dos botones, de forma que se devuelva a la actividad principal el String “Aceptado” o “Rechazado”, según el botón pulsado. Al pulsar cualquier botón se regresará a la actividad anterior.
7. En la actividad principal se modificará el texto del último *TextView* para que ponga “Resultado: Aceptado” o “Resultado: Rechazado”, según lo recibido.



Notas:

- Hay una clase llamada *Editable* para leer textos:
`Editable edNombre = editTextNombre.getText();`
- Se puede comparar una cadena con otra directa:
`String respuesta = data.getExtras().getString("resultado");
//Analizamos la información
if (respuesta.equals("sin nombre")) return;`

Añadiendo un menú en Android

video [Añadiendo un menú en Android](#)

Objetivos:

Introducir el uso de menús en las actividades .

Android permite asignar menús a las actividades. En las versiones de Android anteriores a 3.0 los dispositivos disponían de una tecla para mostrar el menú. En las versiones actuales el menú puede abrirse desde la barra de acciones. Este elemento, que tiene una gran importancia en el diseño de la interfaz de usuario en una aplicación Android, se estudiará en el siguiente punto.

NOTA: En el vídeo se habla de "menús contextuales". Este término no es adecuado dado que puede confundirse con otro tipo de menús. Un término más preciso sería "menú de actividad".

Ejercicio: Añadiendo un menú a una actividad

Podemos asignar un menú a nuestra actividad de forma muy sencilla.

1. Abre el fichero **res / menu / menu_main.xml**.

NOTA: El fichero de menú se crea automáticamente si seleccionas una actividad de tipo: Basic Activity o Scrolling Activity.

2. Reemplaza el contenido que se muestran a continuación:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

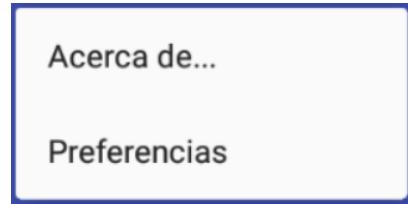
    <item android:id="@+id/action_settings"
          android:title="Preferencias"
          android:icon="@android:drawable/ic_menu_preferences"
          android:orderInCategory="100"
          app:showAsAction="never"/>

    <item android:title="Acerca de..."
          android:id="@+id/acercaDe"
          android:icon="@android:drawable/ic_menu_info_details"/>

</menu>
```

Como puedes ver cada ítem de menú tiene tres atributos principales: *id* que permite identificarlo desde el código, *title*, para asociarle un texto e *icon*, para asociarle un ícono. Los otros dos atributos se aplican cuando el menú actúa como una barra de acciones y serán explicados en el próximo punto.

3. A continuación se muestra la apariencia de este menú. Esta apariencia puede cambiar dependiendo de la versión de Android.



4. Para activar el menú, has de **introducir el siguiente código en la actividad que muestra el menú**. Posiblemente solo tengas que incluir el texto subrayado.

```
@Override public boolean onCreateOptionsMenu(Menu menu) //Infla el menú
{
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
@Override public boolean onOptionsItemSelected(MenuItem item) //Recibe la
selección del menú
{
    int id = item.getItemId(); //Identificador de ítem del menú pulsado
    if (id == R.id.action_settings)
    {
        return true; // No hace nada por ahora
    }
    if (id == R.id.acercaDe)
    {
        lanzarAcercaDe(); // Ejecuta el método que lanza la actividad
        return true;
    }
    // TODO Más opciones del main menú aquí
    //if (id == R.id.action_settings){ TODO tu código aquí; return true;}
    return super.onOptionsItemSelected(item);
}
```

5. Ejecuta el programa y pulsa en el ícono del menú en la esquina superior derecha. Han de aparecer los dos ítems de menú. Selecciona Acerca de... para pasar a la actividad correspondiente.

La Barra de Acciones

video [La barra de acciones \(ActionBar\)](#)

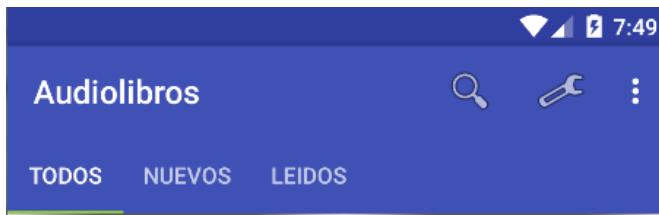
Objetivos:

Mostrar cómo a partir de la versión 3.0 los menús son mostrados en la barra de acciones y como podemos configurar sus opciones.

ActionBar

Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones o **ActionBar**. Situada en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones. La barra de acciones aglutina varios elementos; los más habituales son el ícono de la aplicación con su nombre y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón **Overflow**. Si la aplicación dispone de pestañas (**tabs**), estas podrán situarse en la barra de acciones. También pueden añadirse otros elementos, como listas desplegables y otros tipos de widgets incrustados, como el widget de búsqueda que veremos más adelante.

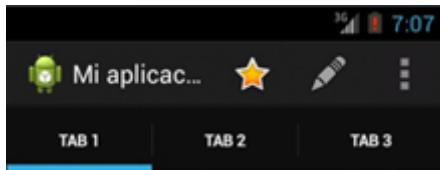
En caso de disponer de menos tamaño de pantalla el sistema puede redistribuir los elementos y pasar alguna acción al menú de «Overflow». Por ejemplo, en un móvil la barra de acciones anterior se podría ver de la siguiente manera:



Existen dos clases que nos permiten añadir la barra de acciones: **ActionBar** y **ToolBar**. La clase **ActionBar** aparece en la versión 3.0. Por defecto la barra de acciones es incluida en todas las actividades. Si queremos que no aparezca tenemos que asignar un tema especial a la actividad. Por ejemplo **Theme.AppCompat.NoActionBar** o cualquiera que acabe en **.NoActionBar**.

ToolBar

La clase **ToolBar** aparece con la versión 5.0. Cambia el diseño de la barra de acciones para que siga las especificaciones de **Material Design**. Puede usarse en versiones anteriores dado que no se incorpora al API de la versión 5.0, sino a una la librería de compatibilidad **appcompat-v7**. A diferencia de **ActionBar**, la barra de acciones no es incrustado de forma automática, si no que hay que incluirla en el layout con la etiqueta <Toolbar>. Esto nos permite situarla en la posición que queramos y nos da más opciones de configuración.



Añadir un *ToolBar* a la aplicación es muy sencillo. Normalmente no es necesario realizarlo si al crear un proyecto has seleccionado Basic Activity o Scrolling Activity, dado que en este caso ya se ha añadido un Toolbar. Se ha incluido el siguiente ejercicio para los casos en que partes de un proyecto donde se ha incluido un ActionBar (seleccionando Empty Activity) o en proyectos creados con versiones anteriores.

Ejercicio: Añadir una barra de acciones con Toolbar.

1. Crea un nuevo proyecto basado en Empty Activity o abre uno que en su layout no aparezca la etiqueta <Toolbar>. Ejecuta el proyecto y verifica que aparece la barra de acciones.
2. El primer paso va a ser anular la creación automática de la barra de acciones basada en ActionBar. Para eso, accede a *res/values/styles.xml* y cambia el tema de la aplicación por *Theme.AppCompat.Light.NoActionBar*. Verifica que al ejecutar la barra desaparece.
3. También puedes ocultar la barra de estado de Android, donde se muestran las notificaciones y la hora. Para ello añade el siguiente ítem al tema de la aplicación:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <!-- Customize your theme here. -->
    <item name="android:windowFullscreen">true</item>
    ...

```

4. Añade al layout de la actividad el código siguiente dentro del contenedor raíz:

```
<androidx.appcompat.widget.Toolbar
    app:title="misTools"
    app:titleTextColor="#FFFFFF"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Con los dos últimos atributos podemos controlar el tema aplicado al Toolbar y al menú de overflow.

5. Si lo añades dentro de un ConstraintLayout aseguráte de indicar las restricciones de posición:

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
```

6. Verifica que la actividad desciende de *AppCompatActivity*.

7. Añade en el método *onCreate()* el siguiente código:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);  
setSupportActionBar(toolbar);
```

Has de utilizar el import:

```
import androidx.appcompat.widget.Toolbar;
```

8. Comprueba que el resultado es equivalente a crear la barra de acciones de forma implícita con ActionBar.

9. Puedes situar libremente la barra de acciones dentro del layout. Trata de que aparezca en la parte inferior.

Menú en toolbar

Ejercicio: Añadiendo una barra de acciones a nuestra aplicación. ¿no será un menú?

Añade en MainActivity.java los imports:

```
import android.view.Menu;  
import android.view.MenuItem;
```

Y dos métodos a la clase MainActivity:

```
@Override public boolean onCreateOptionsMenu(Menu menu) //Infla el menú  
{  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.menu_main, menu);  
    return true;  
}  
@Override public boolean onOptionsItemSelected(MenuItem item) //Recibe la  
selección del menú  
{  
    int id = item.getItemId();  
    //noinspection SimplifiableIfStatement  
    // TODO opciones del main_menú aquí  
    //if (id == R.id.action_settings){ TODO tu código aquí; return true;}  
    return super.onOptionsItemSelected(item);
```

Reemplaza el contenido del fichero res/menu/menu_main.xml por:

```
<?xml version="1.0" encoding="utf-8"?>  
<menu xmlns:android="http://schemas.android.com/apk/res/android"  
      xmlns:app="http://schemas.android.com/apk/res-auto"  
      xmlns:tools="http://schemas.android.com/tools"  
      tools:context=".MainActivity">  
    <item android:id="@+id/action_settings"  
          android:title="Configuración"  
          android:icon="@android:drawable/ic_menu_preferences"  
          android:orderInCategory="5"  
          app:showAsAction="never"/>  
    <item android:title="Acerca de..."  
          android:id="@+id/acercaDe"  
          android:icon="@android:drawable/ic_menu_info_details"  
          android:orderInCategory="10"  
          app:showAsAction="ifRoom|withText"/>
```

```

<item android:title="Buscar"
      android:id="@+id/menu_buscar"
      android:icon="@android:drawable/ic_menu_search"
      android:orderInCategory="115"
      app:showAsAction="always|collapseActionView"/>
</menu>

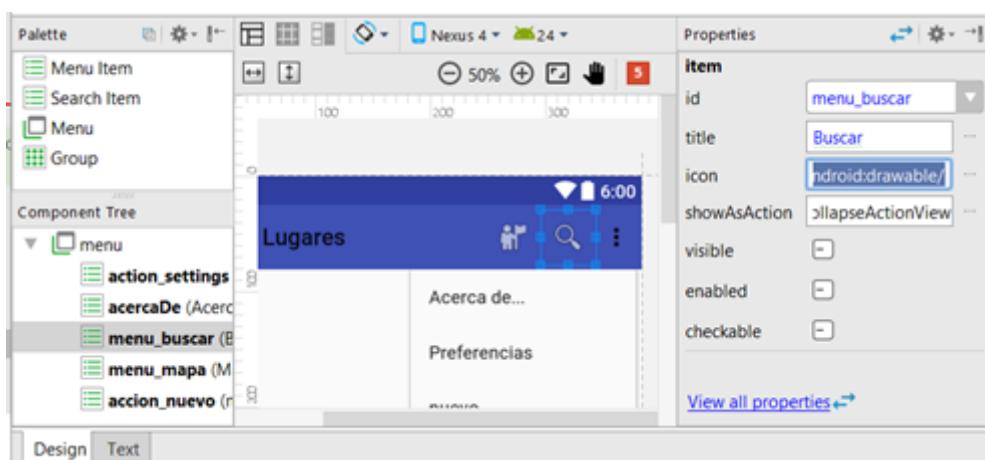
```

En este fichero se define los iconos y las acciones a mostrar en la barra. Las acciones que indiquen en el atributo showAsAction la palabra always se mostrarán siempre, sin importar si caben o no. El uso de estas acciones debería limitarse, lo ideal es que haya una o dos, ya que al forzar que se visualicen todas podrían verse incorrectamente. Las acciones que indiquen ifRoom se mostrarán en la barra de acciones si hay espacio disponible, y se moverán al menú de Overflow si no lo hay. En esta categoría se deberían encontrar la mayoría de las acciones. Si se indica never, la acción nunca se mostrará en la barra de acciones, sin importar el espacio disponible. En este grupo se deberían situar acciones como modificar las preferencias, que deben estar disponibles para el usuario, pero no visibles en todo momento. Las acciones se ordenan de izquierda a derecha según lo indicado en orderInCategory, con las acciones con un número más pequeño más a la izquierda. Si no caben todas las acciones en la barra, las que tienen un número mayor se mueven al menú de Overflow.

Ejecuta la aplicación. Podrás ver como aparece la barra de acciones en la parte de arriba, con los botones que hemos definido.



Android Studio incorpora un editor visual de menús que nos permite crear menús sin necesidad de escribir código xml.



Creación y uso de iconos

video [Añadiendo iconos en Android](#)

Objetivos:

- Conocer algunos conceptos y herramientas para diseñar iconos en Android.

En el apartado anterior hemos creado una barra de acciones donde se mostraban algunos iconos. Se han utilizado iconos disponibles en el sistema Android, es decir, **recursos ya almacenados en el sistema**. Otra alternativa es crear **tus propios iconos** y almacenarlos en la carpeta res/mipmap. En este apartado aprenderemos a hacerlo.

En Android se utilizan diferentes tipos de iconos según su utilidad. La siguiente tabla muestra los más importantes:

Tipo de iconos	Finalidad	Ejemplos
Lanzadores	Representa la aplicación en la pantalla principal del dispositivo	 
Barra de acciones	Opciones disponibles en la barra de acciones	 
Notificaciones	Pequeños iconos que aparecen en la barra de estado (véase capítulo 8)	   
Otros	También es muy frecuente el uso de iconos en cuadros de dialogo y en RecyclerView, etc.	

A la hora de diseñar tus propios iconos, has de tener en cuenta algunos aspectos. En primer lugar, recuerda que Android ha sido concebido para ser utilizado en dispositivos con una gran variedad **de densidades gráficas**. Este **rango puede ir desde 100 píxeles/pulgada (ldpi) hasta 340 píxeles/pulgada (xhdpi)**. Por lo tanto, vas a tener que crear tus iconos en varias densidades gráficas.

Resulta interesante que **plantes tus diseños con una resolución como mínimo de 300 píxeles/pulgada**. Luego puedes usar las herramientas del sistema para obtener estos iconos en

diferentes densidades. Si para alguna densidad el resultado no es el adecuado tendrás que realizar algunos retoques (en el siguiente ejercicio se muestra un ejemplo).

En segundo lugar, tu aplicación se ejecutará dentro de un sistema donde se utilizan ciertas guías de estilo. Si quieres que tus iconos no desentonen, lee la documentación que se indica a continuación.

Enlaces de interés:

Guía de estilo para iconos: La siguiente página describe las guías de estilo para los iconos en Material Design:

<https://material.google.com/style/icons.html>

Recursos de iconos: En las siguientes páginas puedes encontrar gran variedad de iconos:

<https://materialdesignicons.com>

https://design.google.com/icons/#ic_list

<https://android-material-icon-generator.bitdroid.de/>

Ejercicio: Creación de iconos personalizados

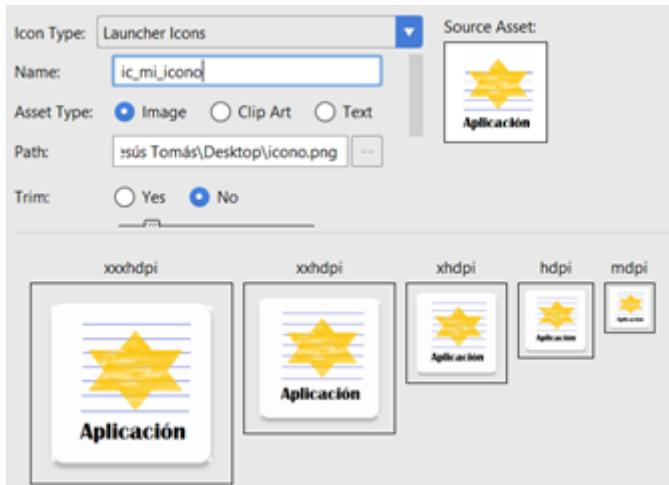
Veamos un ejemplo práctico de como crear un ícono: El diseñador gráfico de nuestra empresa nos acaba de pasar el ícono asociado para iniciar la aplicación que estamos diseñando (*launcher icon*).



(Pulsa con el botón derecho sobre el gráfico anterior para descargarlo)

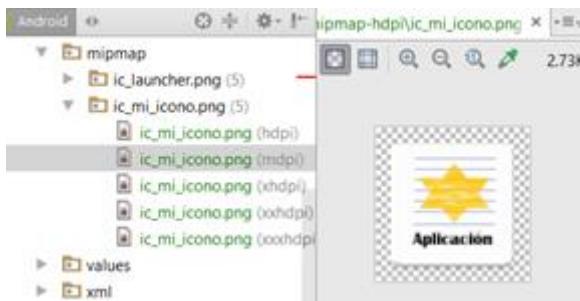
Para asignarlo a una aplicación realiza los siguientes pasos:

1. Descarga el gráfico anterior y llámale *icon.png*.
2. En el explorador proyecto pulsa con el botón derecho sobre la carpeta *res* y **selecciona File / New / Image Assets**.
3. En el campo *Name*: introduce *ic_mi_icono*; en *Asset Type: Image* y en *Path*: indica el fichero descargado en el paso 1. El resultado ha de ser similar a:



En la parte inferior de la página podrás previsualizar cómo quedarán las imágenes para diferentes densidades gráficas.

4. Pulsa el botón *Next* y en la siguiente pantalla pulsa *Finish*.
5. Verifica como dentro de la carpeta *res/mipmap* se han creado 5 recursos alternativos para diferentes densidades gráficas:



6. El resultado es aceptable para algunas opciones. Pero en algunos casos el texto no se lee o unas líneas horizontales quedan más gruesas que otras. Puede ser interesante retocar estos fichero png usando un editor de gráficos. Por ejemplo, retoca los iconos *hdpi* y *mdpi* para que se pueda leer más claramente el texto "Aplicación" y redibuja las líneas horizontales para que tenga el mismo grosor y separación.

7. Para que el nuevo ícono sea utilizado como lanzador de nuestra aplicación, abre *AndroidManifest.xml* y reemplaza el valor del atributo *icon* como se muestra a continuación:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launchermi_icono"
```

8. Visualiza el resultado instalando la aplicación en diferentes dispositivos con diferentes densidades gráficas.

Práctica: Creación de iconos para la aplicación

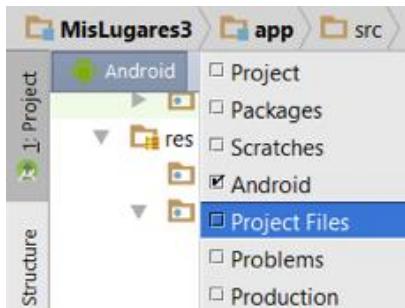
1. Dibuja o busca en Internet un gráfico que sea adecuado para usar como ícono de inicio en la aplicación.
2. Repite los pasos indicados en el ejercicio anterior para crear los iconos en las diferentes densidades gráficas.

Ejercicio: Creación de iconos para la aplicación Mis Lugares

1. Descarga y descomprime el siguiente fichero:

http://www.dcomg.upv.es/~jtomas/android/ficheros/mislugares_iconos.zip

2. Utiliza los iconos que contiene para asociarlos a la aplicación Mis Lugares. Para copiarlos más fácilmente, pulsa en el ícono y cambia la vista del explorador de proyecto a Project Files.



3. Arrastra el fichero *ic_launcher.png* de cada una de las carpetas a las carpetas con el mismo nombre del proyecto.

Añadir preferencias en Android

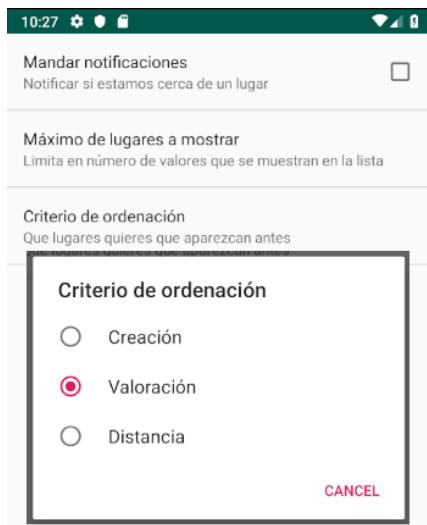
[video\[Tutorial\] Añadir preferencias en Android](#)

Objetivos:

- Definir las preferencias de usuario en una aplicación.

Android nos facilita la **configuración de nuestros programas**, al permitir añadir una lista de **preferencias** que el usuario podrá modificar. Por ejemplo, el usuario podrá indicar con qué frecuencia la aplicación ha de sincronizarse con el servidor o si queremos que se lancen notificaciones. **Las preferencias también pueden utilizarse para que tu aplicación almacene información de forma permanente**. En el capítulo 9 se estudiará cómo realizar esta función.

NOTA: A continuación se proponen una serie de ejercicios para añadir preferencias en “Mis Lugares para que correspondan con la siguiente captura:



El recurso “preference”

Ejercicio paso a paso: Añadiendo preferencias.

- Abre el proyecto Mis Lugares.
- Pulsa con el botón derecho sobre `res` y selecciona la opción `New > Android resource file`.
- Completa los campos `File name: preferencias` y `Resource type: XML`. Se creará el fichero `res/xml/preferencias.xml`. Puede que tengas que crear el directorio `xml`.
- Edita este fichero. Selecciona la lengüeta `Text` e introduce el siguiente código:

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:key="preferencias_principal" >
    <CheckBoxPreference
        android:key="notificaciones"
        android:title="Mandar notificaciones"
        android:summary="Notificar si estamos cerca de un lugar"/>
    <EditTextPreference
        android:key="maximo"
        android:title="Máximo de lugares a mostrar"
        android:summary="Limita en número de valores que se muestran en la lista"
        android:inputType="number"
        android:defaultValue="12"/>
    <ListPreference
        android:key="orden"
        android:title="Criterio de ordenación"
        android:summary="Que lugares quieres que aparezcan antes"
        android:entries="@array/tiposOrden"
        android:entryValues="@array/tiposOrdenValores"
        android:defaultValue="0"/>
</PreferenceScreen>
```

El significado de cada etiqueta y atributo se descubre fácilmente si observas el resultado obtenido que se muestra a continuación. El atributo inputType permite configurar el tipo de teclado que se mostrará para introducir el valor. Coinciden con el atributo de EditText. Para ver los posibles valores

consultar: developer.android.com/reference/android/widget/TextView.html#attr_android:inputType

5. Para almacenar los valores del desplegable, has de crear el fichero */res/values/arrays.xml* con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="tiposOrden">
        <item>Creación</item>
        <item>Valoración</item>
        <item>Distancia</item>
    </string-array>
    <string-array name="tiposOrdenValores">
        <item>0</item>
        <item>1</item>
        <item>2</item>
    </string-array>
</resources>
```

6. Crea ahora una nueva clase PreferenciasFragment con el siguiente código:

```
package com.example.mislugares;
import android.os.Bundle;
import android.preference.PreferenceFragment;
public class PreferenciasFragment extends PreferenceFragment
{
```

```
@Override public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferencias);
}
```

Nota sobre Java: Puedes pulsar **Alt+Intro** para que automáticamente se añadan los paquetes que faltan.

La clase PreferenceFragment permite crear un fragmento que contiene una ventana con las opciones de preferencias definidas en un recurso XML. Un fragmento es un elemento que puede ser incrustado dentro de una actividad. El uso de fragmentos se estudia con más detalle en la última unidad del curso.

7. Ahora vamos a crear una actividad que simplemente muestre el fragmento anterior. Crea la clase PreferenciasActivity con el siguiente código:

```
package com.example.mislugares;
import android.app.Activity;
import android.os.Bundle;
public class PreferenciasActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction()
            .replace(android.R.id.content, new PreferenciasFragment())
            .commit();
    }
}
```

Desde una actividad podemos visualizar un *fragment* en tiempo de ejecución. Para ello utilizamos el manejador de *fragments* de la actividad (`getFragmentManager()`) y comenzamos una transacción (`beginTransaction()`). Una transacción es una operación de insertado, borrado o reemplazo de *fragments*. En el ejemplo vamos a reemplazar el contenido de la actividad por un nuevo *fragment* de la clase `PreferenciasFragment`. Finalmente se llama a `commit()` para que se ejecute la transacción. No hay que olvidar registrar toda nueva actividad en `AndroidManifest.xml`.

8. En el `main:menu.xml` hay que añadir un nuevo ítem:

```
<item android:id="@+id/preferencias"
      android:title="preferencias"
      android:icon="@android:drawable/ic_menu_preferences"
      android:orderInCategory="4"
      app:showAsAction="never"/>
```

9. Añade a `MainActivity.java` el método `lanzarPreferencias()`. Este método ha de tener el mismo código que `lanzarAcercaDe()` pero lanzando la actividad `PreferenciasActivity`. Al pulsar el botón con texto “Preferencias” haz que se llame a `lanzarPreferencias()`.

```
public class MainActivity extends AppCompatActivity
{
    // Métodos de MainActivity

    public void lanzarPreferencias()
    {
        Intent intentPreferencias = new Intent(this, PreferenciasActivity.class);
        startActivity(intentPreferencias);
    }
}
```

10. Para activar la configuración desde la opción de menú añade el siguiente código en el fichero MainActivity.java en el método onOptionsItemSelected().

```
@Override public boolean onOptionsItemSelected(MenuItem item) //Recibe la selección del menú
{
    int id = item.getItemId();
    ...
    if (id==R.id.preferencias)
    {
        lanzarPreferencias();
        return true;
    }
    ...
}
```

MainActivity desde un método, lanza con un intent la activity.

```
Intent intentPreferencias = new Intent(this, PreferenciasActivity.class);
startActivity(intentPreferencias);
```

PreferenciasActivity: en onCreate ejecuta el FragmentManager (FragmentManager)

```
getFragmentManager()// fragment Manager
.beginTransaction()//Comienza la transacción
.replace(android.R.id.content, new PreferenciasFragment())//Introduce nuestro fragment
.commit();//Ejecuta
```

PreferenciasManager: heredada de la clase abstracta PreferenceManager, y se encarga de leer el recurso de vistas de xml (preferencias.xml->usa values.array.xml)

```
addPreferencesFromResource(R.xml.preferencias);
```

11. Arranca la aplicación y verifica que puedes lanzar las preferencias mediante las dos alternativas, el botón y el menú .

Organizando preferencias

Cuando el número de preferencias es grande resulta interesante organizarlas de forma adecuada. Una posibilidad consiste en dividirlas en varias pantallas. De forma que cuando se seleccione una opción en la primera pantalla, se abra una nueva pantalla de preferencias. Para organizar las preferencias de esta forma usa el siguiente esquema:

```
<PreferenceScreen>
    <CheckBoxPreference .../>
    <EditTextPreference .../>
    ...
    <PreferenceScreen android:title="Notificaciones por correo"> <CheckBoxPreference .../>
    <EditTextPreference .../> ... </PreferenceScreen> </PreferenceScreen>
```

Práctica: Organizando preferencias(I)

1. Crea una nueva lista de preferencias <PreferenceScreen> dentro de la lista de preferencias del fichero *res/xml/preferencias.xml*

```
...
<PreferenceScreen android:title="Notificaciones por correo">
    <ListPreference
        android:key="orden"
        android:title="Criterio de ordenación"
        android:summary="Que lugares quieres que aparezcan antes"
        android:entries="@array/tiposNotificacion"
        android:entryValues="@array/tiposNotificacionValores"
        android:defaultValue="0"/>
</PreferenceScreen>
...
```

2. Asígnale al parámetro android:title el valor “Notificaciones por correo”.

3. Crea dos arrays más en arrays.xml

```
<string-array name="tiposNotificacion">
    <item>Bluetooth</item>
    <item>Email</item>
    <item>Wifi</item>
</string-array>

<string-array name="tiposNotificacionValores">
<item>0</item>
<item>1</item>
<item>2</item>
</string-array>
```

5. Prueba el resultado.

Otra alternativa para organizar las preferencias consiste en agruparlas por categorías. Con esta opción se visualizarán en la misma pantalla, pero separadas por grupos. Has de seguir el siguiente esquema:

Práctica: Organizando preferencias(II)

Modifica la práctica anterior para que en lugar de mostrar las propiedades en dos pantallas, aparezcan en una sola.

Como se almacenan las preferencias de usuario

Si un usuario modifica el valor de una preferencia, este quedará almacenado de forma permanente en el dispositivo. Para conseguir esta persistencia Android almacena las preferencias seleccionadas en un fichero XML dentro de la carpeta `data/data/nombre.del.paquete/files/shared_prefs`. Donde `nombre.del.paquete` ha de ser reemplazado por el paquete de la aplicación. El nombre del fichero para almacenar las preferencias de usuario ha de ser siempre `nombre.del.paquete_preferences.xml`. Esto significa que solo puede haber unas preferencias de usuario por aplicación. Pero puede haber otros ficheros de preferencia, pero a diferencia de las preferencias de usuario no pueden ser editadas directamente por el usuario sino que hay que acceder a ellas por código.

Accediendo a los valores de las preferencias con código

Por supuesto, será necesario acceder a los valores de las preferencias para alterar el funcionamiento de nuestra aplicación. El siguiente ejemplo nos muestra cómo realizarlo:

```
public void mostrarPreferencias(View view)
{
    SharedPreferences pref =
        PreferenceManager.getDefaultSharedPreferences(this);
    String s = "notificaciones: " + pref.getBoolean("notificaciones",true)
              + ", máximo a listar: " + pref.getString("maximo","?");
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

El código comienza creando el objeto pref de la clase SharedPreferences y le asigna las preferencias definidas para la aplicación. A continuación crea el String s y le asigna los valores de dos de las preferencias. Se utilizan los métodos `pref.getBoolean()` y `pref.getString()`, que disponen de dos parámetros: el valor de key que queremos buscar ("notificaciones" y "maximo") y el valor asignado por defecto en caso de no encontrar esta key.

Finalmente se visualiza el resultado utilizando la clase `Toast`. Los tres parámetros indicados son el contexto (nuestra actividad), el String a mostrar y el tiempo que se estará mostrando esta información.

Ejercicio paso a paso: Accediendo a los valores de las preferencias

1. Copia la función anterior en la clase `MainActivity`.
2. Asignala al atributo `onClick` del botón *Mostrar Lugares* el método anterior.
3. Visualiza también el resto de las preferencias que hayas introducido.

Acceder a objetos globales de la aplicación

video [La clase Application en Android.](#)

Objetivos:

- Aprender a utilizar objetos globales de la aplicación
-

Cada uno de los componentes de una aplicación se escribe en una clase separada. Esto hace que en muchas ocasiones resulte complicado compartir objetos entre estos componentes. Para poder acceder a una información global desde cualquier clase de nuestro proyecto, podemos utilizar el modificador static. De esta forma, no será necesario conocer la referencia a un objeto de la clase, solo con indicar el nombre de la clase podremos acceder a esta información.

Otra alternativa, muy similar a la anterior, es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia a la que se podrá acceder desde cualquier sitio utilizando un método estático. Lo veremos más adelante.

Una tercera **alternativa específica de Android** consiste en crear un descendiente de la clase Application. En el siguiente punto se explica cómo hacerlo.

La clase Application

Esta clase ha sido **creada en Android para almacenar información global a toda la aplicación**.

Veamos cómo usarla en tres pasos:

1.

1. **Crea una clase descendiente de Application** que contenga la información global y los métodos asociados para acceder a esta información. Mira el ejemplo:

```
public class Aplicacion extends Application {  
    private int saldo;  
    @Override public void onCreate() {  
        super.onCreate();  
        SharedPreferences pref = getSharedPreferences("pref", MODE_PRIVATE);  
        saldo = pref.getInt("saldo_inicial", -1);  
    }  
    public int getSaldo()  
    {  
        return saldo;  
    }  
  
    public void setSaldo(int saldo){  
        this.saldo=saldo;  
    }  
}
```

```
}
```

En nuestra aplicación queremos que el usuario disponga de un saldo de puntos, con los que podrá ir desbloqueando ciertas características especiales. **La clase Application es descendiente de Context**, por lo que tendremos acceso a todos los métodos relativos a nuestro contexto. Entre estos métodos se incluye `getSharedPreferences`, para acceder a un fichero de preferencias almacenado en la memoria interna de nuestra aplicación. La clase Application permite sobrescribir los siguientes métodos:

-**`onCreate()`** llamado cuando se cree la aplicación. Puedes usarlo para inicializar los datos.

-**`onConfigurationChanged(Configuration nuevaConfig)`** llamado cuando se realicen cambios en la configuración del dispositivo, mientras que la aplicación se está ejecutando.

-**`onLowMemory()`** llamado cuando el sistema se está quedando sin memoria trata de liberar toda la memoria que sea posible y podría descargar nuestra app, en este método podemos guardar datos.

-**`onTrimMemory(int nivel)`** (desde nivel API 14) llamado cuando el sistema determina que es un buen momento para que una aplicación recorte memoria. Esto ocurrirá, por ejemplo, cuando está en el fondo de la pila de actividades y no hay suficiente memoria para mantener tantos procesos en segundo plano. Además, se nos pasa como parámetro el nivel de necesidad. Algunos valores posibles son: `TRIM_MEMORY_COMPLETE`, `TRIM_MEMORY_BACKGROUND`, `TRIM_MEMORY_MODERATE`, ... Entonces debemos actuar guardando variables que han de ser recordadas en la próxima ejecución de la aplicación.

2. **Registra la clase creada en AndroidManifest.** Para ello busca la etiqueta `<application>` y añade el atributo `name`, con el nombre de la clase creada:

```
<application  
    android:name="Aplicacion"  
    android:allowBackup="true"  
    android:icon="@drawable/ic_launcher"  
    android:label="@string/app_name"  
    android:theme="@style/AppTheme">  
    ...
```

3. Puedes **obtener una referencia a tu clase `<application>`** con este código:

```
Aplicacion aplicacion = (Aplicacion) contexto.getApplication();
```

Donde contexto es una referencia a la clase Context. En caso de estar en un descendiente de esta clase (como Activity, Service,...) no es necesario disponer de esta referencia, la misma clase ya es un Context. Por lo tanto, podríamos escribir:

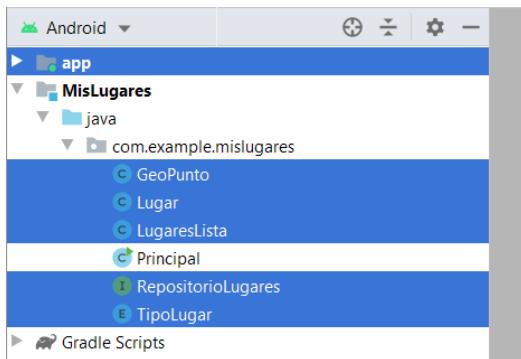
```
Aplicacion aplicacion = (Aplicacion) getApplication();  
incluso directamente:  
int miSaldo = ((Aplicacion) getApplication()).getSaldo();
```

ahora **aplicación** es un objeto desde el que podemos leer las variables globales como:

```
int miSaldo aplicación.getSaldo();
```

Ejercicio: Añadir la clase Application en Mis Lugares.

1. En el primer capítulo se creó el proyecto *MisLugaresJava* Abre este proyecto y selecciona las clases GeoPunto, Lugar, RepositorioLugares, LugaresLista y TipoLugar (puedes seleccionar varios ficheros manteniendo la tecla *Ctrl* pulsada). Con el botón derecho selecciona la opción *Copy*. Con el botón derecho pulsa sobre *com.example.mislugares* del proyecto *MisLugares* y selecciona la opción *Paste*.



2. Abre el proyecto *MisLugares*.
3. Creamos una nueva clase. Para ello pulsa con el botón derecho sobre el *java / com.example.mislugares* y selecciona *New > Java Class*. Introduce como nombre de la clase Aplicacion. En ella vamos a almacenar los objetos que queremos usar globalmente en toda la aplicación como RepositorioLugares. Reemplaza su código por el siguiente:

```
4. package com.example.mislugares;
import android.app.Application;
public class Aplicacion extends Application {
    public RepositorioLugares lugares = new LugaresLista();
    @Override public void onCreate() {
        super.onCreate();
    }
    public RepositorioLugares getLugares() {
        return lugares;
    }
}
```

Nota: Tras incluir nuevas clases tendrás que indicar los imports adecuados. Pulta «Alt+Intro» en Android Studio para que lo haga automáticamente. Aparecerá algún error dado que la clase *RepositorioLugares* aún no ha sido creada. No te preocupes, lo haremos más adelante.

5. Registra el nombre de la clase en *AndroidManifest*, añadiendo la línea que aparece en negrita.

```
<application
```

```
    android:name="Aplicacion"  
    android:allowBackup="true"
```

...

6. Puedes ejecutar el proyecto para verificar que sigue funcionando. No has de notar diferencias con respecto a la versión anterior, no hemos añadido nuevas funcionalidades.

Uso de la arquitectura Clean

Objetivos:

- Aprender a organizar el código
-

Este apartado trata de dar una visión introductoria a la programación en Android, por lo tanto, el diseño de arquitecturas de software queda algo alejado de sus objetivos. No obstante, pensamos que puede ser interesante comentar algunos conceptos y tratar de seguir unos ejemplos con una arquitectura adecuada.

A medida que **una aplicación crece** comprobarás que cada vez resulta más **difícil de mantener**. Si no seguimos unas **reglas claras en el desarrollo**, el caos está garantizado. Un error típico en Android suele darse al tratar de dar demasiada responsabilidad a las actividades. Estas pueden llegar a tener cientos de líneas de código, lo que las hace muy complicadas de mantener. Para estructurar las clases de la aplicación nos vamos a inspirar en la **arquitectura Clean**. Aunque de forma muy simplificada, no vamos a realizar inyección de dependencias, usar patrones como modelo-vista-controlador, ni otros temas que complicarían en exceso la aplicación. En una primera aproximación y dado el tamaño de la aplicación, creo que sería excesivo. Clean no es una arquitectura tal cual, si no **una serie de guías y buenas prácticas en el desarrollo de software**. Fue definida por Rober C Martin (*Uncle Bob*) en su charla “*Architecture the lost years*”, donde exponía una serie de problemas y el alto acoplamiento de los desarrollos de software tanto a los modelos de datos como a la interfaz.

Clean **define una serie de capas** y otorga una responsabilidad a cada una, pero no entra en profundidad en los detalles de implementación y cómo se deben resolver los problemas. El objetivo es **escribir software que esté lo menos acoplado posible a nuestro modelo de datos**, a la representación de este y al framework que estemos usando. Esto va a incrementar la estabilidad de nuestro código ya que **va a hacer más fácil cambiar las partes dependientes del sistema**. Va a facilitar la portabilidad a otros entornos (como iOS o Web) dado que gran parte del código es independiente del framework. También va a permitir postergar decisiones de implementación, como por ejemplo la persistencia o el uso de red. Podemos hacer una primera versión de nuestro software que guarde los datos de forma local y de una forma sencilla cambiarlo a online. O elegir el framework de persistencia cuando nos sea necesario y no antes. Aunque dentro de Clean existe variantes, en la aplicación Mis Lugares vamos a organizar las clases en 4 capas:

Capa de Modelo o Dominio.

También se utiliza el nombre Lógica de Negocio. Está formada por las clases que representan la lógica interna de la aplicación y cómo representamos los datos con los que vamos a trabajar. Muchas clases de esta capa se conocen como **POJO (Plain Old Java Object)** al tratarse de **clases Java puras**. Ejemplos de POJO serían las clases Lugar o GeoPunto. **No es conveniente que en estas clases se utilicen APIs externos**. Si abres las clases que hemos indicado, podrás comprobar que **no necesitan ningún import**.

Capa de Datos.

En esta capa estarían las **clases encargadas de guardar de forma permanente los datos y cómo acceder a ellos**. Suelen representar bases de datos, servicios Web, preferencias, ficheros JSON... También es conocida como capa de almacenamiento o persistencia.

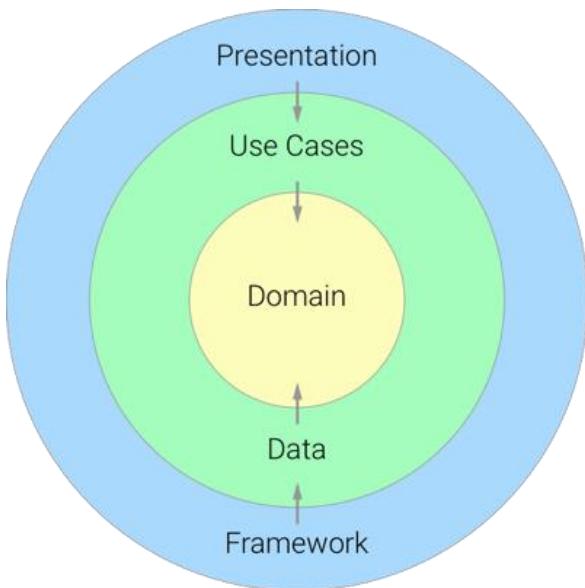
Capa de Casos de Uso.

Los casos de uso son clases que van a definir las operaciones que el usuario puede realizar con nuestra aplicación. Esta capa no sería estrictamente necesaria, pero resulta muy interesante para **tener enumeradas las diferentes acciones que vamos a implementar**. Además, va a permitir **quitar mucha responsabilidad a las actividades**. Los casos de uso también se conocen como interactores.

Capa de Presentación.

Representa la interfaz de usuario, por lo que está formada por las **actividades, fragments, vistas** y otros elementos con los que interactúa el usuario.

Una de las características más importantes de esta arquitectura es la **regla de dependencia entre capas**. Para representar las dependencias se suelen usar un diagrama en forma de círculos concéntricos. **Las capas más internas son aquellas que están más cercanas a nuestra lógica de dominio, no deben depender de las capas más externas del software**, aquellas que están más cerca a los agentes externos como el framework, o el interfaz de usuario.



Por ejemplo, la clase Lugar que pertenecería a la capa de Modelo, va a poder ser utilizada por el resto de las capas y no puede usar otras clases que no sean de su capa. Por el contrario, una actividad perteneciente a la capa de presentación no debería usarse por el resto de las capas y puede usar cualquier capa interior.

Nota: En la literatura la capa de casos de uso es más interna que la de datos. En esta implementación se ha realizado al revés. Reamente no vamos a seguir la Arquitectura Clean de forma estricta. En nuestra implementación la capa de Usos de Datos va a tener dependencias con la capa de Presentación, cosa que habría que evitar.

Organizando las clases en paquetes.

El número de clases de un proyecto Android puede ser muy elevado, por lo que resulta complicado localizarlas. Para resolver este problema, resulta frecuente organizar las clases en diferentes paquetes. Podemos usar diferentes criterios, por ejemplo, por módulos del proyecto (como autenticación, visualización, mapas...) Otro criterio podría ser por entidades (como lugares, usuarios...) También podemos utilizar la función de la clase (como actividades, fragments, adaptadores...). En este ejercicio utilizaremos como criterio la capa de la arquitectura (en concreto modelo, datos, casos de uso y presentación). La organización propuesta se muestra a continuación.

Pero eres libre de usar nombres en inglés o definir tu propio criterio. El paquete donde se encuentra cada clase no afectará a los ejercicios.

Ejercicio: Organizar las clases en paquetes.

1. Pulsa con el botón derecho sobre com.example.mislugares y selecciona *New / Package*. Escribe presentacion, se creará com.example.mislugares.presentacion
2. . Arrastra todas las clases terminadas en Activity a este paquete. El proceso de refactorización se realiza de forma bastante automática. Aunque en algunos casos tendrás que realizar algún pequeño ajuste, como hacer público algún método. Añade también en este paquete fragments y adaptadores.
3. Repite este proceso para los paquetes datos, modelo y casos_uso y organiza el código como en la figura.



Ejercicio: Casos de Uso para lugares

1. Pulsa con el botón derecho sobre com.example.mislugares y selecciona: New / Package. Escribe casos_uso, se creará com.example.mislugares.casos_uso

```

package com.example.mislugares.casos_uso;
import android.app.Activity;
import android.content.Intent;
import com.example.mislugares.datos.RepositorioLugares;

public class CasosUsoLugar
{
    private Activity actividad;
    private RepositorioLugares lugares;
    public CasosUsoLugar(Activity actividad, RepositorioLugares
lugares) {
        this.actividad = actividad;
        this.lugares = lugares;
    }
    // OPERACIONES BÁSICAS
    public void mostrar(int pos) {
        //Intent i = new Intent(actividad, VistaLugarActivity.class);
        //i.putExtra("pos", pos);
        //actividad.startActivity(i);
    }
}

```

2. Dentro de esta clase vamos a añadir diferentes funciones que ejecutarán distintos casos de uso referentes a un lugar. Por ejemplo, compartir un lugar, borrarlo, ... De momento solo añadimos un caso de uso, `mostrar(pos)`, que arrancará una actividad mostrando la información del lugar según su posición en el Repositorio. Se han añadido dos propiedades a la clase. La primera, `actividad`, nos va a permitir extraer el contexto o lanzar otras actividades.
3. Añade en `MainActivity` las siguiente propiedades:

```

private RepositorioLugares lugares;
private CasosUsoLugar usoLugar;

@Override protected void onCreate(Bundle savedInstanceState) {
    ...
    lugares = ((Aplicacion) getApplication()).lugares;
    usoLugar = new CasosUsoLugar(this, lugares);
}

```

Nota: No se cumple la regla de dependencias de la arquitectura Clean, se añade porque va a simplificar el código. La segunda, `lugares`, nos va a permitir acceder al repositorio de los lugares.

4. Cada vez que queramos ejecutar este caso de uso usaremos el código:
`usoLugar.mostrar(pos)`

Práctica: Casos de Uso para arrancar actividades

1. Crea la clase `CasosUsoActividades` dentro del paquete `casos_uso`.
2. Crea la función `lanzarAcerdaDe()`. Ha de contener el código necesario para arrancar `AcerdaDeActivity`.
3. Añade en `MainActivity` el código necesario para usar este caso de uso.

Declara el objeto e instancialo:

Atributo de la clase MainActivity

```
private CasosUsoActividades usoActividades;
```

En onCreate:

```
usoActividades = new CasosUsoActividades(this);
```

Para lanzar la actividad en MainActivity:

```
usoActividades.lanzarAcercaDe();
```

4. En esta clase también se pueden añadir otros casos de uso como lanzarPreferencias() u otras actividades, o abrir las actividades adecuadas.

Una vez que completes la aplicación Mis Lugares las clases para casos de uso acaban conteniendo decenas de métodos. De esta forma, va a ser muy sencillo saber qué hace cada método, donde está y las dependencias que necesita. De lo contrario, todo este código acabaría en las actividades, que contendría cientos de líneas de código, siendo muy difíciles de mantener.

Clase CasosUsoActividades

```
package com.example.mislugares.casos_uso;
import android.content.Intent;
import android.app.Activity;
import com.example.mislugares.presentacion.AcercaDeActivity;//Para Lanzar esta
activity
import com.example.mislugares.presentacion.PreferenciasActivity;//Para Lanzar esta
activity
//Clase con métodos para Lanzar activities
public class CasosUsoActividades
{
    private Activity actividad;
    //Recibe la activity que llama a sus métodos
    public CasosUsoActividades(Activity actividad)
    {
        this.actividad = actividad;
    }
    public void lanzarAcercaDe()
    {
        //La activity que nos pasan como parámetro en el constructor Lanza
        AcercaDeActivity
        actividad.startActivity(new Intent(actividad, AcercaDeActivity.class));
    }
    public void lanzarPreferencias()
    {
        //La activity que nos pasan como parámetro en el constructor Lanza
        PreferenciasActivity
        actividad.startActivity(new Intent(actividad,
        PreferenciasActivity.class));
    }
}
```


Creando actividades en Mis Lugares

Objetivos:

Crear actividades para mostrar y editar información en la aplicación Mis Lugares.

La actividad VistaLugarActivity nos mostrará la información que hemos almacenado de un determinado lugar y nos permitirá realizar una gran cantidad de acciones sobre ese lugar (mostrar en mapa, llamar por teléfono, compartir en redes sociales, etc.). Desde esta actividad podremos cambiar algunos valores de modificación frecuente. En concreto: la valoración, la fecha de visita y la fotografía.



Ejercicio: Creación de la actividad VistaLugarActivity

1. Abre el proyecto *MisLugares*.

2. Descarga <http://www.dcomg.upv.es/~jtomas/android/ficheros/mislugares.zip>

y descomprime en una carpeta. Copia los gráficos que encontrarás en el portapapeles y pegalos dentro de *res/drawable* en el explorador del proyecto.

3. Crea un nuevo *layout* y llámalo *vista_lugar.xml*. Copia el siguiente código para usarlo como base:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <TextView
            android:id="@+id/nombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
        android:layout_gravity="center_vertical"
        android:gravity="center"
        android:text="Nombres del lugar"
        android:textAppearance="?android:attr/textAppearanceLarge" />
<!-- -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <ImageView
        android:id="@+id/logo_tipo"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="logo del tipo"
        android:src="@drawable/otros" />
    <TextView
        android:id="@+id/tipo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="tipo del lugar" />
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    >
    <ImageView
        android:id="@+id/im_direccion"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="dirección"
        android:src="@android:drawable/ic_menu_myplaces" />
    <TextView
        android:id="@+id/direccion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="dirección" />
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    >
    <ImageView
        android:id="@+id/im_telefono"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="telefono"
        android:src="@android:drawable/ic_menu_call" />
    <TextView
        android:id="@+id/telefono"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:textAppearance="?android:attr/textAppearanceMedium"
```

```
        android:text="teléfono" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        >
        <ImageView
            android:id="@+id/im_url"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:contentDescription="url"
            android:src="@android:drawable/ic_menu_mapmode" />
        <TextView
            android:id="@+id/url"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="url" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <ImageView
            android:id="@+id/im_comentario"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:contentDescription="comentario"
            android:src="@android:drawable/ic_menu_info_details" />
        <TextView
            android:id="@+id/comentario"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="comentarios" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <ImageView
            android:id="@+id/im_fecha"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:contentDescription="telefono"
            android:src="@android:drawable/ic_menu_my_calendar" />
        <TextView
            android:id="@+id/fecha"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="fecha" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <ImageView
            android:id="@+id/im_hora"
            android:layout_width="40dp"
            android:layout_height="40dp"
            android:contentDescription="hora"
            android:src="@android:drawable/ic_menu_recent_history" />
        <TextView
            android:id="@+id/hora"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="bottom"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="hora" />
    </LinearLayout>
    <!-- -->
    <RatingBar
        android:id="@+id/valoracion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:rating="3" />
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <ImageView
            android:id="@+id/foto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:adjustViewBounds="true"
            android:contentDescription="fotografía"
            android:src="@drawable/foto_epsg" />
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right" >
            <ImageView
                android:id="@+id/camara"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:contentDescription="logo cámara"
                android:src="@android:drawable/ic_menu_camera" />
            <ImageView
                android:id="@+id/educacion"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:contentDescription="logo educación"
                android:src="@drawable/educacion" />
        </LinearLayout>
    </FrameLayout>
</LinearLayout>
</ScrollView>

```

Observa como el **elemento exterior es un ScrollView**. Esto es conveniente cuando pensemos que los elementos de layout no cabrán en la pantalla. En este caso el usuario podrá desplazar verticalmente el layout arrastrando con el dedo. Dado que **algunas pantallas pueden ser muy pequeñas la mayoría de diseños han de incorporar un ScrollView**.

Dentro de este elemento tenemos un LinearLayout para organizar las vistas verticalmente. La primera vista es un TextView cuyo id es nombre. Se ha asignado un valor para text inicial, que será reemplazado por el nombre del lugar. La única función que tiene este texto inicial es ayudarnos en el diseño. El siguiente elemento es un LinearLayout vertical que contiene un ImageView y un TextView. Este elemento será utilizado para indicar el tipo de lugar.

En el elemento RatingBar podremos introducir una valoración del lugar. El último elemento es un FrameLayout que permite superponer varias vistas. Se dibujarán en el orden en que las indicamos. En el fondo se dibuja un ImageView con una fotografía de la EPSG. El atributo **adjustViewBounds** indica que la imagen sea escalada para ocupar todo el espacio disponible. Sobre la fotografía se dibujará un LinearLayout con dos ImageView. Estos botones permitirán cambiar la fotografía desde la cámara o desde la galería.

4. Para cada elemento haz una copia del <LinearLayout> .

Recurso para ImageView	Id para TextView
@android:drawable/ic_menu_myplaces	@+id/direccion
@android:drawable/ic_menu_call	@+id/telefono
@android:drawable/ic_menu_mapmode	@+id/url
@android:drawable/ic_menu_info_details	@+id/comentario
@android:drawable/ic_menu_my_calendar	@+id/fecha
@android:drawable/ic_menu_recent_history	@+id/hora

5. Crea la clase VistaLugarActivity y reemplaza el código por el siguiente:

```
/**
 * Clase VistaLugarActivity
 * Lee el repositorio de lugares y una posición que apunta a un Lugar
 */
public class VistaLugarActivity extends AppCompatActivity {
    private RepositorioLugares lugares;
    private CasosUsoLugar usoLugar;
    private int pos;
    private Lugar lugar;
    //No tiene constructor, lanza una activity
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Carga el layout vista_lugar
        setContentView(R.layout.vista_lugar);
        //La vista llamadora debe poner datos en la llamada con el valor de "pos"
        Bundle extras = getIntent().getExtras();
        pos = extras.getInt("pos", 0);
    }
}
```

```

//Instancia el repositorio
lugares = ((Aplicacion) getApplication()).lugares;
//Instancia la clase que aporta métodos para manejar la clase Lugar
usoLugar = new CasosUsoLugar(this, lugares);
//Selecciona un Lugar concreto
lugar = lugares.elemento(pos);
actualizaVistas();
}

/**Metodo de
 * Rellena los datos de una vista en concreto La apuntada por "pos"
 */
public void actualizaVistas()
{
    TextView nombre = findViewById(R.id.nombre);
    nombre.setText(lugar.getNombre());
    ImageView logo_tipo = findViewById(R.id.logo_tipo);
    logo_tipo.setImageResource(lugar.getTipo().getRecurso());
    TextView tipo = findViewById(R.id.tipo);
    tipo.setText(lugar.getTipo().getTexto());
    TextView direccion = findViewById(R.id.direccion);
    direccion.setText(lugar.getDireccion());
    TextView telefono = findViewById(R.id.telefono);
    telefono.setText(Integer.toString(lugar.getTelefono()));
    TextView url = findViewById(R.id.url);
    url.setText(lugar.getUrl());
    TextView comentario = findViewById(R.id.comentario);
    comentario.setText(lugar.getComentario());
    TextView fecha = findViewById(R.id.fecha);
    fecha.setText(DateFormat.getDateInstance().format(
        new Date(lugar.getFecha())));
    TextView hora = findViewById(R.id.hora);
    hora.setText(DateFormat.getTimeInstance().format(
        new Date(lugar.getFecha())));
    //Apunta al RatingBar con "valoracion"
    RatingBar valoracion = findViewById(R.id.valoracion);
    valoracion.setRating(lugar.getValoracion());
    //Crea una función listener para el ratingBar "valoracion"
    valoracion.setOnRatingBarChangeListener
    (
        new RatingBar.OnRatingBarChangeListener()
        {//Vamos a utilizar la función onRatingChanged
            @Override public void onRatingChanged(RatingBar ratingBar, float valor,
boolean fromUser)
            {
                lugar.setValoracion(valor);
            }
        });
}
}

```

Nota: Puedes pulsar **Alt+Intro** para que automáticamente se añadan los imports con los paquetes que faltan. Dos clases aparecen en varios paquetes, selecciona `java.text.DateFormat` y `java.util.Date`.

Se definen tres variables globales para que se pueda acceder a ellas desde cualquier método de la clase: lugares corresponde con el repositorio de lugares, cuya referencia se obtiene desde Application, pos es la posición del elemento que vamos a visualizar y lugar es el elemento en sí.

El método `onCreate()` será ejecutado cuando se cree la actividad y en él tenemos que asociar un layout (`setContentView(R.layout.vista_lugar)`) e inicializar todos sus valores. A continuación, se averigua la posición del lugar a mostrar, que ha sido pasado en un *extra*. A partir de este id obtenemos el objeto Lugar a mostrar.

Observa cómo **se obtiene un objeto de cada uno de los elementos de la vista utilizando el método `findViewById()`**. A continuación este **objeto es modificado según el valor del lugar que estamos representando**. Al final se realiza una acción especial con el **objeto `valoracion`, utilizando el método `setOnRatingBarChangeListener()` para asignarle un escuchador de eventos al RatingBar que es creado allí mismo**. Este escuchador de evento se **activará cuando el usuario modifique la valoración**. El código a ejecutar consiste en modificar la propiedad `Valoracion()` del objeto lugar con la nueva valoración.

6. Abre la clase `TipoLugar` y asigna un recurso *drawable* a cada tipo de lugar. Puedes utilizar la opción de autocompletar, es decir, escribe `R.drawable.` y espera a que el sistema te dé una alternativa.

```
public enum TipoLugar {  
    OTROS ("Otros", R.drawable.otros),  
    RESTAURANTE ("Restaurante", R.drawable.restaurante),  
    BAR("Bar", R.drawable.bar),  
    ...
```

7. Añade en `MainActivity` el siguiente método:

```
public void lanzarVistaLugar(View view)  
{  
    usoLugar.mostrar(0);  
}
```

Este método lanzará la actividad `VistaLugarActivity` pasándole como posición del lugar a visualizar siempre 0. Más adelante mostraremos algunas alternativas para que el usuario pueda seleccionar el lugar a mostrar.

8. En el método `onOptionsItemSelected()` de la actividad `MainActivity` añade el siguiente código:

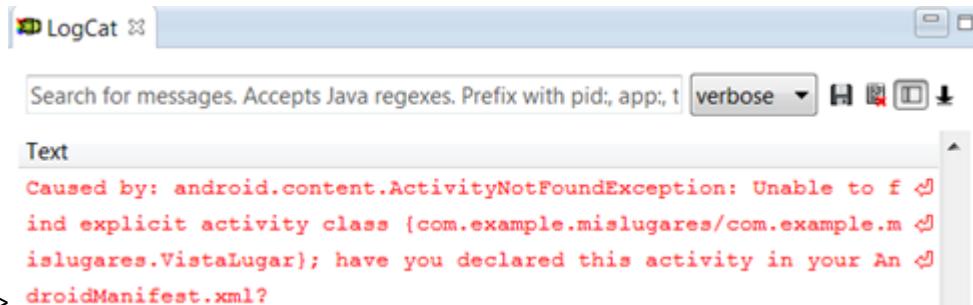
```
@Override public boolean onOptionsItemSelected(MenuItem item) //Recibe la  
selección del menú  
{  
    //Lee el id del ítem del menú pulsado  
    int id = item.getItemId();  
    ...  
    if (id == R.id.menu_buscar)  
    {  
        lanzarVistaLugar(null);  
        return true;  
    }  
    ...
```

Y descomenta el método de `CasosUsoLugar`

```
public void mostrar(int pos)  
{  
    Intent i = new Intent(actividad, VistaLugarActivity.class);  
    i.putExtra("pos", pos);  
    actividad.startActivity(i);
```

}

9. Ejecuta la aplicación. Aparecerá un error cuando selecciones Buscar. Siempre que aparezca un error en ejecución es el momento de visualizar el *LogCat*. No es sencillo analizar la información que se muestra, pero es muy importante que te acostumbres a buscar la causa del problema en el *LogCat*. En este caso la información clave se muestra a continuación:



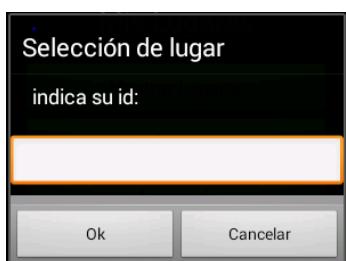
The screenshot shows the Android LogCat window. The title bar says "LogCat". Below it is a search bar with placeholder text "Search for messages. Accepts Java regexes. Prefix with pid:, app:, t:". To the right of the search bar are buttons for "verbose" (set to "verbose"), "H" (highlight), and a download icon. The main area is labeled "Text" and contains a red error message:
Caused by: android.content.ActivityNotFoundException: Unable to find explicit activity class {com.example.mislugares/com.example.mislugares.VistaLugar}; have you declared this activity in your AndroidManifest.xml?

10. Para resolver el error en ejecución registra la nueva actividad en *AndroidManifest.xml*
11. Ejecuta la aplicación y verifica que cuando seleccionas el ícono *buscar* se arranca una actividad que muestra el primer lugar.

Pop-up mediante la clase AlertDialog

Ejercicio: Un cuadro de dialogo para indicar el id de lugar

Tras realizar el ejercicio anterior comprobarás que siempre se visualiza el lugar con posición 0. En este ejercicio vamos a introducir un cuadro de dialogo que permita introducir al usuario el id que desea visualizar.



Ha de quedar claro que esta es la forma más correcta de diseñar el interfaz de usuario. En la siguiente unidad reemplazaremos este cuadro de dialogo por un RecyclerView.

1. Abre la clase MainActivity del proyecto *MisLugares*.
2. Reemplaza el método por el lanzar *LanzaVistaLugar()* siguiente:

```
public void lanzarVistaLugar(View view)
{
    //Crea una vista EditText
    final EditText entrada = new EditText(this);
    entrada.setText("0");
    //Crea un pop-up AlertDialog
    new AlertDialog.Builder(this)
        .setTitle("Selección de lugar")
        .setMessage("indica su id:")
        .setView(entrada)
        .setPositiveButton("Ok", new DialogInterface.OnClickListener()
    {
```

```

        public void onClick(DialogInterface dialog, int whichButton)
    {
        int id = Integer.parseInt (entrada.getText().toString());
        //Aquí habría que poner un filtro para evitar un id incorrecto
        usoLugar.mostrar(id);
    }
}
//setPositiveButton
.setNegativeButton("Cancelar", null)
.show();
}
}

```

Nota En Java es posible crear un objeto sin que este disponga de un identificador de objeto. Este tipo de objeto se conoce como objeto anónimo. El código mostrado a continuación a la derecha es equivalente al de la izquierda.

```
Clase objeto = new Clase();           new Clase().metodo();
objeto.metodo();
```

Objetos anónimos

Un objeto anónimo no tiene identificador por lo que solo puede ser usado donde se crea. En el método anterior se ha creado un objeto anónimo de la clase AlertDialog.Builder. Observa cómo no se llama a un método, sino a una cadena de métodos. Esto es posible porque los métodos de la clase AlertDialog.Builder *retornan el objeto que estamos creando. Por lo tanto, cada método es aplicado al objeto devuelto por el método anterior.*

En Android puedes usar la clase AlertDialog para crear un cuadro de diálogo configurable. Si te fijas en la captura anterior están formados por cuatro elementos, de arriba abajo: título, mensaje, vista y botones. Estos elementos pueden ser configurados mediante los métodos setTitle(), setMessage(), setView(), setPositiveButton() y setNegativeButton().

La vista que se utiliza en este diálogo es un EditText, inicializado con un texto. En caso de necesitar varias entradas se puede crear una vista de tipo layout, que contendría estas entradas. Se han introducido dos botones, indicando el texto del botón y un escuchador de evento que será llamado cuando se pulse el botón. Finalmente se llama al método show() para que se visualice el cuadro de diálogo.

3. Verifica que funciona correctamente. Pero cuidado, no se verifica que el id sea válido, por lo que ocurrirá un error si es incorrecto.

Ocultar o mostrar un View

Práctica: Ocultar elementos en VistaLugarActivity

En ocasiones no se dispondrá de parte la información de un lugar. En estos casos, puede resultar más conveniente desde un punto de vista estético, no mostrar campos sin información en VistaLugarActivity. Por ejemplo, si el campo de teléfono es igual a 0, podríamos usar el siguiente código para que no se muestre:

```

...
if (lugar.getTelefono() == 0)
{
    findViewById(R.id.telefono).setVisibility(View.GONE);
} else
{
    findViewById(R.id.telefono).setVisibility(View.VISIBLE);
}

```

```

        TextView telefono = findViewById(R.id.telefono);
        telefono.setText(Integer.toString(lugar.getTelefono()));
    }
    // url
    if (lugar.getUrl().isEmpty())
    {
        findViewById(R.id.url).setVisibility(View.GONE);
    } else
    {
        findViewById(R.id.url).setVisibility(View.VISIBLE);
        TextView url = findViewById(R.id.url);
        url.setText(lugar.getUrl());
    }
    ...

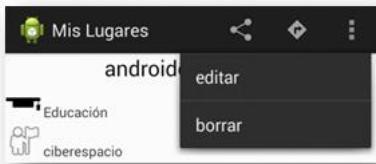
```

Para ocultarlo, en el *layout* telefono, ponemos el valor propiedad visibility al valor GONE. Este atributo es aplicado a cualquier tipo de vista. Otros posibles valores para este atributo son VISIBLE e INVISIBLE. Tanto con GONE como con INVISIBLE la vista no se verá. Pero con INVISIBLE el espacio ocupado por la vista se mantiene, mientras que con GONE este espacio se elimina.

Trata de realizar un proceso similar a este para los campos dirección, telefono, url y comentario. Para verificar si un String es vacío puedes usar el método isEmpty().

Ejercicio: Añadir una barra de acciones a VistaLugarActivity

En este ejercicio vamos a añadir a la actividad un menú a la barra de acciones similar al que se muestra a continuación:



1. En primer lugar crea el fichero *res/menu/vista_lugar.xml* que contendrá las acciones a mostrar. Para ello pulsa con el botón derecho sobre la carpeta *res/menu* y crea el fichero *vista_lugar*.

2. Reemplaza su contenido por el siguiente código:

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/accion_compartir"
        android:title="compartir"
        android:icon="@android:drawable/ic_menu_share"
        android:orderInCategory="10"
        app:showAsAction="ifRoom"/>
    <item
        android:id="@+id/accion_llegar"
        android:title="cómo llegar"
        android:icon="@android:drawable/ic_menu_directions"
        android:orderInCategory="20"

```

```
        app:showAsAction="ifRoom"/>
<item
    android:id="@+id/accion_editar"
    android:title="editar"
    android:icon="@android:drawable/ic_menu_edit"
    android:orderInCategory="30"
    app:showAsAction="ifRoom"/>
<item
    android:id="@+id/accion_borrar"
    android:title="borrar"
    android:icon="@android:drawable/ic_menu_delete"
    android:orderInCategory="40"
    app:showAsAction="ifRoom"/>
</menu>
```

3. En la clase VistaLugarActivity añade los siguientes métodos:

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.vista_lugar, menu);
    return true;
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.accion_compartir:
            return true;
        case R.id.accion_llegar:
            return true;
        case R.id.accion_editar:
            return true;
        case R.id.accion_borrar:
            usoLugar.borrar(pos);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

4. Añade a CasosUsoLugares:

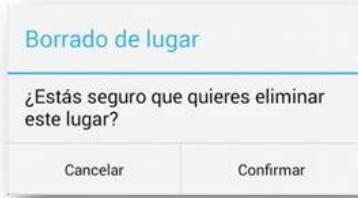
```
public void borrar(final int id) {
    lugares.borrar(id);
    actividad.finish();
}
```

5. Ejecuta la aplicación y borra un lugar. Verifica que, si tratas de visualizar el mismo *id* ahora se muestra el siguiente lugar.

Práctica: Un cuadro de diálogo para confirmar el borrado

Un usuario podría pulsar por error el botón de borrar, por lo que sería muy conveniente pedir una confirmación antes de borrar.

1. En el método crea un cuadro de dialogo siguiendo el esquema planteado en el ejercicio anterior. Puede ser similar al siguiente.



Creando la actividad EdicionLugarActivity

En este apartado crearemos otra actividad en la aplicación Mis Lugares, `EdicionLugarActivity`. Esta actividad nos permitirá modificar la mayoría de valores asignados a un lugar (se excluyen los valores que se modifican desde `VistaLugarActivity`: Valoración, fecha y foto):



Práctica: Creación de la actividad EdicionLugarActivity

1. En el proyecto *MisLugares* verifica que existe el layout `edicion_lugar.xml`. En caso contrario crea el layout ahora.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.core.widget.NestedScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/relativeLayout">
        <TextView
            android:id="@+id/t_nombre"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nombre:"
            android:textAppearance="?android:attr/textAppearanceMedium"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"/>
        <EditText
            android:id="@+id/nombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
</androidx.core.widget.NestedScrollView>
```

```

        android:hint="algo que identifique el lugar"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_nombre"
        android:layout_marginStart="8dp">
        <requestFocus/>
    </EditText>
    <TextView
        android:id="@+id/t_tipo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tipo:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nombre"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <Spinner
        android:id="@+id/t_tipo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="@+id/t_tipo"
        app:layout_constraintLeft_toRightOf="@+id/t_tipo"/>
    <TextView
        android:id="@+id/t_direccion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dirección:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_tipo"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/direccion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="dirección del lugar"
        android:inputType="textPostalAddress"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_direccion"
        android:layout_marginStart="8dp"/>
    <TextView
        android:id="@+id/t_telefono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Teléfono:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/direccion"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/telefono"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="teléfono para contactar"
        android:inputType="phone"
        app:layout_constraintLeft_toRightOf="@+id/t_telefono"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginEnd="8dp"
        app:layout_constraintStart_toEndOf="@+id/t_telefono"
        android:layout_marginStart="8dp"
        app:layout_constraintBaseline_toBaselineOf="@+id/t_telefono"/>
    <TextView
        android:id="@+id/t_url"
        android:layout_width="wrap_content"

```

```

        android:layout_height="wrap_content"
        android:text="Url:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/telefono"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/url"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="página web"
        android:inputType="textUri"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_url"
        android:layout_marginStart="8dp"/>
    <TextView
        android:id="@+id/t_comentario"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comentario:"
        android:textAppearance="?android:attr/textAppearanceMedium"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/url"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"/>
    <EditText
        android:id="@+id/comentario"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="introduce tus notas"
        android:inputType="textMultiLine"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/t_comentario"
        android:layout_marginStart="8dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
</androidx.core.widget.NestedScrollView>
```

2. Crea la clase EdicionLugarActivity y haz que extienda AppCompatActivity. Copia en esta clase los atributos y los método onCreate() y actualizaVistas() de la clase VistaLugarActivity.

```

@Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //Carga el layout vista_lugar
    setContentView(R.layout.edicion_lugar);
    //La vista Llamadora me envía la posición en "pos"
    Bundle extras = getIntent().getExtras();
    pos = extras.getInt("pos", 0);
    //Instancia el repositorio
    lugares = ((Aplicacion) getApplication()).lugares;
    //Instancia la clase que aporta métodos para manejar la clase Lugar
    usoLugar = new CasosUsoLugar(this, lugares);
    //Leo los datos del lugar concreto
    lugar = lugares.elemento(pos);
    //Instancia el spinner
    tipo = findViewById(R.id.tipo);
    //adaptador es un array que permite acceder al enum tipo Lugar y
    //leer los enumerados
    actualizaVistas();
}

public void actualizaVistas()
{
```

```
//Nombre
nombre = findViewById(R.id.nombre);
nombre.setText(lugar.getNombre());
//Dirección
dirección = findViewById(R.id.dirección);
dirección.setText(lugar.getDirección());
//Teléfono
teléfono = findViewById(R.id.teléfono);
teléfono.setText(Integer.toString(lugar.getTeléfono()));
// url
url = findViewById(R.id.url);
url.setText(lugar.getUrl());
// comentario
comentario = findViewById(R.id.comentario);
comentario.setText(lugar.getComentario());
}
```

3. En Java añade los siguientes atributos a la clase:

```
private EditText nombre;
private Spinner tipo;
private EditText dirección;
private EditText teléfono;
private EditText url;
private EditText comentario;
```

De esta forma estos seis objetos serán accesibles desde cualquier método de la clase, en lugar de estar declarados solo en el método onCreate().

4. Como ves se reemplaza la vista a mostrar en setContentView() por edicion_lugar.
5. El paso de parámetros para obtener pos y lugar se realiza de la misma forma.
6. Crea un caso de uso, con la función editar(pos) que abra la actividad que acabas de crear. Usa mostrar(pos) como referencia.
7. En la clase VistaLugarActivity dentro del método onOptionsItemSelected(), añade el código necesario para que se abra la actividad que acabas de crear se llame a esta función.
8. Ejecuta el proyecto. Pero antes, piensa si falta alguna acción por realizar (como actualizar el manifest con la nueva clase, ya que carta una actividad).

Uso del spinner con ArrayAdapter

Ejercicio: Inicializar el Spinner en EdicionLugarActivity

Como has podido verificar en la ejecución anterior el Spinner (lista desplegable) no muestra ningún valor. En este ejercicio trataremos que funcione adecuadamente:



1. Añade el siguiente código el método onCreate():

```
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
```

```
android.R.layout.simple_spinner_item, TipoLugar.getNombres());
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
tipo.setAdapter(adaptador); //Pon los datos del adaptador en el spinner
tipo.setSelection(lugar.getTipo().ordinal()); //Muestra el item de Lugar
```

Para inicializar los valores que puede tomar un Spinner necesitamos una clase especial conocida como Adapter. Esta clase se estudiará en la siguiente unidad. De momento solo adelantamos que un Adapter va a crear una lista de vistas, inicializándolas con unos valores determinados. La clase `ArrayAdapter<String>` es un tipo de Adapter que permite inicializar sus valores a partir de un array de String. Su constructor necesita tres parámetros: un contexto (usamos la actividad actual), una vista para mostrar elemento (usamos un vista definida en el sistema) y un array de String. Para el último parámetro necesitamos un array con todos los valores que puede tomar el enumerado `TipoLugar`. Para obtener este array se define un nuevo método que se muestra a continuación.

El siguiente método, `setDropDownViewResource()`, permite indicar una vista alternativa que se usará cuando se despliegue el Spinner. En la versión 4.x esta vista es un poco más grande que la usada en el método anterior para poder seleccionarla cómodamente con el dedo. En la versión 2.x muestra círculos seleccionables a la derecha de cada elemento. Este código concluye asignando el adaptador al Spinner y poniendo un valor inicial según el tipo actual de lugar.

2. Añade el siguiente método a la clase `TipoLugar`:

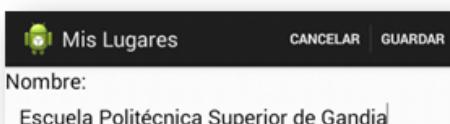
```
public static String[] getNombres()
{
    //Array de strings con tantos strings como elementos del enum
    String[] resultado = new String[TipoLugar.values().length];
    for (TipoLugar tipo : TipoLugar.values())//values es método static de clase enum
        {//para cada item del enum lo copia en tipo y pone el campo texto en el array
    //result

        resultado[tipo.ordinal()] = tipo.getTexto();
    }
    return resultado;
}
```

3. Ejecuta la aplicación y verifica que la lista desplegable funciona correctamente..

Práctica: Añadir una barra de acciones a EdicionLugarActivity

En esta práctica vamos a añadir a la actividad un menú en la barra de acciones similar al que se muestra a continuación:



1. Crea un nuevo recurso de menú con las opciones que se indican.
2. Asocia este menú a la actividad `EdicionLugarActivity` con el método `onCreateOptionsMenu()`.

3. Crea el método onOptionsItemSelected() de manera que cuando se seleccione la acción *Guardar* se ejecute el siguiente código:

```
lugar.setNombre(nombre.getText().toString());
lugar.setTipo(TipoLugar.values()[tipo.getSelectedItemPosition()]);
lugar.setDireccion(direccion.getText().toString());
lugar.setTelefono(Integer.parseInt(telefono.getText().toString()));
lugar.setUrl(url.getText().toString());
lugar.setComentario(comentario.getText().toString());
usoLugar.guardar(pos, lugar);
finish();
```

4. Cuando se seleccione la acción cancelar, simplemente se saldrá de la actividad.

Añade a CasosLugar la siguiente función:

```
public void guardar(int id, Lugar lugarCambiado)
{
    lugares.actualiza(id, lugarCambiado);
}
```

5. Ejecuta la aplicación. Modifica algún lugar y pulsa en *Guardar*. Al regresar a la actividad anterior los valores permanecen sin variación. Sin embargo si pulsas la tecla volver y entras a visualizar el mismo lugar, los cambios sí que son actualizados. ¿Qué puede estar pasando?

Ejercicio: Refrescar VistaLugarActivity tras entrar en EdicionLugarActivity

Parece que al regresar a VistaLugarActivity desde EdicionLugarActivity no estamos indicando que vuelva a obtener los datos mostrados en las vistas. Para actualizar estos valores puedes hacer los siguientes pasos:

1. Cambia un poco la función en CasosUsoLugar, para que

```
public void editar(int pos, int codidoSolicitud)
{
    Intent i = new Intent(actividad, EdicionLugarActivity.class);
    i.putExtra("pos", pos);
    actividad.startActivityForResult(i, codidoSolicitud);
}
```

2. Añade la siguiente constante a VistaLugarActivity

```
final static int RESULTADO_EDITAR = 1;
```

3. En el método onOptionsItemSelected() añade el nuevo parámetro.
4. Añade el siguiente método: VistaLugarActivity

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == RESULTADO_EDITAR) {
        actualizaVistas();
        findViewById(R.id.scrollView1).invalidate();
```

```
    }  
}
```

Una vez regresamos de la actividad EdicionLugarActivity lo que hacemos es actualizar los valores de las vistas y forzar al sistema a que repinte la vista con id scrollView1. Esta vista corresponde al ScrollView que contiene todo el *layout*.

Unidad 5 :RecyclerView e Intenciones

Introducción a la unidad

Comenzamos la unidad introduciendo el uso de RecyclerView, un elemento de gran importancia en el diseño de una interfaz de usuario en Android. Nos va a permitir definir una lista de elementos personalizados.

En esta unidad también trataremos las intenciones. Son una herramienta fundamental en Android, que nos permite lanzar componentes para realizar tareas en nuestra aplicación. Estos componentes podrán ser parte de nuestra aplicación, por ejemplo como hicimos en la unidad anterior, para arrancar nuevas actividades. También nos van a permitir lanzar componentes de otras aplicaciones, o del sistema, para realizar tareas como tomar una fotografía o realizar una llamada de teléfono.

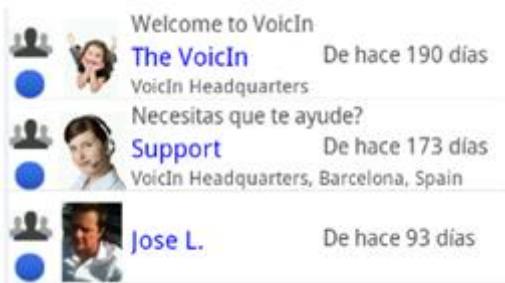
Terminaremos la unidad añadiendo a la aplicación Mis Lugares la capacidad de trabajar con fotografías y con fechas.

Objetivos:

- RecyclerView.
- Describir el uso de intenciones para invocar actividades estándar en Android.
- Enumerar las intenciones más útiles para realizar acciones estándar en Android.
- Aprender a manipular fichero con fotografías y mostrarlos en un ImageView.
- Mostrar el uso de ventanas de dialogo para seleccionar fechas.

La vista RecyclerView

La vista RecyclerView visualiza una lista o cuadrícula deslizable de varios elementos, donde cada elemento puede definirse mediante un layout. Su utilización es algo compleja, pero muy potente. Un ejemplo lo podemos ver en la siguiente figura:



Dentro del API de Android encontramos las vistas **ListView** y **GridView** nos ofrece una alternativa a **RecyclerView**. Esta última no ha sido añadida a ningún API si no que se añade en una librería de compatibilidad. A pesar de que resulta algo más compleja de manejar, recomendamos el uso de **RecyclerView**, en lugar de ListView o GridView, al ser más eficiente y flexible. Aunque su uso se describe con detalle en El Gran Libro de Android Avanzado, hacemos en este punto una introducción de sus funcionalidades básicas. Las principales ventajas que ofrece RecyclerView frente a ListView o GridView son:

- Reciclado de vistas (RecyclerView.ViewHolder)
- Distribución de vistas configurable (LayoutManager)
- Animaciones automáticas (ItemAnimator)
- Separadores de elementos (ItemDecoration)
- Trabaja conjuntamente con otros widgets introducidos en Material Design (CoordinatorLayout)

video [Creación de listas con RecyclerView.](#)

Crear una lista (o cuadrícula) de elementos con un RecyclerView conlleva los siguientes pasos:

- Diseñar un Layout que contiene el RecyclerView.
- Implementar la actividad que visualice el RecyclerView
- Diseñar un Layout individual con una vista que se repetirá en la lista
- Personalizar cada una de los *Layouts* individuales según nuestros datos utilizando un adaptador.
- Definir como queremos que se posicen los elementos en las vistas. Por ejemplo en forma de lista o de cuadrícula.

Los tres primeros pasos anteriores son similares al uso de cualquier otro tipo de vista. Los dos últimos sí que requieren una explicación más extensa:

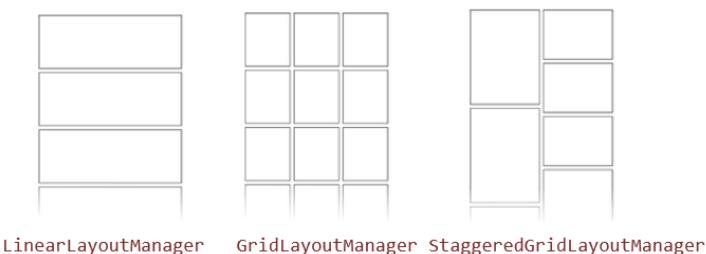
Personalizar los datos a mostrar

Para **personalizar los elementos a mostrar en un RecyclerView hemos de usar un adaptador**. La creación de adaptadores puede ser delicada, en algunos casos podemos tener problemas de eficiencia. Para evitar estos problemas, Google ha cambiado la forma de trabajar con RecyclerView. Ya no se puede utilizar la interfaz Adapter, si no que **se ha de utilizar la clase RecyclerView.Adapter**.

Video [El patrón ViewHolder y su uso en un RecyclerView.](#)

Distribuir los elementos

A diferencia ListView o GridView, que muestran los elementos usando una determinada configuración, **RecyclerView puede configurar esta distribución por medio de la clase LayoutManager**. El sistema nos proporciona tres descendientes de LayoutManager, que son mostrados en la siguiente figura. También podemos crear nuestro descendiente de LayoutManager.



En los siguientes ejercicios usaremos un RecyclerView en Mis Lugares. La actividad inicial de la aplicación nos permite escoger entre cuatro botones. En una aplicación como la desarrollada, sería mucho más interesante que en esta actividad se visualizara directamente una lista con los lugares almacenados.



Ejercicio paso a paso: Un RecyclerView en Mis Lugares

1. Añade al fichero Gradle Scripts/Bulid.gradle (Module:app) la siguiente dependencia

```
dependencies {  
    ...  
}
```

```
        implementation 'androidx.recyclerview:recyclerview:1.0.0'  
    }
```

NOTA: Si lo deseas, puedes saltarte este paso. Más adelante, cuando aparezca RecyclerView en el código Java, la clase aparecerá marcada en rojo, al no encontrar su declaración. Al pulsar sobre la clase, aparecerá una bombilla roja donde podrás elegir la opción Add dependency on androidx.recyclerview:recyclerview. El mismo añadirá la dependencia, con la ventaja de seleccionar la última versión disponible.

La clase ReciclerView no ha sido añadida al API de Android, si no que se encuentra en una librería externa. Esto tiene la gran ventaja de que aunque esta clase aparece en la versión 5 de Android, puede ser usada en versiones anteriores.

2. Reemplaza en el *layout content_main.xml* por el siguiente código

```
<androidx.recyclerview.widget.RecyclerView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    android:id="@+id/recycler_view"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

3. En la práctica “*Recursos alternativos en Mis Lugares*” se crea un recurso alternativo para este *layout* en *res/layout-land/content_main.xml*. Elimina este recurso alternativo.
NOTA: Al borrarlo has de desactivar la opción *Safe delete*.

4. En la actividad MainActivity añade el código subrayado:

```
public class MainActivity extends AppCompatActivity {  
    ...  
    private RecyclerView recyclerView;  
    public AdaptadorLugares adaptador;  
  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        ...  
        adaptador = ((Aplicacion) getApplication()).adaptador;  
        recyclerView = findViewById(R.id.recyclerView);  
        recyclerView.setHasFixedSize(true);  
        recyclerView.setLayoutManager(new LinearLayoutManager(this));  
        recyclerView.setAdapter(adaptador);  
    }  
    ...
```

En Java declaramos recyclerView y lo inicializamos con findViewById(). Creamos un adaptador y se lo asignamos al RecyclerView. La clase AdaptadorLugar será definida a continuación. Además, indicamos que las vistas a mostrar serán de tamaño fijo y que usaremos un LayoutManager de tipo LinearLayoutManager.

5. De ser necesario, elimina del método onCreate() el código destinado a inicializar los botones. Los botones van a ser reemplazados por el RecyclerView.

6. En la clase Aplicacion crea la variable adaptador:

```
public AdaptadorLugares adaptador = new AdaptadorLugares(lugares);
```

7. Ahora hemos de definir el layout que representará cada uno de los elementos de la lista. Crea el fichero res/layout/elemento_lista.xml con el siguiente código:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="4dp">
    <ImageView android:id="@+id/foto"
        android:layout_width="?android:attr/listPreferredItemHeight"
        android:layout_height="?android:attr/listPreferredItemHeight"
        android:contentDescription="fotografía"
        android:src="@drawable/bar"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"/>
    <TextView android:id="@+id/nombre"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Nombres del lugar"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textStyle="bold"
        android:maxLines="1"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toEndOf="@+id/foto"
        app:layout_constraintEnd_toEndOf="parent"/>
    <TextView android:id="@+id/direccion"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:maxLines="1"
        android:text="dirección del lugar"
        app:layout_constraintTop_toBottomOf="@+id/nombre"
        app:layout_constraintStart_toEndOf="@+id/foto"
        app:layout_constraintEnd_toEndOf="parent"/>
    <RatingBar android:id="@+id/valoracion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="?android:attr/ratingBarStyleSmall"
        android:isIndicator="true"
        android:rating="3"
        app:layout_constraintTop_toBottomOf="@+id/direccion"
        app:layout_constraintLeft_toRightOf="@+id/foto"
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Para combinar las vistas se ha escogido un ConstraintLayout. El primer elemento que contiene es un ImageView alineado a la izquierda. Su altura se establece a partir de un parámetro de configuración del sistema ?android:attr/listPreferredItemHeight (altura preferida para ítem de lista). Su anchura es la misma, por lo tanto, la imagen será cuadrada. A la derecha se muestran dos textos. En el texto de mayor tamaño se visualizará el nombre del lugar y en el de menor tamaño, la dirección. Bajo estos textos se ha incluido un RatingBar.

8. El siguiente paso será crear la clase AdaptadorLugares, que se encargará de rellenar el ReciclwView.

```
public class AdaptadorLugares extends
    RecyclerView.Adapter<AdaptadorLugares.ViewHolder> {
    protected RepositorioLugares lugares;           // Lista de lugares a mostrar
    public AdaptadorLugares(RepositorioLugares lugares) {
        this.lugares = lugares;
```

```
}

//Creamos nuestro ViewHolder, con los tipos de elementos a modificar

public static class ViewHolder extends RecyclerView.ViewHolder {
    public TextView nombre, direccion;
    public ImageView foto;
    public RatingBar valoracion;
    public ViewHolder(View itemView) {
        super(itemView);
        nombre = itemView.findViewById(R.id.nombre);
        direccion = itemView.findViewById(R.id.direccion);
        foto = itemView.findViewById(R.id.foto);
        valoracion= itemView.findViewById(R.id.valoracion);
    }
    // Personalizamos un ViewHolder a partir de un lugar
    public void personaliza(Lugar lugar) {
        nombre.setText(lugar.getNombre());
        direccion.setText(lugar.getDireccion());
        int id = R.drawable.otros;
        switch(lugar.getTipo()) {
            case RESTAURANTE:id = R.drawable.restaurant; break;
            case BAR:    id = R.drawable.bar;      break;
            case COPAS:   id = R.drawable.copas;    break;
            case ESPECTACULO:id = R.drawable.espectaculos; break;
            case HOTEL:   id = R.drawable.hotel;    break;
            case COMPRAS: id = R.drawable.compras;  break;
            case EDUCACION: id = R.drawable.educacion; break;
            case DEPORTE: id = R.drawable.deporte;  break;
            case NATURALEZA: id = R.drawable.naturaleza; break;
            case GASOLINERA: id = R.drawable.gasolinera; break; }
        foto.setImageResource(id);
        foto.setScaleType(ImageView.ScaleType.FIT_END);
        valoracion.setRating(lugar.getValoracion());
    }
}

// Creamos el ViewHolder con la vista de un elemento sin personalizar

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    // Inflamos la vista desde el xml
    View v = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.elemento_lista, parent, false);
    return new ViewHolder(v);
}

// Usando como base el ViewHolder y lo personalizamos
@Override
public void onBindViewHolder(ViewHolder holder, int posicion) {
    Lugar lugar = lugares.elemento(posicion);
    holder.personaliza(lugar);
}
// Indicamos el número de elementos de la lista
@Override public int getItemCount() {
    return lugares.tamanyo();
}
}
```

Un adaptador es un mecanismo estándar en Android que nos permite crear una serie de vistas que han de ser mostradas dentro de un contenedor. Con las vistas ListView, GridView, Spinner o Gallery has de crear el adaptador utilizando la interfaz Adapter. Pero con RecyclerView has de utilizar la clase RecyclerView.Adapter.

En el constructor se inicializa el conjunto de datos a mostrar (en el ejemplo lugares) y otras variables globales a la clase. El objeto inflador nos va a permitir crear una vista a partir de su XML.

Luego se crea la clase ViewHolder, que contendrá las vistas que queremos modificar de un elemento (en concreto: dos TextView con el nombre y la dirección, un ImageView con la imagen del tipo de lugar y un RatingBar). Esta clase es utilizada para evitar tener que crear las vistas de cada elemento desde cero. Lo va a hacer es utilizar un ViewHolder que contendrá las cuatro vistas ya creadas, pero sin personalizar. De forma que, gastará el mismo ViewHolder para todos los elementos y simplemente lo personalizaremos según la posición. Es decir, reciclamos el ViewHolder. Esta forma de proceder mejora el rendimiento del Recycler View, haciendo que funcione más rápido.

El método onCreateViewHolder() devuelve una vista de un elemento sin personalizar. Podríamos definir diferentes vistas para diferentes tipos de elementos utilizando el parámetro viewType. Usamos el método inflate() para crear una vista a partir del layout XML definido en elemento_lista. En este método se indica como segundo parámetro el layout padre que contendrá a la vista que se va a crear. En este caso, resulta imprescindible indicarlo, ya que queremos que la vista hijo ha de adaptarse al tamaño del padre (en elemento_lista se ha indicado layout_width="match_parent"). El tercer parámetro del método permite indicar si queremos que la vista sea insertada en el padre. Indicamos false, dado que esta operación la va a hacer el Recycler View.

El método onBindViewHolder() personaliza un elemento de tipo ViewHolder según su posición. A partir del ViewHolder que personalizamos ya es el sistema quien se encarga de crear la vista definitiva que será insertada en el Recycler View. Finalmente, el método getItemCount() se utiliza para indicar el número de elementos a visualizar.

9. Ejecuta la aplicación y verifica el resultado.

Ejercicio paso a paso: Selección de un elemento en un RecyclerView

En este ejercicio veremos cómo detectar que se ha pulsado sobre uno de los elementos del RecyclerView. En las vistas ListView y GridView podíamos realizar esta tarea usando el método setOnItemClickListener(). Sin embargo, en RecyclerView no se ha incluido este método. Google prefiere que asignemos un escuchador de forma independiente a cada una de las vistas que va a contener RecyclerView. Existen muchas alternativas para hacer este trabajo ([Escuchadores de eventos y manejadores de eventos en Android](#)). A continuación, explicamos una de ellas:

1. En Java añade a la clase AdaptadorLugares la siguiente declaración:

```
protected View.OnClickListener onClickListener;
```

Para poder modificar el campo anterior añade el siguiente setter:

```
public void setOnItemClickListener(View.OnClickListener onClickListener) {  
    this.onClickListener = onClickListener;  
}
```

- 2.** Solo nos queda aplicar este escuchador a cada una de las vistas creadas. Añade la línea subrayada en el método indicado:

```
@Override  
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
    // Inflamos la vista desde el xml  
    View v = inflador.inflate(R.layout.elemento_lista, null);  
v.setOnClickListener(onClickListener);  
    return new ViewHolder(v);  
}
```

- 3.** Desde la clase MainActivity vamos a asignar un escuchador. Para ello añade el siguiente código al final del método onCreate():

```
adaptador.setOnItemClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        int pos = recyclerView.getChildAdapterPosition(v);  
        usoLugar.mostrar(pos);  
    }  
});
```

El método getChildAdapterPosition(), nos indicarán la posición de una vista dentro del adaptador.

- 4.** Ejecuta la aplicación y verifica el resultado.

Las intenciones en Android

Objetivos:

Estudiar el abanico de posibilidades que nos ofrecen las intenciones en Android.

Una intención representa la voluntad de realizar alguna acción o tarea, como realizar una llamada de teléfono o visualizar una página web. Una intención nos permite lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente. Tienen un gran potencial en Android, por lo que resulta importante conocerlas y dominarlas.

video [Las intenciones en Android](#)

Existen dos tipos de intenciones:>

- **Intenciones explícitas:** se indica **exactamente el componente a lanzar**. Su utilización típica es la de ir ejecutando los diferentes **componentes internos de una aplicación**. Por ejemplo, desde la actividad MisLugares lanzamos AcercaDeActivity por medio de una intención explícita.
- **Intenciones implícitas:** pueden solicitar **tareas abstractas**, como “quiero tomar una foto” o “quiero enviar un mensaje”. Además las intenciones se **resuelven en tiempo de ejecución**, de forma que el sistema mirará cuantos **componentes han registrado la posibilidad de ejecutar ese tipo de intención**. Si encuentra varias el sistema puede preguntar al usuario el componente que prefiere utilizar.

Además, como se ha estudiado en el apartado “*Comunicación entre actividades*” las intenciones ofrecen un servicio **de paso de mensajes que permite interconectar datos entre componentes**.

En concreto se utilizan intenciones cada vez que queramos:

- Lanzar una *actividad* (`startActivity()` y `startActivityForResult()`)
- Lanzar un *servicio* (`startService()`)
- Lanzar un *anuncio de tipo broadcast* (`sendBroadcast()`)
- Conectarnos con un *servicio* (`bindService()`)

En muchas ocasiones una *intención* no será inicializada por la aplicación, si no por el sistema, por ejemplo, cuando pedimos visualizar una página Web. En otras ocasiones será necesario que la **aplicación inicialice su propia intención**. Para ello **se creará un objeto de la clase Intent**.

Cuando se crea una Intención (es decir, se instancia un objeto de tipo Intent) esta contiene información de interés para que el sistema trate adecuadamente la intención o para el componente que recibe la intención. Puede incluir la siguiente información:

Nombre del componente: Identificamos el componente que queremos lanzar con la intención. Hay que utilizar el nombre de **clase totalmente cualificado que queremos lanzar** (`org.example.MisLugares.AcercaDeActivity`). El nombre del componente es opcional. En caso de no indicarse se utilizará otra información de la intención para obtener el componente a lanzar. A este tipo de intenciones se les conocía como intenciones explícitas.

Acción: Una cadena de caracteres donde indicamos la acción a realizar o en caso de un Receptor de anuncios (*Broadcast receiver*), la acción que tuvo lugar y que queremos reportar. La clase Intent define una serie de constantes para acciones genéricas que se enumeran a continuación:

Constante	componente a lanzar	Acción
ACTION_CALL	Actividad	Inicializa una llamada de teléfono.
ACTION_EDIT	Actividad	Visualiza datos para que el usuario los edite.
ACTION_MAIN	Actividad	Arranca como actividad principal de una tarea. (sin datos de entrada y sin devolver datos)
ACTION_SYNC	Actividad	Sincroniza datos en un servidor con los datos de un dispositivo móvil.
ACTION_BATTERY_LOW	Receptor anuncios	Advertencia de batería baja.
ACTION_HEADSET_PLUG	Receptor anuncios	Se han conectado o desconectado los auriculares.
ACTION_SCREEN_ON	Receptor anuncios	Se ha activado la pantalla.
ACTION_TIMEZONE_CHANGED	Receptor anuncios	Se cambia la selección de zona horaria.

Algunas acciones estándar de las Intenciones

También puedes definir tus propias acciones. En este caso has de indicar el paquete de tu aplicación como prefijo. Por ejemplo: org.example.mislugares.MUESTRA_MAPA_LUGARES

Categoría: Complementa a la acción. **Indica información adicional sobre el tipo de componente que ha de ser lanzado.** El número de categorías puede ampliarse arbitrariamente. No obstante, en la clase Intent se definen una serie de categorías genéricas que podemos utilizar.

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser invocada con seguridad por el navegador para mostrar los datos referenciados por un enlace (por ejemplo, una imagen o un mensaje de correo electrónico).
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando se presiona la tecla <i>Home</i> .
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias.

Algunas categorías estándar de las Intenciones

Una categoría suele utilizarse junto con una acción para aportar información adicional. Por ejemplo, indicaremos ACTION_MAIN a las actividades que pueden utilizarse como puntos de entrada de una aplicación. Indicaremos además CATEGORY_LAUNCHER para que la actividad sea mostrada en la pantalla de inicio.

Datos: Referencia a los datos con los que trabajaremos. Hay que expresar estos datos por medio de una URI (el mismo concepto ampliamente utilizado en Internet). Ejemplos de URIs son: <tel:963228525>, <http://www.androidcurso.com>, content://call_log/calls... En muchos casos resulta importante saber el tipo de datos con el que se trabaja. Con este propósito se indica el tipo MIME asociado a la URI, es decir, se utiliza el mismo mecanismo que en Internet. Ejemplos de tipos MIME son text/xml, image/jpeg, audio/mp3...

Extras: Información adicional que será recibida por el componente lanzado. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase **Bundle**. Su utilización ha sido descrita en la sección *Comunicación entre actividades*. Recordemos cómo se introducían estos valores en un Intent.

```
intent.putExtra("usuario", "Pepito Perez")
intent.putExtra("edad", 27);
```

En el apartado "*Creación de nuevas actividades*" hemos aprendido a lanzar una actividad de forma explícita utilizando el constructor Intent(Context contexto, Class<?> clase). Por ejemplo, para lanzar la actividad AcercaDeActivity escribíamos:

```
Intent intent = new Intent(this, AcercaDeActivity.class);
startActivity(intent);
```

Para lanzar una actividad de forma implícita podemos usar el constructor

```
Intent(String action, Uri uri). Por ejemplo:  
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:962849347"));  
startActivity(intent);
```

También se puede utilizar startActivityForResult() si esperamos que la actividad nos devuelva datos.

Ejercicio: Uso de intenciones implícitas

1. Crea un nuevo proyecto con nombre *Intenciones* y tipo Empty Activity.
2. El *Layout* de la actividad inicial ha de estar formado por cinco botones, tal y como se muestra a continuación:



3. Abre la actividad principal e incorpora los siguientes métodos:

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
  
    public void pgWeb(View view) {  
        Intent intent = new Intent(Intent.ACTION_VIEW,  
            Uri.parse("http://www.androidcurso.com/"));  
        startActivity(intent);  
    }  
  
    public void llamadaTelefono(View view) {  
        Intent intent = new Intent(Intent.ACTION_DIAL,  
            Uri.parse("tel:656894953"));  
        startActivity(intent);  
    }  
  
    public void googleMaps(View view) {  
        Intent intent = new Intent(Intent.ACTION_VIEW,  
            Uri.parse("geo:41.656313,-0.877351"));  
        startActivity(intent);  
    }  
  
    public void tomarFoto(View view) {  
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
        startActivity(intent);  
    }  
  
    public void mandarCorreo(View view) {  
        Intent intent = new Intent(Intent.ACTION_SEND);  
        intent.setType("text/plain");  
        intent.putExtra(Intent.EXTRA_SUBJECT, "asunto");  
        intent.putExtra(Intent.EXTRA_TEXT, "texto del correo");  
    }  
}
```

```

        intent.putExtra(Intent.EXTRA_EMAIL, new String[]{"jtomás@upv.es"});
        startActivity(intent);
    }

    public void accionCall(View view) {
        Intent intent = new Intent(Intent.ACTION_CALL,
            Uri.parse("tel:656894953"));

        //if (ActivityCompat.checkSelfPermission(MainActivity.this,
        Manifest.permission.CALL_PHONE)!= PackageManager.PERMISSION_GRANTED)
        //    return;
        //else
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) !=
PackageManager.PERMISSION_GRANTED) {
            // TODO: Consider calling
            //      ActivityCompat#requestPermissions
            // here to request the missing permissions, and then overriding
            // public void onRequestPermissionsResult(int requestCode, String[] permissions,
            //                                         int[] grantResults)
            // to handle the case where the user grants the permission. See the documentation
            // for ActivityCompat#requestPermissions for more details.
            return;
        }
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.CALL_PHONE) !=
PackageManager.PERMISSION_GRANTED) {
            // TODO: Consider calling
            //      ActivityCompat#requestPermissions
            // here to request the missing permissions, and then overriding
            // public void onRequestPermissionsResult(int requestCode, String[] permissions,
            //                                         int[] grantResults)
            // to handle the case where the user grants the permission. See the documentation
            // for ActivityCompat#requestPermissions for more details.
            return;
        }
        startActivity(intent);
    }

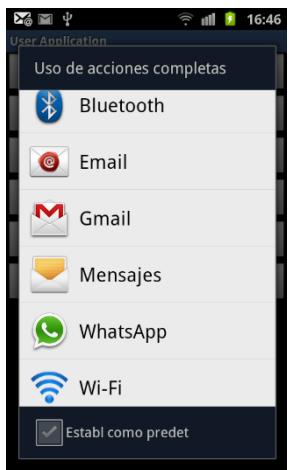
    public void buscarWeb(View view)
    {
        Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
        intent.putExtra(SearchManager.QUERY, "SEGAINVEX");
        startActivity(intent);
    }
}

```

4. Asocia el atributo onClick de cada uno de los botones al método correspondiente.

Si ejecutas esta aplicación en un emulador, es muy posible que los botones de mandar correo o GoogleMaps no funcionen. La razón es que no hay ninguna aplicación instalada en el emulador que sea capaz de realizar este tipo de acciones. Si tienes estos problemas, abre el AVD Manager y crea un dispositivo virtual con Google API. Estos dispositivos incorporan, además de las API de Android, algunas de las API de Google, como la de Google Maps (estas API se estudiarán más adelante).

. 6. Ejecuta la aplicación en un terminal real. Observa como el botón *mandar Correo* te permite seleccionar entre diferentes aplicaciones con esta funcionalidad.



7. Este resultado puede variar en función de las aplicaciones instaladas.

Recursos adicionales: Tabla con intenciones que podemos utilizar de aplicaciones Google

Aplicación	URI	Acción	Resultado
Browser	<code>http://dirección_web</code> <code>https://dirección_web</code>	VIEW	Abre una ventana de navegador con una URL.
	<code>"" (cadena vacía)</code> <code>http://dirección_web</code> <code>https://dirección_web</code>	WEB_SEARCH	Realiza una búsqueda web. Se indica la cadena de búsqueda en el extra SearchManager.QUERY
Dialer	<code>tel:número_telefono</code>	CALL	Realiza una llamada de teléfono. Los números válidos se definen en IETF RFC 3966 . Entre estos se incluyen: <code>tel:2125551212</code> y <code>tel:(212) 5551212</code> . Necesitamos el permiso android.permission.CALL_PHONE
	<code>tel:número_telefono voicemail:</code>	DIAL	Introduce un número sin llegar a realizar la llamada.
Google Maps	<code>geo:latitud,longitud</code> <code>geo:lat,long?z=zoom</code> <code>geo:0,0?q=dirección</code> <code>geo:0,0?q=búsqueda</code>	VIEW	Abre la aplicación Google Maps para una localización determinada. El campo z especifica el nivel de zoom.
Google Streetview	<code>google.streetview: cbll=latitud,longitud& cbp=1,yaw,pitch,zoom& mz=mapZoom</code>	VIEW	Abre la aplicación Street View para la ubicación dada. El esquema de URI se basa en la sintaxis que utiliza Google Maps. Solo el campo cbll es obligatorio.

Práctica: Uso de intenciones implícitas

1. Crea nuevos botones en la aplicación del ejercicio anterior y experimenta con otro tipo de acciones y URI. Puedes consultar la tabla anterior. A continuación tienes algunas propuestas:
2. Compara las acciones VIEW y WEB_SEARCH. ¿Encuentras alguna diferencia?
3. Compara las acciones CALL y DIAL. ¿Encuentras alguna diferencia?
4. Experimenta con Google Streetview.

Uso de intenciones en Mis Lugares

Objetivos:

Aplicar lo aprendido sobre intenciones en la aplicación Mis Lugares.

Ejercicio: Intenciones implícitas en Mis Lugares

1. En la clase CasosUsoLugar añade cuatro nuevos casos de uso:

```
// INTENCIÓNES
public void compartir(Lugar lugar) {
    Intent i = new Intent(Intent.ACTION_SEND);
    i.setType("text/plain");
    i.putExtra(Intent.EXTRA_TEXT,
              lugar.getNombre() + " - " + lugar.getUrl());
    actividad.startActivity(i);
}

public void llamarTelefono(Lugar lugar) {
    actividad.startActivity(new Intent(Intent.ACTION_DIAL,
        Uri.parse("tel:" + lugar.getTelefono())));
}

public void verPgWeb(Lugar lugar) {
    actividad.startActivity(new Intent(Intent.ACTION_VIEW,
        Uri.parse(lugar.getUrl())));
}

public final void verMapa(Lugar lugar) {
    double lat = lugar.getPosicion().getLatitud();
    double lon = lugar.getPosicion().getLongitud();
    Uri uri = lugar.getPosicion() != GeoPunto.SIN_POSICION
        ? Uri.parse("geo:" + lat + ',' + lon)
        : Uri.parse("geo:0,0?q=" + lugar.getDireccion());
    actividad.startActivity(new Intent("android.intent.action.VIEW", uri));
}
```

El primer método crea una intención implícita con acción ACTION_SEND. **Mandaremos un texto plano formado por el nombre del lugar y su URL. Esta información podrá ser recogida por cualquier aplicación que se haya registrado como enviadora de mensajes (WhatsApp, Gmail, SMS, etc.).**

El segundo método realiza **una llamada telefónica** al número del lugar. El tercero abre su **página Web**. El cuarto obtiene **la latitud y longitud del lugar**. Si alguna de las dos es distinta de cero, consideraremos que se ha introducido esta información y crearemos una URI basada en estos valores. Si son cero, consideraremos que no se han introducido, por lo que crearemos una URI basándonos en la dirección del lugar.

2. En la clase VistaLugarActivity hay que llamar a dos de estos métodos desde el menú. En el método onOptionsItemSelected() añade el código subrayado:

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.accion_compartir:
```

```
    usoLugar.compartir(lugar);
    return true;
case R.id.accion_llegar:
    usoLugar.verMapa(lugar);
    return true;
...
```

3. Ejecuta la aplicación, selecciona alguno de los lugares y utiliza la barra de acciones para verificar estas opciones. Para verificar la segunda opción, es importante que el terminal disponga de algún *software* de mapas (como Google Maps).

4. Abre el *layout vista_lugar.xml*. Localiza el LinearLayout que contiene el ícono y el texto con la dirección. Añade el atributo onClick como se muestra a continuación:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:onClick="VerMapa">
    <ImageView
        android:id="@+id/im_direccion"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="dirección"
        android:src="@android:drawable/ic_menu_myplaces" />
    <TextView
        android:id="@+id/direccion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="dirección" />
</LinearLayout>
...
```

5. Añade también el atributo onClick en los LinearLayout que contienen el teléfono y la URL utilizando el método adecuado.

6. Añade a VistaLugarActivity los siguientes métodos:

```
public void verMapa(View view)
{
    usoLugar.verMapa(lugar);
}
public void llamarTelefono(View view)
{
    usoLugar.llamarTelefono(lugar);
}
public void verPgWeb(View view)
{
    usoLugar.verPgWeb(lugar);
}
```

7. Verifica el funcionamiento de las nuevas intenciones implícitas.

Añadiendo fotografías en Mis Lugares

Objetivos:

Añadir la posibilidad de tomar, seleccionar y visualizar fotografías en la aplicación Mis Lugares.

En este apartado seguiremos trabajando con el uso de intenciones aplicándolas a la aplicación Mis Lugares. En concreto permitiremos que el usuario pueda asignar fotografías a cada lugar utilizando ficheros almacenados en su dispositivo o la cámara.

Poner y cambiar una foto

Ejercicio: Añadiendo fotografías desde la galería

1. En la clase VistaLugarActivity añade las siguientes constantes y atributos:

```
final static int RESULTADO_GALERIA = 2;
final static int RESULTADO_FOTO = 3;
private ImageView imageView;
```

Desde la actividad VistaLugarActivity llamamos a diferentes actividades y algunas de ellas nos tienen que devolver información. En estos casos llamamos a la actividad con startActivityForResult() pasándole un código que identifica la llamada. Cuando esta actividad termine se llamará al método onActivityResult(), que nos indicará el mismo código usado en la llamada. Como vamos a hacerlo con tres actividades diferentes, hemos creado tres constantes, con los respectivos códigos de respuesta. Actuando de esta forma conseguimos un código más legible.

2. El método onCreate() añade antes de actualizarVistas():

```
foto = findViewById(R.id.foto); //Es el ImageView de la foto del Lugar
```

3. Busca en vista_lugar.xml el ImageView con descripción "fotografia" y añade el atributo:

```
android:onClick="ponerDeGaleria"
```

4. Añade la siguiente función en la actividad:

```
public void ponerDeGaleria(View view)
{
    usoLugar.ponerDeGaleria(imageView); //imageView es dummy, solo por compatibilidad
}
```

5. Añade el siguiente caso de uso a la clase CasosUsoLugares:

```
// FOTOGRAFÍAS
public void ponerDeGaleria(View view) //view es dummy, no se usa
{
    String action;
    if (android.os.Build.VERSION.SDK_INT >= 19)
    { // API 19 - Kitkat
        action = Intent.ACTION_OPEN_DOCUMENT;
    }
}
```

```

    else
    {
        action = Intent.ACTION_PICK;
    }
    //Instancia el intent
    Intent intent = new
Intent(action,MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    //Configura el intent
    intent.addCategory(Intent.CATEGORY_OPENABLE);
    intent.setType("image/*");
    //Lanza el intent
    actividad.startActivityForResult(intent, RESULTADO_GALERIA);
}

```

Este método crea una intención indicando que queremos seleccionar contenido. El contenido será proporcionado por el *Content Provider* MediaStore, además le indicamos que nos interesan imágenes del almacenamiento externo. Tipicamente se abrirá la aplicación galería de fotos (u otra similar). Observa, que se usan dos acciones diferentes: ACTION_OPEN_DOCUMENT solo está disponible a partir del API 19, tiene la ventaja que no requiere que la aplicación pida permiso de lectura. Cuando el usuario selecciona un fichero el Content Provider, dará a nuestra aplicación permiso de lectura (o incluso de escritura) pero solo para el archivo solicitado. No se considera una acción peligrosa dado que es el usuario quien selecciona el archivo a compartir con la aplicación. Si se ejecuta en un API anterior al 19, tendremos que usar ACTION_PICK, que sí que requiere dar permisos de lectura en la memoria externa. Una vez concedido este permiso la aplicación podría aprovechar y leer otros ficheros sin la intervención directa del usuario. Como necesitamos una respuesta usamos startActivityForResult() con el código adecuado.

- Si la versión del dispositivo es anterior al API 19, vamos a tener que pedir permiso para leer ficheros de la memoria externa. En *AndroidManifest.xml* añade dentro de la etiqueta <manifest ...> </manifest> el siguiente código:

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

- En el método onActivityResult() añade la sección else if que se muestra:

```

...
else if (requestCode == RESULTADO_GALERIA)
{
    if (resultCode == Activity.RESULT_OK)
    {
        usoLugar.ponerFoto(pos, data.getDataString(), foto);
    }
    else
    {
        Toast.makeText(this, "Foto no cargada", Toast.LENGTH_LONG).show();
    }
}
...

```

Comenzamos verificando que volvemos de la actividad lanzada por la intención anterior. Comprobamos que el usuario no ha cancelado la operación. En este caso, nos tiene que haber pasado en la intención de respuesta, data, una URI con la foto seleccionada. Esta URI puede ser del tipo file://..., content://...o http://... dependiendo de que aplicación haya resuelto esta intención. El siguiente paso consiste en modificar el contenido de la vista que muestra la foto, imageView, con esta URI. Lo hacemos en el método ponerFoto():

- Añade el siguiente CasosUsoLugar:

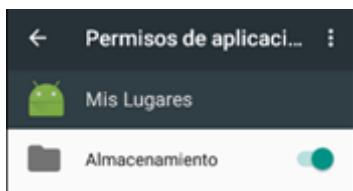
```

public void ponerFoto(int pos, String uri, ImageView foto)
{
    Lugar lugar = lugares.elemento(pos); //Apunta al Lugar
    lugar.setFoto(uri); //Le pone la foto que nos han pasado en el URI
    visualizarFoto(lugar, foto, 0); //Muestra la foto
}

public void visualizarFoto(Lugar lugar, ImageView fotoFondo, int id)
{
    if (lugar.getFoto() != null && !lugar.getFoto().isEmpty())
    {
        fotoFondo.setImageURI(Uri.parse(lugar.getFoto()));
    }
    else
    {
        //Si no tenemos una foto que visualizar ponemos una por defecto
        if(id!=0) fotoFondo.setId(id); //Ya le hemos pasado el contexto
        else fotoFondo.setImageBitmap(null);
    }
}

```

El primer método comienza obteniendo el lugar que corresponde al id para modificar la URI de su foto. Luego llama a `visualizarFoto()`. Este verifica que la URI que acabamos de asignar no está vacía. Si es así, la asigna al `ImageView` que nos han pasado para que la imagen se represente en pantalla. En caso contrario, se le asigna un `Bitmap` igual a `null`, que es equivalente a que no se represente ninguna imagen.



9. Ya puedes ejecutar la aplicación. Si añades una fotografía a un lugar, esta se visualizará. Sin embargo, si vuelve a la lista de lugares y seleccionas el mismo lugar al que asignaste la fotografía, ésta ya no se representa. La razón es que no hemos visualizado la foto al crear la actividad.

10. En el método `actualizarVistas()` añade la siguiente línea al final:

```
usoLugar.visualizarFoto(lugar, foto, R.id.foto);
```

11. Verifica de nuevo el funcionamiento de la aplicación.

Poner fotos la App desde la cámara

Ejercicio paso a paso: Añadiendo fotografías desde la cámara

1. Añade los siguientes atributos a la clase `VistaLugarActivity`:

```
private Uri uriUltimaFoto;
```

Como veremos, necesitamos esta variable en dos métodos diferentes. Por lo tanto, la declaramos de forma global.

2. Añade el siguiente método a la clase `CasosUsoLugar`:

```

public Uri tomarFoto(int codidoSolicitud)
{
    try //creo un nombre de fichero de foto reciente
    {
        Uri uriUltimaFoto;
        //creo nombre fichero llamado img_DATE.jpg que estaría en el fileProvider
        File file = File.createTempFile(
            "img_" + (System.currentTimeMillis() / 1000), ".jpg",
            actividad.getExternalFilesDir(Environment.DIRECTORY_PICTURES));
        if (Build.VERSION.SDK_INT >= 24)
        {
            uriUltimaFoto = FileProvider.getUriForFile(
                actividad, "com.example.mislugares.fileProvider", file);
        }
        else
        {
            uriUltimaFoto = Uri.fromFile(file);
        }
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        //Configuro el intent con el medio y el uri que quiero
        intent.putExtra(MediaStore.EXTRA_OUTPUT, uriUltimaFoto);
        actividad.startActivityForResult(intent, codidoSolicitud);
        return uriUltimaFoto; //devuelvo el control con el uri
    }
    catch (IOException ex)
    {
        Toast.makeText(actividad, "Error al crear fichero de imagen",
                      Toast.LENGTH_LONG).show();
        return null;
    }
}

```

Este método crea una intención indicando que queremos capturar una imagen desde el dispositivo. Típicamente se abrirá la aplicación cámara de fotos. A esta intención vamos a añadirle un extra con una URI al fichero donde queremos que se almacene la fotografía. Para crear el fichero, se utiliza `createTempFile()` indicando nombre, extensión y directorio. El método `currentTimeMillis()` nos da el número de milisegundos transcurridos desde 1970. Al dividir entre 1000, tenemos el número de segundos. El objetivo que se persigue es que, al crear un nuevo fichero, su nombre nunca coincida con uno anterior. El directorio del fichero será el utilizado para almacenar fotos privadas en la memoria externa. Estos ficheros serán de uso exclusivo para tu aplicación. Además, si desinstalas la aplicación este directorio será borrado. Si quieres que los ficheros sean de acceso público utiliza `getExternalStoragePublicDirectory()`. Una vez tenemos el fichero, hay dos alternativas para crear la URI. Si el API de Android donde se ejecuta la aplicación es 24 o superior, podemos crear el fichero asociado a un Content Provider nuestro. Esta acción no requiere solicitar permiso de escritura. Si el API es anterior al 24, no se dispone de esta acción, y el fichero será creado de forma convencional en la memoria externa. El inconveniente es que para realizar esta acción tendremos que pedir al usuario permiso de escritura en la memoria externa. Al concedernos este permiso, también podremos borrar o sobreescribir cualquier fichero que el usuario tenga en esta memoria. Por lo que muchos usuarios no querrán darnos este permiso. Finalmente se añade la extensión del fichero. Al final llamamos a `startActivityForResult()` con el código que nos han pasado.

3. Si la versión mínima de API es anterior a 24, vamos a tener que pedir permiso para leer ficheros de la memoria externa. En `AndroidManifest.xml` añade dentro de la etiqueta `<manifest ...></manifest>` el siguiente código:

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

NOTA: Observa como hemos tenido que solicitar permiso para acceder a la memoria externa, pero en este no es necesario solicitar permiso para tomar una fotografía. La razón es que realmente nuestra aplicación no toma la fotografía directamente, si no que por medio de una intención lanzamos otra aplicación, que si que tiene este permiso.

Creacion de un FileProvider

- En un punto anterior hemos utilizado un Content Provider para almacenar ficheros. Para crearlo añade en AndroidManifest.xml dentro de la etiqueta <application ...> </application> el siguiente código:

```
...
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.example.mislugares.fileProvider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths" />
</provider>
...
```

- Crea un nuevo recurso con nombre "file_paths.xml" en la carpeta res/ xml:

```
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-files-path name="my_images" path="/" />
</paths>
```

Has de **com.example.mislugares.fileProvider** por un identificador único que incluya tu nombre o empresa y la aplicación. Ha de coincidir con el valor indicado en la función tomarFoto(). Cambia com.example.package.name por el paquete de la aplicación.

- Busca en *vista_lugar.xml* el ImageView con descripción “logo cámara” y añade el atributo:

android:onClick="tomarFoto"

- En VistaLugarActivity añade:

```
public void tomarFoto(View view) //view es dummy
{
    uriUltimaFoto = usoLugar.tomarFoto(RESULTADO_FOTO);
}
```

- En el método onActivityResult() añade la sección else if que se muestra:

```
} else if (requestCode == RESULTADO_GALERIA
    ...
}
else if (requestCode == RESULTADO_FOTO)
{
    if (resultCode == Activity.RESULT_OK && uriUltimaFoto!=null)
    {
        lugar.setFoto(uriUltimaFoto.toString());
        usoLugar.ponerFoto(pos, lugar.getFoto(), foto);
    } //if
else
{
    Toast.makeText(this, "Error en captura", Toast.LENGTH_LONG).show();
```

```
    } //else  
} //else if
```

Comenzamos verificando que volvemos de la actividad lanzada por la intención anterior, que el usuario no ha cancelado la operación y que uriUltimaFoto ha sido inicializado. En este caso, se nos pasa información en la intención de respuesta, pero sabemos que en uriUltimaFoto está almacenada la URI con el fichero donde se ha almacenado la foto. Guardamos esta URI en el campo adecuado de lugar e indicamos que se guarde y se represente en la vista foto.

9. Verifica de nuevo el funcionamiento de la aplicación.

NOTA: *En algunos dispositivos puede aparecer un error de memoria si la cámara está configurada con mucha resolución. En estos casos puedes probar con la cámara delantera.*

Eliminar foto

Ejercicio paso a paso: Añadiendo un botón para eliminar fotografías

1. En el layout *vista_lugar.xml* añade el siguiente botón dentro del LinearLayout donde están los botones para la cámara y para galería:

```
<ImageView  
    android:layout_width="40dp"  
    android:layout_height="40dp"  
    android:contentDescription="Eliminar foto"  
    android:onClick="eliminarFoto"  
    android:src="@android:drawable/ic_menu_close_clear_cancel" />
```

2. Añade el siguiente método a la clase *VistaLugarActivity*:

```
public void eliminarFoto(View view) {  
    usoLugar.ponerFoto(pos, "", foto);  
}
```

3. Verifica el funcionamiento del nuevo botón.

NOTA: *Si lo deseas puedes poner un cuadro de dialogo para confirmar la eliminación.* Véase la práctica «Un cuadro de diálogo para confirmar el borrado». No obstante, es reversible; se puede volver a asignar la fotografía buscándola en la galería.

Adaptar la resolución de las fotos a la pantalla

Cargar fotografías grandes de forma eficiente

Las fotografías introducidas por el usuario pueden tener muy diversas procedencias, pudiendo llegar a los 16 Mpx en dispositivos de altas prestaciones. Cuando trabajas con fotografías es muy importante que tengas en cuenta que la memoria es un recurso limitado. Por lo tanto, es muy probable que cuando trates de cargar una imagen de gran tamaño, tu aplicación se detenga, mostrando en el *LogCat* el siguiente error:

Tag	Text
AndroidRuntime	FATAL EXCEPTION: main
AndroidRuntime	java.lang.OutOfMemoryError

Otro posible error cuando intentas cargar una imagen de ancho o alto mayor de 4096 px, es que la imagen no se muestre, aunque la aplicación continua ejecutándose. En este caso abre el *LogCat* y busca si aparece el siguiente *warning*:

Tag	Text
OpenGLRenderer	Bitmap too large to be uploaded into a texture (4128x2322, max=4096x4096)

Trabajar con una fotografía de muy alta definición, no tiene mucho sentido si la vamos a representar en una la pantalla de móvil, donde en raras ocasiones se supera 1 Mpx. Este problema se agrava cuando queremos mostrar una miniatura de la fotografía (*thumbnail*). ¿Para que cargar en memoria una fotografía en alta resolución, cuando solo la mostramos a baja resolución?

Una estrategia para evitar los problemas que hemos enumerado, consiste en **cargar en memoria, no la fotografía original, sino una versión con una resolución adaptada a nuestras necesidades**. La clase **BitmapFactory** nos ofrece herramientas para trabajar de forma adecuada[\[1\]](#). Esta clase permite **reescalar una imagen** a una resolución menor, cargando en memoria un Bitmap de tamaño adecuado a nuestras necesidades. BitmapFactory solo admite factores de reducción que sean potencias de dos (1, 2, 4, 8, 16, etc.). Este factor de reducción se aplica al ancho y alto de la imagen. Por ejemplo, una imagen con resolución 2048x1536 con un factor de reducción 4, produce un Bitmap de aproximadamente 512x384. En el siguiente ejercicio se indica los pasos que tenemos que seguir para carga imágenes con una resolución adecuada.

Ejercicio: Evitando errores de memoria con fotografías grandes

1. Trata de cargar imágenes de gran resolución en la aplicación *Mis Lugares* y observa si que producen alguno de los errores comentados.
2. En el método `visualizaFoto()` de la clase `CasosUsoLugar` reemplaza:

```
imageView.setImageResource(Uri.parse(uri));
```

por:

```
fotoFondo.setImageBitmap(reduceBitmap(actividad, lugar.getFoto(), 1024, 1024));
```

Antes del cambio, poníamos la URI que nos indicaba el usuario con la foto, directamente en el ImageView. Esto suponía que lo que se cargaba en memoria todos los pixeles de la imagen. En una fotografía de 16 Mpx, codificando cada pixel con 4 bytes, supone 64 Mb de memoria, lo que podría ocasionar graves problemas de memoria.

En el nuevo código propuesto, vamos a crear un objeto Bitmap para asignarlo al ImageView. Este Bitmap es creado por el método `reduceBitmap()` a partir de la URI indicada. Pero ahora, reescalado a una resolución que no ha de superar al ancho y alto indicado. En *Mis Lugares* se ha decidido usar un ancho y alto inferior o igual a 1024. De esta forma, la imagen tendrá una resolución inferior a 1 Mpx y un consumo de memoria inferior a 4 Mb.

3. Añade el siguiente método a la clase `CasosUsoLugar`:

```

private Bitmap reduceBitmap(Context contexto, String uri, int maxAncho, int maxAlto)
{
    try
    {
        InputStream input = null;
        Uri u = Uri.parse(uri);
        if (u.getScheme().equals("http") || u.getScheme().equals("https"))
        {
            input = new URL(uri).openStream();
        }
        else
        {
            input = contexto.getContentResolver().openInputStream(u);
        }
        final BitmapFactory.Options options = new BitmapFactory.Options();
        options.inJustDecodeBounds = true;
        options.inSampleSize = (int) Math.max(
            Math.ceil(options.outWidth / maxAncho),
            Math.ceil(options.outHeight / maxAlto));
        options.inJustDecodeBounds = false;
        return BitmapFactory.decodeStream(input, null, options);
    }
    catch (FileNotFoundException e)
    {
        makeText(contexto, "Fichero/recurso de imagen no encontrado",
LENGTH_LONG).show();
        e.printStackTrace();
        return null;
    }
    catch (IOException e)
    {
        makeText(contexto, "Error accediendo a imagen", LENGTH_LONG).show();
        e.printStackTrace();
        return null;
    }
}

```

El propósito de este método es obtener un Bitmap a partir de la URI indicada pero reescalandolo a una resolución que no supere un ancho y un alto. El nuevo tamaño siempre se obtendrá dividiendo ancho y alto por una potencia de dos. Es decir, el nuevo alto y ancho será la mitad, o una cuarta parte, o un octavo, ... de las dimensiones originales.

La clase BitmapFactory nos ofrece varias alternativas para decodificar un Bitmap desde distintas fuentes (decodeByteArray(), decodeFile(), decodeResource(), etc.). En *Mis Lugares* la fotografía puede provenir de un fichero, cuando se toma de la cámara o desde un ContentProvider cuando se selecciona de la galería. Para cubrir las procedencias consideramos que nos han pasado un identificador de URI (`Uri.parse(uri)`). Así, podemos abrir el Stream asociado a la URI (`openInputStream()`) y este Stream puede ser decodificado utilizando el método `decodeStream()`.

En una primera decodificación no nos interesa cargar toda la fotografía en memoria, si no solo estamos interesados en obtener sus dimensiones originales. Para conseguir esto creamos un objeto de la clase `BitmapFactory.Options` y activamos la opción `inJustDecodeBounds`. De esta forma, la siguiente llamada a `decodeStream()` no decodificará toda la fotografía y la cargará en memoria, si no que se limitará obtener sus dimensiones y almacenarlas en `options.outWidth` y `options.outHeight`.

La siguiente línea calcula el factor de reducción deseado y lo almacena en `inSampleSize`. Para ello, se divide el ancho de la foto entre el ancho máximo redondeando hacia arriba. Se realiza la misma acción con el alto y se selecciona el mayor de los dos factores.

A continuación desactivamos la opción `inJustDecodeBounds`, dado que ahora si queremos que se decodifique la image, y hacemos la llamada a `decodeStream()`. En esta decodificación la opción `inSampleSize` actuará como factor de reducción.

Como hemos indicado, `BitmapFactory` solo trabaja con valores de reducción que coincidan con potencias de dos. Si se indica en `inSampleSize` un valor que no coincide con 1, 2, 4, 8, etc. se tomará la potencia de dos inferior. Un valor inferior a 1 se considerará como 1.

4. Ejecuta de nuevo la aplicación y verifica que han desaparecido los problemas con la memoria. También es posible que con esta modificación las imágenes pierdan nitidez. Recuerda que ahora el ancho/alto mayor puede haberse reducido a un valor entre 512 y 1024. Si consideras que la resolución es insuficiente reemplaza los valores 1024 por algo mayor, como 2048.

[1] <http://developer.android.com/training/displaying-bitmaps/load-bitmap.html>

Unidad 6 : Ciclo de vida de una Actividad y Seguridad

Introducción a la unidad

En esta unidad comenzaremos con un aspecto de vital importancia en el desarrollo de aplicaciones en Android, el **ciclo de vida de una actividad**. Es decir, cómo las actividades son creadas, ejecutadas, puestas en espera y finalmente destruidas.

Continuaremos con una breve introducción sobre multimedia en Android. El **API de Android** viene preparado con excelentes características de reproducción multimedia, permite la **reproducción de gran variedad de formatos**, tanto de audio como de vídeo.

Terminamos la unidad estudiando los **fundamentos del sistema de seguridad que incorpora Android**. Se trata de un aspecto vital para protegernos de aplicaciones mal intencionadas que intenten violar la privacidad del usuario y evitar que realicen acciones no deseada. Gracias al **sistema de permisos**, se consigue impedir que las aplicaciones realicen acciones comprometidas, si previamente no han solicitado el permiso adecuado.

Objetivos:

- Comprender el ciclo de vida de una actividad Android.
- Utilizar de forma correcta los diferentes eventos relacionados con el ciclo de vida.
- Aprender cuándo y cómo guardar el estado de una actividad.
- Repasar las facilidades multimedia disponibles en Android, que formatos soporta y las clases que hemos de utilizar.
- Mostrar los pilares de la seguridad en Android
- Describir como Android crea un usuario Linux asociado a cada aplicación.
- Describir el esquema de permisos en Android y enumerar los permisos más importantes.

Ciclo de vida de una actividad

Objetivos:

Describir el ciclo de vida de una actividad en Android.

El ciclo de vida de una aplicación Android es bastante diferente al ciclo de vida de una aplicación en otros S.O., como Windows. La mayor diferencia es que, en Android el ciclo de vida es controlado principalmente por el sistema, en lugar de ser controlado directamente por el usuario.

video[Tutorial] [Ciclo de vida de una aplicación en Android](#)

Una aplicación en Android está formada por un conjunto de elementos básicos de interacción con el usuario, conocidos como actividades. Además de varias actividades, una aplicación también puede contener servicios. El ciclo de vida de los servicios se estudiará en el capítulo 8. Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, sino de actividad. El sistema mantiene una pila con las actividades previamente visualizadas, de forma que el usuario puede regresar a la actividad anterior pulsando la tecla “retorno”.

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso se crea con la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlada directamente por la aplicación, sino que es el sistema el que determina cuándo destruir el proceso. Lo hace basándose en el conocimiento que tiene de las partes de la aplicación que están corriendo (actividades y servicios), en la importancia de dichas partes para el usuario y en cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esa aplicación. En estos casos, será responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sean reiniciadas conserven su estado.

Como vemos, Android es sensible al ciclo de vida de una actividad; por lo tanto, necesitas comprender y manejar los eventos relacionados con el ciclo de vida si quieres crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

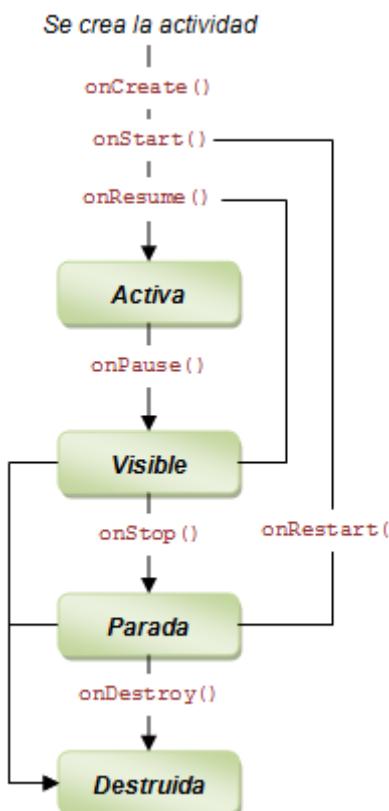
Activa (Running): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

Visible (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.

Parada (Stopped): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.

Destruida (Destroyed): Cuando la actividad termina al invocarse el método `finish()`, o es matada por el sistema.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.



Ciclo de vida de una actividad.

onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para **realizar todo tipo de inicializaciones**, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

onStart(): Nos indica que la actividad está **a punto de ser mostrada** al usuario.

onResume(): Se llama cuando la **actividad va a comenzar a interactuar con el usuario**. Es un buen lugar para lanzar las animaciones y la música.

onPause(): Indica que la actividad está **a punto de ser lanzada a segundo plano**, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

onStop(): La actividad **ya no va a ser visible para el usuario**. ¡Ojo si hay muy poca memoria! es posible que la actividad se destruya sin llamar a este método.

onRestart(): Indica que la actividad **va a volver a ser representada después de** haber pasado por **onStop()**.

onDestroy(): Se llama **antes de que la actividad sea totalmente destruida**. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Ejercicio: ¿Cuándo se llama a los eventos del ciclo de vida en una actividad?

En este ejercicio vamos a implementar todos los métodos del ciclo de vida de la actividad principal de MisLugares y añadiremos un toast para mostrar cuando se ejecuta. De esta forma comprenderemos mejor cuando se llama a cada método.

1. Abre la actividad *MainActivity* del proyecto *Mis Lugares*.

2. Añade en el método *onCreate()* el siguiente código:

```
Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
```

3. Añade los siguientes métodos:

```
@Override protected void onStart() {
    super.onStart();
    Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
}

@Override protected void onResume() {
    super.onResume();
    Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
}

@Override protected void onPause() {
    Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
    super.onPause();
}

@Override protected void onStop() {
    Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
    super.onStop();
}

@Override protected void onRestart() {
    super.onRestart();
    Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
}

@Override protected void onDestroy() {
    Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
    super.onDestroy();
}
```

4. Ejecuta la aplicación y observa la secuencia de Toast.

5. Selecciona la opción *Acerca de...* y luego regresa a la actividad. Observa la secuencia de Toast.

6. Selecciona la opción *Preferencias* y luego regresa a la actividad. Observa la secuencia de Toast.

7. Sal de la actividad y observa la secuencia de Toast.

Ciclo de vida de los procesos en Android

Objetivos:

Mostrar cómo Android clasifica los procesos y que criterio sigue para eliminarlos en caso de necesitar memoria.

¿Qué proceso se elimina?

Objetivos:

Ilustrar mediante un ejemplo el ciclo de vida de los procesos en Android.

Como hemos comentado, **Android mantiene en memoria todos los procesos que quepan aunque estos no se estén ejecutando**. Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, **el sistema ha de determinar qué proceso de los que están en ejecución ha de ser eliminado**. Android ordena los procesos en una lista jerárquica, asignándole a cada uno de ellos una determinada "importancia". Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (actividades y servicios) y el estado de estos componentes.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

Proceso de primer plano: (*Foreground process*) Hospeda una actividad en la superficie de la pantalla y con la cual el usuario está interactuando (su método `onResume()` ha sido llamado). Debería haber solo uno o unos pocos procesos de este tipo. **Sólo serán eliminados como último recurso, si es que la memoria está tan baja** que ni siquiera estos procesos pueden continuar corriendo.

Proceso visible: (*Visible process*) Hospeda **una actividad que está visible en la pantalla, pero no en el primer plano** (su método `onPause()` ha sido llamado). Considerado importante, no será eliminado a menos que sea necesario para mantener los procesos de primer plano.

Proceso de servicio: (*Service process*) Hospeda un **servicio que ha sido inicializado con el método `startService()`**. Aunque estos procesos **no son directamente visibles** al usuario, **generalmente están haciendo tareas que para el usuario son importantes** (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos). El sistema siempre tratará de mantener esos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.

Proceso de fondo: (*Background process*) Hospeda **una actividad que no es actualmente visible al usuario** (su método `onStop()` ha sido llamado). Si estos procesos son eliminados no tendrán un directo impacto en la experiencia del usuario. Como, hay muchos de estos procesos, el sistema asegura que el último proceso visto por el usuario sea el último en ser eliminado.

Proceso vacío: (*Empty process*) No hospeda a ningún componente de aplicación activo. La única razón para mantener **ese proceso es tener un "caché" que permita mejorar el tiempo de activación en la próxima vez** que un componente de su aplicación sea ejecutado.

video[Tutorial] [Ciclo de vida de los procesos en Android](#)

video[Tutorial] [Ciclo de vida de las aplicación en Android: Un ejemplo paso a paso](#)

Práctica: Aplicando eventos del ciclo de vida en la actividad inicial.

Los conceptos referentes al ciclo de vida de una aplicación son imprescindible para el desarrollo de aplicaciones estables en Android. Para reforzar estos conceptos te proponemos el siguiente ejercicio en el que vamos a reproducir una música de fondo en la actividad principal.

1. Abre el proyecto *Mis Lugares*.

2. Busca un fichero de audio (en este capítulo se listan los formatos soportados por Android). Renombra este fichero a *audio.xxx* y cópialo a la carpeta *res/raw*. Si *raw* no existe créalo.

NOTA: Cada vez que ejecutes el proyecto este fichero será añadido al paquete .apk. Si este fichero es muy grande la aplicación también lo será, lo que ralentizará su instalación. Para agilizar la ejecución te recomendamos un fichero muy pequeño, por ejemplo un .mp3 de corta duración o un fichero MIDI (.mid). Si no encuentras ninguno puedes descargar este: <http://www.dcomg.upv.es/~jtomás/android/ficheros/audio.mid>.

3. Abre la actividad MainActivity y declara el siguiente objeto:

```
MediaPlayer mp;
```

4. Añade las siguientes líneas en el método onCreate():

```
mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

5. Ejecuta el proyecto y verifica que cuando sales de la actividad la música sigue sonando un cierto tiempo.

6. Utilizando los eventos del ciclo de vida queremos que cuando la actividad deje de estar **activa** el audio deje de escucharse. Puedes utilizar los métodos:

```
mp.pause();
mp.start();
```

7. Verifica que funciona correctamente.

Práctica: Aplicando eventos del ciclo de vida en la actividad inicial (II)

1. Tras realizar el ejercicio anterior, ejecuta la aplicación y abre la actividad Acerca de... La música ha de detenerse.

2. Nos interesa que mientras parte de esta actividad esté visible (como ha ocurrido en el punto anterior) la música se escuche. Es decir, utilizando los eventos del ciclo de vida queremos que cuando la actividad deje de estar **visible** el audio deje de escucharse.

3. Verifica que cuando abres la actividad Acerca de... la música continua reproduciéndose.

4. Pasa ahora a una actividad que ocupe la totalidad de la pantalla (por ejemplo la actividad Juego o VistaLugarActivity). En teoría la música tendría que detenerse, dado que la

actividad MainActivity ya no es visible y tendría que haber pasado a estado parada. Observa como la música acaba deteniéndose, pero es posible que tarde unos segundos. Esto se debe a que la llamada al método onStop() no es prioritaria, por lo que el sistema puede retardar su ejecución. En caso de tratarse de información visual, en lugar de acústica, este retardo no tendría una repercusión directa para el usuario, dado que la actividad no es visible.

5. Tras el problema detectado en el punto anterior, deshaz los cambios introducidos en esta práctica y deja la aplicación como se pedía en la práctica anterior.

Guardar el estado de las Actividades en Android

video [Guardar el estado de las actividades en Android](#)

Objetivos:

Describir en que ocasiones hemos de guardar el estado de una actividad en Android.

Cuando el usuario ha estado **utilizando una actividad, y tras cambiar a otras, regresa a la primera, lo habitual es que esta permanezca en memoria y continúe su ejecución sin alteraciones**. Como hemos explicado, **en situaciones de escasez de memoria**, es posible que el sistema **haya eliminado** el proceso que ejecutaba la actividad. En este caso, el proceso será creado de nuevo, pero se habrá perdido su estado, es decir, se **habrá perdido el valor de sus variables y el puntero de programa**. Como consecuencia, si el usuario estaba a mitad de un proceso de edición o estaba reproduciendo un audio en un punto determinado perderá esta información. En este apartado estudiaremos un mecanismo sencillo que nos proporciona Android para resolver este problema.

NOTA: *Cuando se ejecuta una actividad sensible a la inclinación del teléfono, es decir puede verse en horizontal o en vertical, se presenta un problema similar al anterior. La actividad es destruida y vuelta a construir con las nuevas dimensiones de pantalla y por lo tanto se llama de nuevo al método onCreate. Antes de que la actividad sea destruida también resulta fundamental guardar su estado.*

Para guardar el estado de una actividad has de utilizar los siguientes dos métodos:

onSaveInstanceState(Bundle): Invoca el sistema cuando ha de destruir una actividad, que más adelante ha de restaurar (por cambiar su inclinación o por falta de memoria), para permitir a la actividad guardar su estado.

onRestoreInstanceState(Bundle): Se invoca cuando se restaura la actividad para recuperar el estado guardado por onSaveInstanceState().

NOTA: *Nunca utilices el método onSaveInstanceState() para guardar los datos generados por una actividad (como guardar un formulario en una base de datos). Este método es llamado cuando el sistema ha de destruir una actividad que va a ser recuperada en un futuro. Si esta actividad es destruida llamando a finish() o pulsando el botón atrás, el método onSaveInstanceState() no será llamado. Para guardar estos datos utiliza onPause() o onStop().*

Veamos un ejemplo de lo sencillo que resulta guardar la información de una variable tipo cadena de caracteres y entero.

```
String var;  
int pos;  
  
@Override protected void onSaveInstanceState(Bundle guardarEstado)  
{  
    super.onSaveInstanceState(guardarEstado);  
}
```

```

        guardarEstado.putString("variable", var);
        guardarEstado.putInt("posicion", pos);
    }

@Override protected void onRestoreInstanceState(Bundle recEstado) {
    super.onRestoreInstanceState(recEstado);
    var = recEstado.getString("variable");
    pos = recEstado.getInt("posicion");
}

```

Práctica: Guardando el estado en la actividad inicial.

1. Ejecuta el proyecto *Mis Lugares*.
2. Cambia de orientación el teléfono. Observarás como la música se reinicia cada vez que lo haces.
3. Utilizando los métodos para guardar el estado de una actividad, trata de que cuando se volteá el teléfono, el audio continúe en el mismo punto de reproducción. Puedes utilizar los siguientes métodos:

```
int pos = mp.getCurrentPosition();
mp.seekTo(pos);
```

5. Verifica el resultado.

Solución: A continuación se muestra una posible solución al ejercicio :

En MainActivity añade los métodos:

```

/***********************
 * Guarda el estado de la actividad antes de ser destruida por el sistema
 * @param estadoGuardado Bundle para guardar el estado
***********************/
@Override protected void onSaveInstanceState(Bundle estadoGuardado)
{
    super.onSaveInstanceState(estadoGuardado);
    if (mp != null) {
        int pos = mp.getCurrentPosition();
        estadoGuardado.putInt("posicion", pos);
    }
}
/***********************
 * Recupera el estado de la actividad cuando esta fué destruida por el
 * sistema
 * @param estadoGuardado
***********************/
@Override protected void onRestoreInstanceState(Bundle estadoGuardado)
{
    super.onRestoreInstanceState(estadoGuardado);
    if (estadoGuardado != null && mp != null)
    {
        int pos = estadoGuardado.getInt("posicion");
        mp.seekTo(pos);
    }
}

```

Multimedia en Android

video [Multimedia en Android](#)

Objetivos:

Introducir las capacidades multimedia disponibles en Android.

La integración de contenido multimedia en nuestras aplicaciones resulta muy sencilla gracias a la gran variedad de facilidades que nos proporciona el API.

Concretamente podemos reproducir audio y vídeo desde orígenes distintos:

- Desde un fichero almacenado en el dispositivo.
- Desde un recurso que está incrustado en el paquete de la aplicación (fichero .apk).
- Desde un *stream* que es leído desde una conexión de red. En este punto admite dos posibles protocolos (`http://` y `rstp://`)

También resulta sencilla la grabación de audio y vídeo, siempre que el *hardware* del dispositivo lo permita.

En la siguiente lista se muestran las clases de Android que nos permitirán acceder a los servicios Multimedia:

MediaPayer: Reproducción de audio/vídeo desde ficheros o *streams*.

MediaController: Visualiza controles estándar para mediaPlayer (pausa, stop...).

VideoView: Vista que permite la reproducción de vídeo.

MediaRecorder: Permite grabar audio y vídeo.

AsyncPlayer: Reproduce lista de audios desde un *thread* secundario.

AudioManager: Gestiona varias propiedades del sistema (volumen, tonos...).

AudioTrack: Reproduce un búfer de audio PCM directamente por *hardware*.

SoundPool: Maneja y reproduce una colección de recursos de audio.

JetPlayer: Reproduce audio y video interactivo creado con JetCreator.

Camera: Cómo utilizar la cámara para tomar fotos y vídeo.

FaceDetector: Identifica la cara de la gente en un bitmap.

La plataforma Android soporta una gran variedad de formatos, muchos de los cuales pueden ser tanto decodificados como codificados. A continuación, mostramos una tabla con los formatos multimedia soportados. No obstante algunos modelos de móviles pueden soportar formatos adicionales que no se incluyen en la tabla, como por ejemplo DivX.

Cada desarrollador es libre de usar los formatos incluidos en el núcleo del sistema o aquellos que solo se incluyen en algunos dispositivos.

Tipo	Formato	Codifica	Decodifica	Detalles	fichero soportado
Audio	AAC LC/LTP	X	X	Mono/estéreo con cualquier combinación estándar de frecuencia > 160 Kbps y ratios de muestreo de 8 a 48kHz	3GPP (.3gp) MPEG-4(.mp4) No soporta raw AAC (.aac) MPEG-TS (.ts)
	HE-AACv1	a partir 4.1	X		
	HE-AACv2		X		
	AAC ELD	a partir 4.1	a partir 4.1	Mono/estéreo, 16-8kHz	
	AMR-NB	X	X	4.75 a 12.2 Kbps muestreada a @ 8kHz	3GPP (.3gp)
	AMR-WB	X	X	9 ratios de 6.60 Kbps a 23.85 Kbps a @ 16kHz	3GPP (.3gp)
	MP3		X	Mono/estéreo de 8 a 320 Kbps, bit rate constante (CBR) o variable (VBR)	MP3 (.mp3)
	MIDI		X	MIDI tipo 0 y 1. DLS v1 y v2. XMF y XMF móvil. Soporte para tonos de llamada RTTTL	Tipo 0 y 1 (.mid, .xmf, .mxmf). RTTTL / RTX

				/ RTX, OTA y iMelody.	(.rttl, .rtx), OTA (.ota) iMelody (.imy)
	Ogg Vorbis		X		Ogg (.ogg) Matroska (.mkv a partir 4.0)
	FLAC		a partir 3.1	mono/estereo (no multicanal)	FLAC (.flac)
	PCM/WAVE	a partir 4.1	X	8 y 16 bits PCM lineal (frecuencias limitadas por el hardware)	WAVE (.wav)
Imagen	JPEG	X	X	Base + progresivo	JPEG (.jpg)
	GIF		X		GIF (.gif)
	PNG	X	X		PNG (.png)
	BMP		X		BMP (.bmp)
	WEBP	a partir 4.0	a partir 4.0		WebP (.webp)
Video	H.263	X	X		3GPP (.3gp)

					MPEG-4 (.mp4)
H.264 AVC	a partir 3.0	X	Baseline Profile (BP)	3GPP (.3gp) MPEG-4 (.mp4)	
MPEG-4 SP		X		3GPP (.3gp)	
VP8	a partir 4.3	a partir 2.3.3	Streaming a partir 4.0	WebM (.webm) Matroska (.mkv)	

Formatos multimedia soportados en Android.

Los tres pilares de la seguridad en Android

video [Seguridad en Android](#)

video: [La firma digital](#)

Objetivos:

Estudiar los principios en los que se basa la seguridad en Android.

La seguridad es un aspecto clave de todo sistema. Si nos descargáramos una aplicación maliciosa de Internet o de Google Play Store, esta podría leer nuestra lista de contactos, averiguar nuestra posición GPS, mandar toda esta información por Internet y terminar enviando 50 mensajes SMS.

En algunas plataformas antiguas, como Windows Mobile, estábamos prácticamente desprotegidos ante aplicaciones maliciosas. Por lo tanto, los usuarios tenían que ser muy cautos antes de instalar una aplicación.

En otras plataformas, como en iOS, toda aplicación ha de ser validada por Apple antes de poder ser instalada en un terminal. Además, solo está permitido instalar aplicaciones de la tienda oficial de Apple. Esto limita a los pequeños programadores y da un poder excesivo a Apple. Se trata de un planteamiento totalmente contrario al *software libre*.

Android propone un esquema de seguridad que protege a los usuarios, sin la necesidad de imponer un sistema centralizado y controlado por una única empresa. La seguridad en Android se fundamenta en los tres pilares siguientes:

- - Como se ha comentado en el primer capítulo Android está basado en Linux, por lo tanto, vamos a poder aprovechar la seguridad que incorpora este sistema operativo. De esta forma Android puede impedir que las aplicaciones tengan acceso directo al *hardware* o interfieran con recursos de otras aplicaciones.
- - Toda aplicación ha de ser firmada con un certificado digital que identifique a su autor. La firma digital también nos garantiza que el fichero de la aplicación no ha sido modificado. Si se desea modificar la aplicación estará destinada a ser firmada de nuevo, y esto solo podrá hacerlo el propietario de la clave privada.
- - Si queremos que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad del sistema hemos de utilizar un modelo de permisos, de forma que el usuario conozca los riesgos antes de instalar la aplicación.

En los siguientes apartados se describe con más detalle el primer y tercer punto. El proceso de firmar una aplicación será descrito en el último capítulo. Si no estás familiarizado con este concepto te recomendamos que veas el siguiente vídeo:

[La firma digital](#)

video: [La firma digital](#)

Usuario Linux y acceso a ficheros

Objetivos:

Mostrar el esquema de permisos para acceder a ficheros y en que consiste el usuario Linux de una aplicación.

Cada aplicación Android va a ser ejecutada en un proceso Linux independiente. Esto va a limitar su acceso directo al hardware y que pueda interferir con otras aplicaciones. Es lo que se conoce como ejecución en caja de arena.

Para impedir que otras aplicaciones puedan acceder a los ficheros creados por nuestra aplicación, Android crea una cuenta de usuario Linux (*user ID*) nueva por cada paquete (.apk) instalado en el sistema. Este usuario se crea cuando se instala la aplicación y permanece hasta que la aplicación es desinstalada.

Cualquier dato almacenado por la aplicación será asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones. No obstante, cuando crees un fichero puedes usar los modos MODE_WORLD_READABLE y/o MODE_WORLD_WRITEABLE para permitir que otras aplicaciones puedan leer o escribir en el fichero. Aunque otras aplicaciones puedan escribir el fichero, el propietario siempre será el usuario asignado a la aplicación que lo creó.

Dado que las restricciones de seguridad se garantizan a nivel de proceso, el código de dos paquetes no puede, normalmente, ejecutarse en el mismo proceso. Para que dos aplicaciones se ejecuten en un mismo proceso, sería necesario usar el mismo usuario. Puedes utilizar el atributo sharedUserId en AndroidManifest.xml para asignar un mismo usuario Linux a dos aplicaciones. Con esto conseguimos que a efectos de seguridad ambas aplicaciones sean tratadas como una sola. Por razones de seguridad, ambas aplicaciones han de estar firmadas con el mismo certificado digital.

Los permisos en Android

video [Los permisos en Android](#)

Objetivos:

Describir el esquema de permisos en Android.

Para proteger ciertos recursos y características especiales, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Cuando el usuario instala una aplicación este podrá examinar la lista de permisos que solicita la aplicación y decidir si considera oportuno instalar dicha aplicación. A partir de la versión 6, Android clasifica los permisos en peligrosos y normales. Como veremos en el siguiente apartado, a partir de esta versión, el usuario va a poder conceder o retirar los permisos peligrosos en cualquier momento. A continuación se muestra una lista con todos los permisos que pueden solicitar nuestras aplicaciones.

PERMISOS PELIGROSOS:



Almacenamiento Externo:

- [WRITE_EXTERNAL_STORAGE](#)– Modificar/eliminar almacenamiento USB (API 4). Permite el borrado y la modificación de archivos en la memoria externa. Lo ha de solicitar toda aplicación que necesite escribir un fichero en la memoria externa; por ejemplo, exportar datos en XML. Pero al permitirlo también podrán modificar/eliminar ficheros externos creados por otras aplicaciones.
- [READ_EXTERNAL_STORAGE](#)– Leer almacenamiento USB (API 16). Permite leer archivos en la memoria externa. Este permiso se introdujo en la versión 4.1. En versiones anteriores todas las aplicaciones pueden leer en la memoria externa. Por lo tanto, has de tener cuidado con la información que dejas en ella.



Ubicación:

- [ACCESS_COARSE_LOCATION](#)–Localización no detallada (basada en red). Localización basada en telefonía móvil (*Cell-ID*) y Wi-Fi. Aunque en la actualidad esta tecnología suele ofrecernos menos precisión que el GPS, no siempre es así. Por ejemplo, se está aplicando en el interior de aeropuertos y museos con precisiones similares.
- [ACCESS_FINE_LOCATION](#)–Localización GPS detallada. Localización basada en satélites GPS. Al dar este permiso también estamos permitiendo la localización basada en telefonía móvil y Wi-Fi ([ACCESS_COARSE_LOCATION](#)).



Teléfono:

- [CALL_PHONE](#)—Llamar a números de teléfono directamente **Servicios por los que tienes que pagar**. Permite realizar llamadas sin la intervención del usuario. Nunca solicites este permiso en tus aplicaciones, muchos usuarios no instalarán tu aplicación. Si has de realizar una llamada, es mejor realizarla por medio de una intención. A diferencia de la llamada directa, no necesitas ningún permiso, dado que el usuario ha de pulsar el botón de llamada para que comience.
- [READ_PHONE_STATE](#)—Consultar identidad y estado del teléfono. Muchas aplicaciones, como los juegos, piden este permiso para ponerse en pausa cuando recibes una llamada. Sin embargo, también permite el acceso al número de teléfono, IMEI (identificador de teléfono GSM), IMSI (identificador de tarjeta SIM) y al identificador único de 64 bits que Google asigna a cada terminal. Incluso si hay una llamada activa, podemos conocer el número al que se conecta la llamada.
- [READ_PHONE_NUMBERS](#)—Leer los números almacenados en el dispositivo. Está incluido en las capacidades del permiso READ_PHONE_STATE pero se permite su utilización independiente en las Instant Apps.
- [READ_CALL_LOG y WRITE_CALL_LOG](#)—Leer y modificar el registro de llamadas telefónicas. Como realizar estas acciones se describe al final del capítulo 9.
- [ADD_VOICEMAIL](#)—Añadir mensajes de voz. Permite crear nuevos mensajes de voz en el sistema.
- [USE_SIP](#)—Usar Session Initial Protocol. (API 9). Permite a tu aplicación usar el protocolo SIP.
- [PROCESS_OUTGOING_CALLS](#)—Procesar llamadas salientes. Permite a la aplicación controlar, modificar o abortar las llamadas salientes.
- [ANSWER_PHONE_CALLS](#)—Contestar llamadas entrantes.



Mensajes de texto (SMS):

- [SEND_SMS](#)—Enviar mensaje SMS **Servicios por los que tienes que pagar**. Permite la aplicación mandar de texto SMS sin la validación del usuario. Por iguales razones que [CALL_PHONE](#), a no ser que tu aplicación tenga que mandar SMS sin la intervención del usuario, resulta más conveniente enviarlos por medio de una intención.
- [RECEIVE_SMS](#)—Recibir mensajes de texto. Permite a la aplicación recibir y procesar SMS. Una aplicación puede modificar o borrar los mensajes recibidos
- [READ_SMS](#)—Leer mensajes de texto. Permite a la aplicación leer los mensajes SMS entrantes.
- [RECEIVE_MMS](#)—Recibir mensajes MMS. Permite monitorizar los mensajes multimedia entrantes, pudiendo acceder a su contenido.
- [RECEIVE_WAP_PUSH](#)—Recibir mensajes WAP Push. Permite monitorizar los mensajes WAP Push entrantes. Un mensaje WAP PUSH es un tipo de SMS que se usa para acceder de manera sencilla a una página WAP en lugar de teclear su dirección URL en el navegador.



Contactos:

- [READ_CONTACTS](#) – Leer datos de contactos. Permite leer información sobre los contactos almacenados (nombres, correos electrónicos, números de teléfono). Algunas aplicaciones podrían utilizar esta información de forma no lícita
- [WRITE_CONTACTS](#) – Escribir datos de contactos. Permite modificar los contactos.
- [GET_ACCOUNTS](#) – Obtener Cuentas. Permiten acceder a la lista de cuentas en el Servicio de Cuentas[1].



Calendario:

- [READ_CALENDAR](#) – Leer datos de contactos. Permite leer información del calendario del usuario.
- [WRITE_CONTACTS](#) – Escribir datos de contactos. Permite escribir en el calendario, pero no leerlo.



Camara:

- [CAMARA](#) – Hacer fotos / grabar vídeos. Permite acceso al control de la cámara y a la toma de imágenes y vídeos. El usuario puede no ser consciente.



Microfono:

- [RECORD_AUDIO](#) – Grabar audio. Permite acceso grabar sonido desde el micrófono del teléfono.



Sensores corporales :

- [BODY_SENSORS](#) – Leer sensores corporales. Da acceso a los datos de los sensores que están monitorizando el cuerpo del usuario. Por ejemplo, el lector de ritmo cardiaco.

PERMISOS NORMALES:



Comunicaciones:

- [INTERNET](#) – Acceso a Internet sin límites. Permite establecer conexiones a través de Internet. Este es un permiso muy importante, en el que hay que fijarse a quién se otorga. La mayoría de las aplicaciones lo piden, pero no todas lo necesitan. Cualquier malware necesita una conexión para poder enviar datos de nuestro dispositivo.
- [ACCESS_NETWORK_STATE](#) – Ver estado de red. Información sobre todas las redes. Por ejemplo para saber si tenemos conexión a internet.
- [CHANGE_NETWORK_STATE](#) – Cambiar estado de red. Permite cambiar el estado de conectividad de redes.
- [NFC](#) – Near field communication. (API 19) Algunos dispositivos disponen de un trasmisor infrarrojo para el control remoto de electrodomésticos.

- [TRASMIT_R](#) – Trasmitir por infrarrojos. (API 19) Algunos dispositivos disponen de un trasmisor infrarrojo para el control remoto de electrodomésticos.



Conección WIFI:

- [ACCESS_WIFI_STATE](#) – Ver estado de Wi-Fi. Permite conocer las redes Wi-Fi disponibles.
- [CHANGE_WIFI_STATE](#) – Cambiar estado de Wi-Fi. Permite cambiar el estado de conectividad Wi-Fi.
- [CHANGE_WIFI_MULTICAST_STATE](#) – Cambiar estado multicast Wi-Fi (API 4). Permite pasar al modo Wi-Fi Multicast.



Bluetooth:

- [BLUETOOTH](#) – Crear conexión Bluetooth. Permite a una aplicación conectarse con otro dispositivo Bluetooth. Antes ambos dispositivos han de emparejarse
- [BLUETOOTH_ADMIN](#) – Emparejar Bluetooth. Permite descubrir y emparejarse con otros dispositivos Bluetooth.



Consumo de batería:

- [WAKE_LOCK](#) – Impedir que el teléfono entre en modo de suspensión. Para algunas aplicaciones, como un navegador GPS, puede ser importante que no sean suspendidas nunca. Realmente, a lo único que puede afectar es a nuestra batería.
- [FLASHLIGHT](#) – Linterna. Permite encender el flash de la cámara.
- [VIBRATE](#) – Control de la vibración. Permite hacer vibrar al teléfono. Los juegos suelen utilizarlo.



Aplicaciones:

- [RECEIVE_BOOT_COMPLETED](#) – Ejecución automática al encender el teléfono. Permite a una aplicación recibir el anuncio broadcast ACTION_BOOT_COMPLETED enviado cuando el sistema finaliza un inicio. Gracias a esto la aplicación pondrá ponerse en ejecución al arrancar el teléfono.
- [BROADCAST_STICKY](#) – Enviar anuncios broadcast permanentes. Un broadcast permanente llegará a los receptores de anuncios que actualmente estén escuchando, pero también a los que se instancien en un futuro. Por ejemplo, el sistema emite el anuncio broadcast ACTION_BATTERY_CHANGED de forma permanente. De esta forma, cuando se llama a registerReceiver() se obtiene la intención de la última emisión de este anuncio. Por lo tanto, puede usarse para encontrar el estado de la batería sin necesidad de esperar a un futuro cambio en su estado. Se ha incluido este permiso dado que las aplicaciones mal intencionadas pueden ralentizar el dispositivo o volverlo inestable al demandar demasiada memoria.
- [KILL_BACKGROUND_PROCESSES](#) – Matar procesos en Background(API 9). Permite llamar a killBackgroundProcesses(String). Al hacer esta llamada el sistema mata de inmediato a todos los procesos de fondo asociados con el paquete indicado. Es el mismo

método que usa el sistema cuando necesita memoria. Estos procesos serán reiniciados en el futuro, cuando sea necesario.

- [REORDER_TASKS](#) – Reordenar tareas. Permite a una aplicación cambiar el orden de la lista de tareas.
- [INSTALL_SHORTCUT](#) y [UNINSTALL_SHORTCUT](#)– Instalar y desinstalar acceso directo(API 19). Permite a una aplicación añadir o eliminar un acceso directo a nuestra aplicación en el escritorio.
- [GET_PACKAGE_SIZE](#) – Obtener tamaño de un paquete. Permite a una aplicación conocer el tamaño de cualquier paquete.
- [EXPAND_STATUS_BAR](#) – Expandir barra de estado. Permite a una aplicación expandir o contraer la barra de estado
- [FOREGROUND_SERVICE](#) – Crear servicios en primer plano. (API 28). Permite a una aplicación crear servicios en primer plano.



Configuraciones del sistema:

- [SET_WALLPAPER](#) – Poner fondo de pantalla. Permite establecer fondo de pantalla en el escritorio.
- [SET_WALLPAPER_HINTS](#) – Sugerencias de fondo de pantalla. Permite a las aplicaciones establecer sugerencias de fondo de pantalla.
- [SET_ALARM](#) – Establecer Alarma. Permite a la aplicación enviar una intención para poner una alarma o temporizador en la aplicación Reloj.
- [SET_TIME_ZONE](#) – Cambiar zona horario. Permite cambiar la zona horaria del sistema.
- [ACCESS_NOTIFICATION_POLICY](#) – Acceso a política de notificaciones (API 23). Permite conocer la política de notificaciones del sistema.



Audio:

- [MODY_AUDIO_SETTINGS](#)– Cambiar ajustes de audio. Permite cambiar ajustes globales de audio, como el volumen.



Sincronización:

- [READ_SYNC_SETTINGS](#) – Leer ajustes de sincronización. Permite saber si tienes sincronización en segundo plano con alguna aplicación (como con un cliente de Twitter o Gmail).
- [WRITE_SYNC_SETTINGS](#) – Escribir ajustes de sincronización. Permite registrar tu aplicación como adaptador de sincronización (SyncAdapter).
- [READ_SYNC_STATS](#) – Leer estadísticas de sincronización.



Ubicación:

- ACCESS_LOCATION_EXTRA_COMMANDS – Mandar comandos extras de localización. Permite a una aplicación acceder a comandos adicionales de los proveedores de localización. Por ejemplo, tras pedir este permiso podríamos enviar el siguiente comando al GPS, con el método: sendExtraCommand("gps", "delete_aiding_data", null);.



Seguridad:

- USE_FINGERPRINT – Usar huella digital(API 23). Permite usar el hardware de reconocimiento de huella digital.
- DISABLE_KEYGUARD – Deshabilitar bloqueo de teclado. Permite a las aplicaciones desactivar el bloqueo del teclado si no es seguro.

NOTA: Los permisos peligrosos pertenecen a uno de los 9 grupos anteriores. Estos grupos son importantes dado que el usuario concede o deniega el permiso a un grupo entero. Por el contrario, a partir de la versión 6.0 los permisos normales ya no se clasifican en grupos. Se han organizado en este texto por grupos para una mejor organización.

NOTA: Existen otros permisos que no han sido incluidos en esta lista dado que no podemos solicitarlos en nuestras aplicaciones al estar reservados para aplicaciones del sistema.

Para solicitar un determinado permiso en tu aplicación, no tienes más que incluir una etiqueta <uses-permission> en el fichero AndroidManifest.xml de tu aplicación. En el siguiente ejemplo se solicitan dos permisos:

```
<manifest package="org.example.mi_aplicacion" >
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    ...
</manifest>
```

Permisos en Android 6 Marshmallow

video [Permisos en Android 6.0 Marshmallow](#)

video [Trabajando con permisos en Android 6.0](#)

Objetivos:

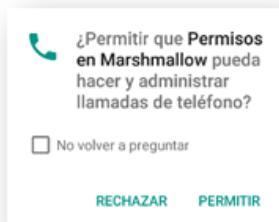
- Conocer cómo funcionan los permisos en Android 6 Marshmallow

Permisos en Android 6 Marshmallow

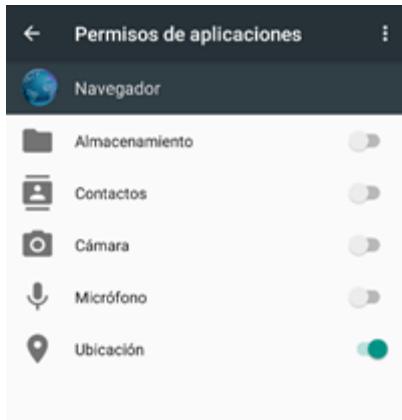
Marcar esta página

En un dispositivo con versión de Android anterior a Marshmallow un usuario concede los permisos a una aplicación en el momento de la instalación. Si no está de acuerdo con algún permiso, la única alternativa para el usuario es no instalar la aplicación. Una vez instalada la aplicación, puede realizar las acciones asociadas a estos permisos tantas veces como desee y cuando desee. Esta forma de trabajar dejaba a los usuarios indefensos ante posibles abusos. Por ejemplo, si queremos utilizar WhatsApp o jugar a Apalabradados tenemos que aceptar la larga lista de permisos innecesarios que nos solicitan. El usuario se resigna y acaba aceptando prácticamente cualquier permiso.

En la versión 6 se introducen importantes novedades a la hora de conceder los permisos a las aplicaciones. En primer lugar los permisos son divididos en normales y peligrosos. A su vez los permisos peligrosos se dividen en 9 grupos: almacenamiento, localización, teléfono, SMS, contactos, calendario, cámara, micrófono y sensor de ritmo cardíaco. En el proceso de instalación el usuario da el visto bueno a los permisos normales, de la misma forma como se hacía en la versión anterior. Por el contrario, los permisos peligrosos no son concedidos en la instalación. La aplicación consultará al usuario si quiere conceder un permiso peligroso en el momento de utilizarlo:



Además se recomienda que la aplicación indique para qué lo necesita. De esta forma el usuario tendrá más elementos de juicio para decidir si da o no el permiso. Si el usuario no concede el permiso la aplicación ha de tratar de continuar el proceso sin este permiso. Otro aspecto interesante es que el usuario podrá configurar en cualquier momento qué permisos concede y cuáles no. Por ejemplo, podemos ir al administrador de aplicaciones y seleccionar la aplicación Navegador. En el apartado permisos se nos mostrará los grupos de permisos que podemos conceder:



Observa como de los grupos de permisos solicitados, en este momento solo concedemos el permiso de Ubicación.

El usuario concede o rechaza los permisos por grupos. Si en el manifiesto se ha pedido leer y escribir en la SD, concedemos los dos permisos o ninguno. Es decir, no podemos conceder permiso de lectura, pero denegar el de escritura.

Para reforzar los conceptos que acabamos de exponer es recomendable que hagas el siguiente ejercicio:

Ejercicio: Trabajando con permisos en Android Marshmallow

1. Crea un nuevo proyecto con los siguientes datos:

Phone and Tablet / Basic Activity

Name: Permisos en Marshmallow

Package name: org.example.permisosenmarshmallow

Language: Java ó Kotlin

Minimum API level: API 19 Android 4.4 (KitKat)

2. En el método `onCreate()` de `MainActivity` elimina las líneas tachadas, y en su lugar, añade las subrayadas.

```
public void onClick(View view) {  
    borrarLlamada();  
    Snackbar.make(view, "Replace with your ...", Snackbar.LENGTH_LONG)  
        .setAction("Action", null).show();  
}
```

3. En la etiqueta `<ConstraintLayout>` de `ccontent_main.xml` añade:

`android:id="@+id/vista_principal"`

4. En Java declara la siguiente variable al principio de la clase:

```
private View vista_principal;
```

En el método `onCreate()` añade:

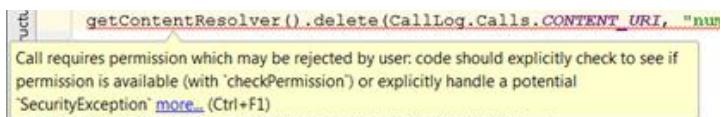
```
vista = findViewById(R.id.content_main);
```

5. Añade el siguiente metodo:

```
void borrarLlamada() {  
    getContentResolver().delete(CallLog.Calls.CONTENT_URI,  
                               "number='555555555'", null);  
    Snackbar.make(vista_principal, "Llamadas borradas del registro.",  
                  Snackbar.LENGTH_SHORT).show();  
}
```

Como se describirá en el capítulo 9, este código elimina del registro de llamadas del teléfono todas las llamadas cuyo número sea 555555555. La segunda línea muestra un cuadro de texto tipo *Snackbar* para avisar que la acción se ha realizado.<

6. Observa cómo el sistema nos advierte de que estamos actuando de forma no correcta:



7. Ignora esta advertencia y ejecuta el proyecto. Si pulsas en el botón flotante, aparecerá el siguiente error:

Se ha detenido la aplicación Permisos en
Marshmallow.

[ACEPTAR](#)

8. Abre el Log cat para verificar la causa del error:

```
Caused by: java.lang.SecurityException: Permission Denial: opening provider  
com.android.providers.contacts.CallLogProvider from ... requires  
android.permission.READ_CALL_LOG or android.permission.WRITE_CALL_LOG
```

Es decir, la aplicación se ha detenido porque está realizando una acción que requiere de la solicitud de un permiso

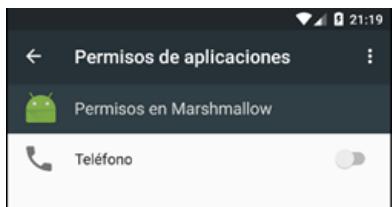
9. Añade en AndroidManifest.xml:

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

10. Si ejecutas de nuevo el proyecto en un dispositivo con una versión anterior a la 6.0, podrás verificar que ya no se produce el error.

11. Si ejecutas ahora en un dispositivo con versión 6.0 o superior (si no dispones de uno utiliza un emulador), observarás que el error continúa.

12. Para entender lo que ha ocurrido, ve a *Ajustes / Aplicaciones / Permisos en Marshmallow / Permisos*:



Desde aquí podrás configurar los permisos peligrosos que quieras otorgar a la aplicación. Observa como el grupo de permisos referentes al teléfono está desactivado. Cuando instalamos una aplicación no se le concede ningún permiso peligroso.

13. Activa el permiso:



14. Vuelve a ejecutar la aplicación y verificar que ya no se produce el error:

Como acabamos de comprobar la aplicación anterior va a funcionar correctamente en dispositivos con una versión anterior a la 6.0. Sin embargo, cuando se ejecute en las nuevas versiones, se producirá un error. Aunque hemos visto cómo el usuario puede evitarlo, no es desde luego la forma correcta de trabajar.

A partir de Android Marshmallow trabajar con acciones que necesiten de un permiso va a suponer un esfuerzo adicional para el programador. Antes de realizar la acción tendremos que verificar si tenemos el permiso. En caso negativo hay que exponer al usuario para qué lo queremos y pedírselo. Si el usuario no nos diera el permiso, tendremos qué decidir qué hacer. ¿Podemos realizar la acción solicitada aunque no dispongamos de cierta información? ¿Dejamos de hacer la acción solicitada? ¿O salimos de la aplicación? En el siguiente ejercicio veremos cómo realizar esta tarea.

Ejercicio:*Solicitud de permisos en Android Marshmallow*

- 1.** El primer paso va a ser verificar que tenemos el permiso adecuado antes de realizar una acción que lo requiera. Resulta sencillo, simplemente has de añadir el if que se muestra a continuación en `borrarLlamada()`:

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission
        .WRITE_CALL_LOG) == PackageManager.PERMISSION_GRANTED) {
    getContentResolver().delete(CallLog.Calls.CONTENT_URI,
        "number='5555555555'", null);
    Snackbar.make(vista_principal, "Llamadas borradas del registro.",
        Snackbar.LENGTH_SHORT).show();
}
```

- 2.** Ejecuta de nuevo la aplicación en un dispositivo con versión 6.0 o superior y verifica que ya no se produce el error.

- 3.** Esto no resuelve el problema. Nuestra aplicación no puede limitarse a no realizar la acción cuando no disponga del permiso. Ha de avisar al usuario y solicitar el permiso. Para ello añade una sección `else` a el if anterior.>

```
if (ContextCompat.checkSelfPermission(this,
        Manifest.permission.WRITE_CALL_LOG)
    == PackageManager.PERMISSION_GRANTED) {
    ...
} else {
```

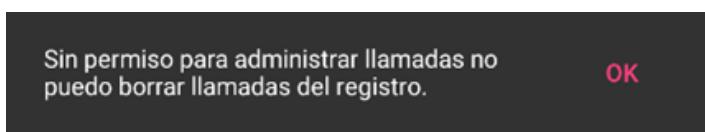
```
solicitarPermiso(Manifest.permission.WRITE_CALL_LOG, "Sin el permiso"+  
    " administrar llamadas no puedo borrar llamadas del registro.",  
    SOLICITUD_PERMISO_WRITE_CALL_LOG, this);  
}
```

4. Añade el siguiente método:

```
public static void solicitarPermiso(final String permiso, String  
    justificacion, final int requestCode, final Activity actividad) {  
    if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,  
        permiso)){  
        new AlertDialog.Builder(actividad)  
            .setTitle("Solicitud de permiso")  
            .setMessage(justificacion)  
            .setPositiveButton("Ok", new DialogInterface.OnClickListener() {  
                public void onClick(DialogInterface dialog, int whichButton) {  
                    ActivityCompat.requestPermissions(actividad,  
                        new String[]{permiso}, requestCode);  
                }  
            }).show();  
    } else {  
        ActivityCompat.requestPermissions(actividad,  
            new String[]{permiso}, requestCode);  
    }  
}
```

Es posible que tengas que solicitar permisos desde diferentes puntos de la aplicación. Por esta razón se ha declarado este método público y estático. Además, se ha pasado a parámetros toda la información que necesita: el permiso a solicitar, la justificación de porque lo necesitamos, un código de solicitud y la actividad que recogerá la respuesta. Una vez el usuario decide si da el permiso, se llamará al método onRequestPermissionsResult(), que tendrás que declarar en la actividad que se pasa en el cuarto parámetro. El código es un valor numérico que permitirá identificar diferentes solicitudes.

Android nos recomienda que indiquemos al usuario para qué le estamos solicitando el permiso. Si consideras que no es necesario, puedes eliminar la primera parte del método y dejar solo el código que aparece dentro del else. Antes de mostrar la explicación usando un AlertDialog, se verifica en el if si interesa mostrar esta información. Si el usuario ha indicado que no nos da el permiso y además ha marcado la casilla de que no quiere que volvamos a preguntar, no es conveniente insistir. El sistema se encarga de recordar esta información, nosotros simplemente tenemos que usar el método shouldShowRequestPermissionRationale().



NOTA: *Este código se ejecuta en el hilo principal, por lo tanto nunca utilices un método para preguntar al usuario que pueda bloquear el hilo. Observa como en el ejemplo se utilizan llamadas asíncronas.*

El trabajo más importante lo hace el método requestPermissions() que muestra un cuadro de diálogo como el siguiente y registra el permiso según la respuesta del usuario:



5. Una vez que el usuario escoja se realizará una llamada a `onRequestPermissionsResult()`. Aquí podremos procesar la respuesta. Añade el siguiente método:

```
@Override public void onRequestPermissionsResult(int requestCode,
                                                 String[] permissions, int[] grantResults) {
    if (requestCode == SOLICITUD_PERMISO_WRITE_CALL_LOG) {
        if (grantResults.length == 1 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            borrarLlamada();
        } else {
            Toast.makeText(this, "Sin el permiso, no puedo realizar la " +
                               "acción", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Este método ha de estar declarado en una actividad. En caso de que el usuario nos conceda el permiso, tenemos que volver a realizar la acción que no pudo realizarse (en el ejemplo, `borrarLlamada()`). En caso de que hayas solicitado el permiso desde diferentes acciones o que hayas solicitado diferentes permisos, el valor de `requestCode` permitirá diferenciar cada caso.

6. Declara la siguiente variable al principio de la clase:

```
private static final int SOLICITUD_PERMISO_WRITE_CALL_LOG = 0;
```

7. Verifica que la aplicación funciona correctamente.

Práctica: Solicitud de permisos en Mis Lugares

En el ejercicio Añadiendo fotografías desde la galería era imprescindible disponer del permiso para acceder a la memoria externa. Para poder ejecutar la aplicación en un dispositivo con una versión de Android 6 o superior, tuvimos que conceder este permiso manualmente. Un usuario final no debe realizar esta solicitud desde Ajustes. Va a ser imprescindible, solicitar este permiso desde la aplicación

Cuando el usuario intente obtener una fotografía desde la galería o tomar una fotografía, solicita el permiso `READ_EXTERNAL_STORAGE` tal y como se ha realizado en el ejercicio anterior

Unidad 7 : Posicionamiento y Mapas

Introducción a la unidad

Objetivos:

- - Describir los diferentes sistemas de posicionamiento disponibles en los dispositivos móviles actuales.
 - - Describir las APIs de Android para la geolocalización.
 - - Mostrar varias estrategias para elegir un proveedor de localización.
 - - Ver lo sencillo que resulta incorporar en nuestra aplicación un servicio de un tercero. En concreto mostraremos mapas con Google Maps.
-

En esta unidad se describe el API que incorpora Android para permitir conocer la posición geográfica del dispositivo. Estos servicios se basan principalmente en el GPS, pero también disponemos de novedosos servicios de localización basados en telefonía móvil y redes Wi-Fi. A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán aprender a utilizar estas funciones.

Terminamos la unidad describiendo como podemos incorporar a nuestra aplicación servicios realizados por terceros. En concreto instalaremos una vista que permite representar un mapa de Google Maps.

Sistemas de geolocalización en dispositivos móviles

Objetivos:

Mostrar las posibilidades y alternativas de los sistemas de localización en los dispositivos móviles.

[video Sistema de geolocalización en móviles](#) [video Los sistemas de posicionamiento global por satélite](#)

Los Sistemas de posicionamiento global por Satélite (GPS)

Objetivos:

Describir cómo funcionan los sistemas de posicionamiento global por satélite (GPS) y los sistemas DGPS y AGPS.

El API de Localización de Android

Objetivos:

Describir cómo utilizar el API de localización en Android.

video[Tutorial] [La API de localización de Android](#)

Ejercicio: *El API de localización de Android.*

En este ejercicio crearemos una aplicación que es capaz de leer información de localización del dispositivo y actualizarla cada vez que se produce un cambio.

1. Crea un nuevo proyecto con los siguientes datos:

ApplicationName: Localizacion

Package Name: org.example.localizacion

RPhone and Tablet

Minimum SDK: API 15 Android 4.0.3 (IceCreamSandwich)

Add an activity: *Empty Activity*

2. Por razones de privacidad acceder a la información de localización está en principio prohibido a las aplicaciones. Si estas desean hacer uso de este servicio han de solicitar el permiso adecuado. En concreto hay que solicitar ACCESS_FINE_LOCATION para acceder a cualquier tipo de sistema de localización o ACCESS_COARSE_LOCATION para acceder al sistema de localización basado en redes. Añade la siguiente línea en el fichero *AndroidManifest.xml* dentro de la etiqueta <manifest>:

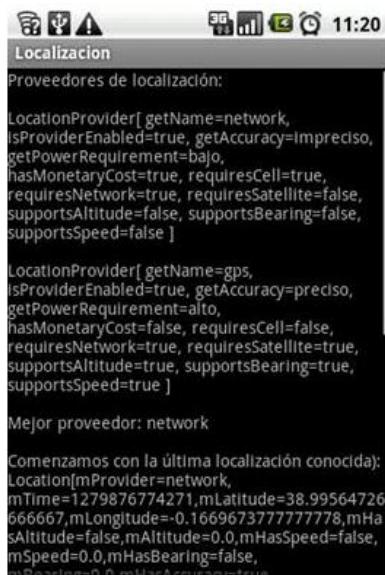
```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Por lo tanto en este ejemplo vamos autilizar, tanto la localización fina, que nos proporciona el GPS, como una menos precisa,que nos proporcionada las torres de telefonía celular y las redes WiFi.

3. Sustituye el fichero res/layout/activity_main.xml por:

```
<ScrollView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <TextView  
        android:id="@+id/salida"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content" />  
</ScrollView>
```

En este ejemplo nos limitaremos a mostrar en modo de texto la información obtenida desde el API de localización. Para ello usaremos un TextView dentro de un ScrollView, tal y como se muestra en la siguiente pantalla:



4. Abre la clase MainActivity y copia el siguiente código:

```

public class MainActivity extends AppCompatActivity
        implements LocationListener {
    static final long TIEMPO_MIN = 10 * 1000 ; // 10 segundos
    static final long DISTANCIA_MIN = 5; // 5 metros
    static final String[] A = { "n/d", "preciso", "impresiso" };
    static final String[] P = { "n/d", "bajo", "medio", "alto" };
    static final String[] E = { "fuera de servicio",
            "temporalmente no disponible", "disponible" };
    LocationManager manejador;
    String proveedor;
    TextView salida;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        salida = findViewById(R.id.salida);
        manejador = (LocationManager) getSystemService(LOCATION_SERVICE);
        log("Proveedores de localización: \n ");
        muestraProveedores();

        Criteria criterio = new Criteria();
        criterio.setCostAllowed(false);
        criterio.setAltitudeRequired(false);
        criterio.setAccuracy(Criteria.ACCURACY_FINE);
        proveedor = manejador.getBestProvider(criterio, true);
        log("Mejor proveedor: " + proveedor + "\n");
        log("Comenzamos con la última localización conocida:");
        Location localizacion = manejador.getLastKnownLocation(proveedor);
        muestraLocaliz(localizacion);
    }
}
  
```

La primera línea que nos interesa es la llamada a `getSystemService(LOCATION_SERVICE)` que crea el objeto manejador de tipo `LocationManager`. La siguiente línea hace una llamada al método `log()`, que se definirá más adelante. Simplemente muestra por el `TextView` `salida`, el

texto indicado. La siguiente llamada a `muestraProveedores()` también es un método definido por nosotros, que listará todos los proveedores de localización disponibles.

En las tres siguientes líneas vamos a seleccionar uno de estos proveedores de localización. Comenzamos creando un objeto de la clase `Criteria`, donde se podrá indicar las características que ha de tener el proveedor buscado. En este ejemplo indicamos que no ha de tener coste económico, ha de poder obtener la altura y ha de tener precisión fina. Para consultar otras restricciones, consultar documentación de la clase `Criteria`[\[1\]](#). Con estas restricciones parece que estamos interesados en el proveedor basado en GPS, aunque de no estar disponible, se seleccionará otro que cumpla el mayor número de restricciones. Para seleccionar el proveedor usaremos el método `getBestProvider()`. En este método hay que indicar el criterio de selección y un valor booleano, donde indicamos si solo nos interesa los sistemas que el usuario tenga actualmente habilitados. Nos devolverá un String con el nombre del proveedor seleccionado.

Algunos proveedores, como el GPS, pueden tardar un cierto tiempo en darnos una primera posición. No obstante, Android recuerda la última posición que fue devuelta por este proveedor. Es lo que nos devuelve la llamada a `getLastKnownLocation()`. El método `muestraLocaliz()` será definido más tarde y muestra en pantalla una determinada localización.

5. A continuación copia el resto del código:

```
// Métodos del ciclo de vida de la actividad

@Override protected void onResume() {
    super.onResume();
    manejador.requestLocationUpdates(proveedor, TIEMPO_MIN, DISTANCIA_MIN,
                                      this);
}

@Override protected void onPause() {
    super.onPause();
    manejador.removeUpdates(this);
}

// Métodos de la interfaz LocationListener
@Override public void onLocationChanged(Location location) {
    log("Nueva localización: ");
    muestraLocaliz(location);
}

@Override public void onProviderDisabled(String proveedor) {
    log("Proveedor deshabilitado: " + proveedor + "\n");
}

@Override public void onProviderEnabled(String proveedor) {
    log("Proveedor habilitado: " + proveedor + "\n");
}

@Override public void onStatusChanged(String proveedor, int estado,
                                    Bundle extras) {
    log("Cambia estado proveedor: " + proveedor + ", estado="
        + E[Math.max(0, estado)] + ", extras=" + extras + "\n");
}

// Métodos para mostrar información
private void log(String cadena) {
    salida.append(cadena + "\n");
```

```

}

private void muestraLocaliz(Location localizacion) {
    if (localizacion == null)
        log("Localización desconocida\n");
    else
        log(localizacion.toString() + "\n");
}

private void muestraProveedores() {
    log("Proveedores de localización: \n ");
    List<String> proveedores = manejador.getAllProviders();
    for (String proveedor : proveedores) {
        muestraProveedor(proveedor);
    }
}

private void muestraProveedor(String proveedor) {
    LocationProvider info = manejador.getProvider(proveedor);
    log("LocationProvider[ " + "getName=" + info.getName()
        + ", isEnabled="
        + manejador.isProviderEnabled(proveedor) + ", getAccuracy="
        + A[Math.max(0, info.getAccuracy())] + ", getPowerRequirement="
        + P[Math.max(0, info.getPowerRequirement())]
        + ", hasMonetaryCost=" + info.hasMonetaryCost()
        + ", requiresCell=" + info.requiresCell()
        + ", requiresNetwork=" + info.requiresNetwork()
        + ", requiresSatellite=" + info.requiresSatellite()
        + ", supportsAltitude=" + info.supportsAltitude()
        + ", supportsBearing=" + info.supportsBearing()
        + ", supportsSpeed=" + info.supportsSpeed() + " ]\n");
}
}

```

Para conseguir que se notifiquen cambios de posición hay que llamar al método `requestLocationUpdates()` y para indicar que se dejen de hacer las notificaciones hay que llamar a `removeUpdates()`. Dado que queremos ahorrar batería nos interesa que se reporten notificaciones solo cuando la aplicación esté activa. Por lo tanto tenemos que reescribir los métodos `onResume()` y `onPause()`.

El método `requestLocationUpdates()` dispone de 4 parámetros: el nombre del proveedor, el tiempo entre actualizaciones en ms (se recomienda valores mayores de 60.000 ms), la distancia mínima (de manera que si es menor, no se notifica) y un escuchador de eventos que implemente el interface `LocationListener`.

Como nuestra actividad es un `LocationListener` tenemos que implementar los siguientes métodos: `onLocationChanged()` se activará cada vez que se obtenga una nueva posición. Los otros tres métodos pueden ser usados para cambiar de proveedor en caso de que se active uno mejor o deje de funcionar el actual. Sería buena idea llamar de nuevo aquí al método `getBestProvider()`.

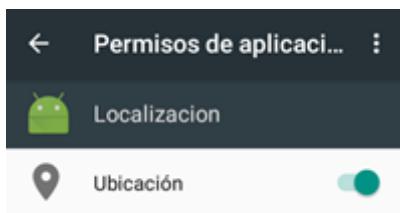
El resto del código resulta fácil de interpretar.

6. Observa cómo aparecen algunos errores. El sistema nos advierte de que estamos actuando de forma no correcta:

```
Location localizacion = manejador.getLastKnownLocation(proveedor);
```

Call requires permission which may be rejected by user: code should explicitly check to see if
permission is available (with 'checkPermission') or explicitly handle a potential 'SecurityException'
[more...](#) (Ctrl+F1)

7. Ignora esta advertencia y ejecuta el proyecto.
8. Si ejecutas el proyecto en un dispositivo con una versión 6.0 o superior, podrás verificar que se produce el error. Para evitar que se produzca, en el dispositivo accede a *Ajustes / Aplicaciones / Localización / Permisos*: y activa el permiso de *Ubicación*:



Vuelve a ejecutar la aplicación y verificar que ya no se produce el error.

NOTA: *Para aligerar el código del ejercicio no hemos incluido el código necesario para solicitar permiso a partir de la versión 6.0. Si vas a distribuir la aplicación, resulta impresindible realizar los pasos descritos en la sección Permisos en Android 6 Marshmallow.*

9. Verifica el funcionamiento del programa, si es posible con un dispositivo real con el GPS activado.

[1] <http://developer.android.com/reference/android/location/Criteria.html>

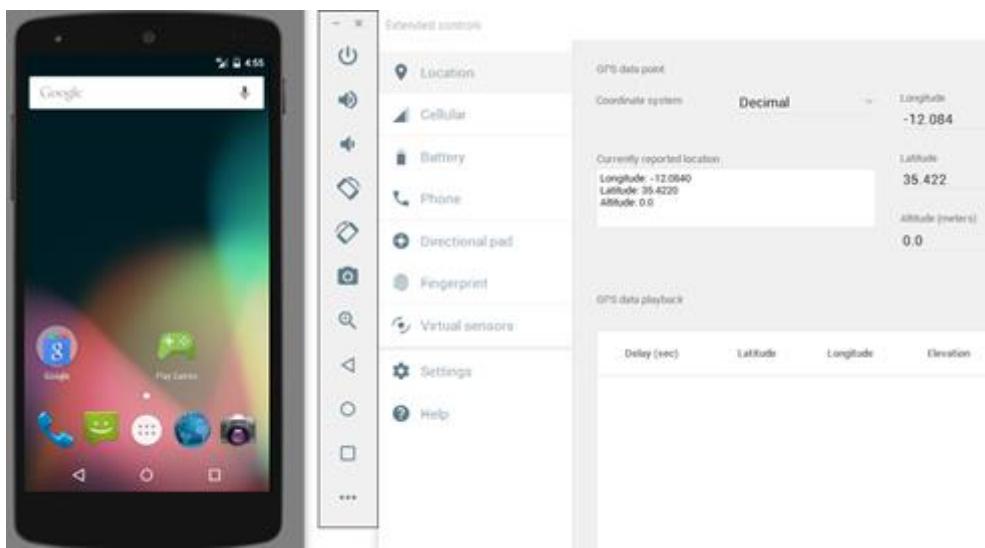
Emulación del GPS con Android Studio

Objetivos:

Aprender a emular el GPS conAndroid Studio.

Probar una aplicación de localización con un dispositivo real requiere que te desplaces para cambiar tu posición. Suele resultar más práctico probar este tipo de aplicaciones desde un emulador, ya que estos incorporan un sistema de emulación de posición GPS. Para abrir los controles extendidos de un emulador pulsa en los tres puntos que aparecen en la parte inferior de la barra de herramientas y selecciona la pestaña *Location*.

Desde aquí podremos enviar nuevas posiciones al dispositivo que está siendo emulado. El botón *LOAD GPX / KML* nos permiten realizar pruebas más complejas en nuestras aplicaciones de localización, sin necesidad de dar vueltas con el dispositivo en la mano. Un fichero GPX o KML registra una secuencia temporal de localizaciones. Existen muchos programas (Google Earth) que permiten grabar este tipo de ficheros. Luego podremos reproducir esta secuencia tantas veces como queramos hasta que nuestro programa funcione perfectamente.



Práctica:Emulación del GPS

Ejecuta el proyecto anterior en un emulador y prueba la emulación del GPS.

Estrategias de Localización en Android

video[Tutorial] [Estrategias de localizacion en Android.](#)

Objetivos:

Mostrar varias estrategias a la hora de utilizar el API de localización en Android.

Determinar cuál es el proveedor de localización idóneo para nuestra aplicación puede resultar una tarea compleja. Además esta decisión puede variar con el tiempo según el usuario cambie de posición, o desactivar alguno de los proveedores. A continuación se plantean tres posibles estrategias:

Usar siempre el mismo tipo de proveedor

Los dos proveedores de localización disponibles en Android tienen características muy diferentes. Muchas aplicaciones tienen algún tipo de requisito que hace que podamos decantarnos de entrada por un sistema en concreto. Veamos algunos ejemplos.

Usaremos GPS si:

- La aplicación requiere una precisión inferior a 10 m (ej. navegación).
- Está pensada para su uso al aire libre (ej. senderismo).

Usaremos localización por redes si:

- El consumo de batería es un problema.
- Está pensada para su uso en el interior de edificios (visita museo).

Una vez decidido, usaremos las constantes `GPS_PROVIDER` o `NETWORK_PROVIDER` de la clase `LocationManager` para indicar el proveedor deseado.

Existe un tercer tipo de proveedor identificado con la constante `PASSIVE_PROVIDER`. Puedes usarlo si quieras observar pasivamente actualizaciones de ubicación provocadas por otras aplicaciones, pero no quieras que se lancen nuevas lecturas de posición. De esta manera no provocamos consumo de energía adicional.

El mejor proveedor según un determinado criterio

Como vimos en el apartado anterior, el API de localización de Android nos proporciona la clase `Criteria` para seleccionar un proveedor de localización según el criterio indicado. Recordemos el código utilizado:

```
Criteria criterio = new Criteria();
criterio.setCostAllowed(false);
criterio.setAltitudeRequired(false);
criterio.setAccuracy(Criteria.ACCURACY_FINE);
proveedor = manejador.getBestProvider(criterio, true);
```

Los proveedores pueden variar de estado, por lo que podría ser interesante consultar cual es el mejor proveedor cada vez que cambie su estado.

Usar los dos proveedores en paralelo

Otra alternativa podría ser programar actualizaciones de los dos proveedores de localización disponibles. Luego podríamos seleccionar la mejor localización entre las suministradas. Para estudiar esta alternativa realiza el siguiente ejercicio:

Ejercicio: Añadiendo localización en Mis Lugares.

1. Añade en *AndroidManifest.xml* de Mis Lugares el siguiente permiso:

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

2. Crea a la clase CasosUsoLocalizacion con los siguientes atributos y su inicialización:

```
public class CasosUsoLocalizacion {  
    private static final String TAG = "MisLugares";  
    private Activity actividad;  
    private int codigoPermiso;  
    private LocationManager manejadorLoc;  
    private Location mejorLoc;  
    private GeoPunto posicionActual;  
    private AdaptadorLugares adaptador;  
  
    public CasosUsoLocalizacion(Activity actividad, int codigoPermiso) {  
        this.actividad = actividad;  
        this.codigoPermiso = codigoPermiso;  
        manejadorLoc = (LocationManager) getSystemService(LOCATION_SERVICE);  
        posicionActual = ((Aplicacion) actividad.getApplication())  
            .posicionActual;  
        adaptador = ((Aplicacion) actividad.getApplication()).adaptador;  
        ultimaLocalizacion();  
    }  
}
```

La variable manejadorLoc nos permite acceder a los servicios de localización de Android. La variable mejorLoc, de tipo Location, almacena la mejor localización actual. La variable posicionActual almacena la misma información, pero en formato GeoPunto. Estará almacenada en Aplicacion para que sea accesible desde cualquier parte de la aplicación. Es necesario tener la información en dos formatos, ya que la primera variable es usada para disponer de la fecha de obtención o proveedor que nos la ha dado y la segunda al ser el formato usado en el resto de la aplicación. Finalmente obtenemos una referencia al adaptador del RecyclerView para poder actualizarlo cuando haya cambios de localización.

3. En la clase Aplicacion crea la variable posicionActual:

```
public GeoPunto posicionActual = new GeoPunto(0.0, 0.0);
```

Los valores (0, 0) representa que no se dispone de localización.

4. En MainActivity, añade:

```
private static final int SOLICITUD_PERMISO_LOCALIZACION = 1;  
private CasosUsoLocalizacion usoLocalizacion;
```

```
@Override protected void onCreate(Bundle savedInstanceState) {  
    ...  
    usoLocalizacion = new CasosUsoLocalizacion(this,  
                                                SOLICITUD_PERMISO_LOCALIZACION);  
}
```

5. Vamos a verificar varias veces si el usuario nos ha dado permiso de localización. Para ello añade en CasosUsoLocalizacion el siguiente código:

```
public boolean hayPermisoLocalizacion() {  
    return (ActivityCompat.checkSelfPermission(  
        actividad, Manifest.permission.ACCESS_FINE_LOCATION)  
        == PackageManager.PERMISSION_GRANTED);  
}
```

6. Añade el siguiente método:

```
void ultimaLocalizacion(){  
    if (hayPermisoLocalizacion()) {  
        if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {  
            actualizaMejorLocaliz(manejadorLoc.getLastKnownLocation(  
                LocationManager.GPS_PROVIDER));  
        }  
        if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){  
            actualizaMejorLocaliz(manejadorLoc.getLastKnownLocation(  
                LocationManager.NETWORK_PROVIDER));  
        } else {  
            solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,  
                            "Sin el permiso localización no puedo mostrar la distancia"+  
                            " a los lugares.", codigoPermiso, actividad);  
        }  
    }  
}
```

Antes de obtener una localización se debe verificar que tenemos permiso para hacerlo. Para más información consultar Permisos en Android 6 Marshmallow. En caso de tener permiso buscamos la última localización disponible. Usamos el método getLastKnownLocation() aplicado a los dos proveedores que vamos a utilizar. El método actualizaMejorLocaliz() se explicará más adelante. Si no disponemos del permiso, lo solicitamos al usuario.

7. La función getLastKnownLocation() estará marcado con el error “*Call requiered permission ...*”, avisándonos que hemos de comprobar que tenemos permiso antes de hacer la llamada. Realmente lo hemos hecho. Para desactivar la advertencia añade @SuppressLint("MissingPermission") antes de la función.

8. Copia a esta clase el método solicitarPermiso() del ejercicio Permisos en Android 6 Marshmallow. Declara también la constante SOLICITUD_PERMISO_LOCALIZACION.

9. Una vez conteste el usuario se llamará a onRequestPermissionsResult de MainActivity. Añade a la clase los siguientes métodos:

```
@Override public void onRequestPermissionsResult(int requestCode,  
                                                String[] permissions, int[] grantResults) {
```

```

        if (requestCode == SOLICITUD_PERMISO_LOCALIZACION
            && grantResults.length == 1
            && grantResults[0] == PackageManager.PERMISSION_GRANTED)
            usoLocalizacion.permisoConcedido()

    }

```

Si el usuario contesta afirmativamente a la solicitud de permiso llamamos a un caso de uso para que se actúe en consecuencia.

10. Añade el siguiente caso de uso en CasosUsoLocalizacion:

```

public void permisoConcedido() {
    ultimaLocalizazion();
    activarProveedores();
    adaptador.notifyDataSetChanged();
}

private void activarProveedores() {
    if (hayPermisoLocalizacion()) {
        if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            manejadorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                20 * 1000, 5, this);
        }
        if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
            manejadorLoc.requestLocationUpdates(LocationManager
                .NETWORK_PROVIDER, 10 * 1000, 10, this);
        }
    } else {
        solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
            "Sin el permiso localización no puedo mostrar la distancia"+
            " a los lugares.", codigoPermiso, actividad);
    }
}

```

La primera función se llama cuando nos conceden permiso de localización. Miramos si ya disponen de una última posición conocida, llamamos a la segunda función que activa los eventos de localización y refrescamos el RecyclerView.

La segunda función hace que nuestra clase (this) sea informada con cada actualización del proveedor de localización. Lo hacemos para el proveedor basado en GPS (cada 10s y si hay un cambio de más de 5m) y con el basado en redes (cada 20s y si hay un cambio de más de 10m).

11. Para recibir los eventos de localización la clase CasosUsoLocalizacion implementa la interfaz LocationListener. Añade las siguientes funciones:

```

@Override public void onLocationChanged(Location location) {
    Log.d(TAG, "Nueva localización: "+location);
    actualizaMejorLocaliz(location);
    adaptador.notifyDataSetChanged();
}
@Override public void onProviderDisabled(String proveedor) {
    Log.d(TAG, "Se deshabilita: "+proveedor);
    activarProveedores();
}

```

```

    }
@Override public void onProviderEnabled(String proveedor) {
    Log.d(TAG, "Se habilita: "+proveedor);
    activarProveedores();
}
@Override
public void onStatusChanged(String proveedor, int estado, Bundle extras) {
    Log.d(TAG, "Cambia estado: "+proveedor);
    activarProveedores();
}

```

Las acciones a realizar resultan evidentes: cuando la actualizamos cambie la posición y cuando cambie el estado tratamos de activar nuevos proveedores.

12. Ahora añade la siguiente función:

```

private static final long DOS_MINUTOS = 2 * 60 * 1000;

private void actualizaMejorLocaliz(Location localiz) {
    if (localiz != null && (mejorLoc == null
        || localiz.getAccuracy() < 2*mejorLoc.getAccuracy()
        || localiz.getTime() - mejorLoc.getTime() > DOS_MINUTOS)) {
        Log.d(TAG, "Nueva mejor localización");
        mejorLoc = localiz;
        ((Aplicacion) getApplication()).posicionActual.setLatitud(
            localiz.getLatitude());
        ((Aplicacion) getApplication()).posicionActual.setLongitud(
            localiz.getLongitude());
    }
}

```

En la variable mejorLoc almacenamos la mejor localización. Esta solo será actualizada con la nueva propuesta si: todavía no ha sido inicializada; o la nueva localización tiene una precisión aceptable (al menos la mitad que la actual); o la diferencia de tiempo es superior a dos minutos. Una vez comprobado si se cumple alguna de las tres condiciones, actualizamos mejorLocaliz y copiamos la posición en posicionActual.

13. Si dejáramos activos los escuchadores de eventos mientras la aplicación está en segundo plano, podríamos quedarnos sin batería. Para evitar esta situación añade en MainActivity:

```

@Override protected void onResume() {
    super.onResume();
    usoLocalizacion.activar();
}

@Override protected void onPause() {
    super.onPause();
    usoLocalizacion.desactivar();
}

```

14. Añade los dos nuevos casos de uso:

```
public void activar() {
    if (hayPermisoLocalizacion()) activarProveedores();
}

public void desactivar() {
    if (hayPermisoLocalizacion()) manejadorLoc.removeUpdates(this);
}
```

- 15.** Una vez que ya disponemos de la posición actual, vamos a tratar de mostrar la distancia a cada lugar en el RecyclerView de la actividad principal. Abre el *layout elemento_lista.xml* y añade al final del ConstraintLayout la nueva vista que se indica:

```
...
<TextView android:id="@+id/distancia"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:layout_toRightOf="@+id/valoracion"
    android:gravity="right"
    android:text="... Km" />
</...ConstraintLayout>
```

- 16.** AdaptadorLugares dentro de la clase ViewHolder la siguiente variable:

```
public TextView distancia;
En el constructor de ViewHolder añade al final
```

```
distancia = itemView.findViewById(R.id.distancia);
```

- 17.** Dentro del método personalizaVista() añade el siguiente código al final:

```
GeoPunto pos=((Aplicacion) itemView.getContext().getApplicationContext())
            .posicionActual;
if (pos.equals(GeoPunto.SIN_POSICION) ||
        lugar.getPosicion().equals(GeoPunto.SIN_POSICION)) {
    distancia.setText("... Km");
} else {
    int d=(int) pos.distancia(lugar.getPosicion());
    if (d < 2000) distancia.setText(d + " m");
    else           distancia.setText(d / 1000 + " Km");
}
```

Nos aseguramos de que la posición actual y la del lugar existen. Luego calculamos la distancia en la variable d. Si la distancia es inferior a 2000, se muestra en metros; en caso contrario se muestra en Km.

- 18.** Ejecuta la aplicación y verifica el resultado obtenido:



Nota: En este ejercicio se ha decidido extraer todo el código que nos permite mantener la localización del dispositivo a una nueva clase. Se podría haber integrado dentro de MainActivity, como se hizo en el ejercicio “La API de localización de Android”. Hacerlo de esta forma divide las responsabilidades entre las dos clase, lo que las hace más fáciles de entender y de mantener. Además el código es más reutilizable, si en un futuro queremos que otra actividad acceda a la localización, podremos usar la clase CasosUsoLocalizacion sin tener que cambiarla.

Google Maps API v2.

Objetivos:

Se describe como utilizar las librerías de Google Maps V2.

Google Maps nos proporciona un servicio de cartografía *online* que podremos utilizar en nuestras aplicaciones Android. Veamos las claves necesarias para utilizarlo. Estudiaremos la versión 2 de la API, que incorpora interesantes ventajas respecto a la versión anterior. Entre estas ventajas destaca el menor tráfico intercambiado con el servidor, la utilización de *fragments* y los gráficos en 3D. Como inconveniente cabe resaltar que la nueva versión solo funciona en el dispositivo con Google Play instalado.

Conviene destacar que, a diferencia de Android, Google Maps no es un *software libre*, por lo que está limitado a una serie de condiciones de servicio. Desde Julio de 2018 se ha introducido una serie de restricciones de uso[1] que debe:

- -Mientras la política de utilización anterior nos permitía centralizar 18 APIs de localización distintas, ahora se han reducido a 3: mapas, rutas y lugares.
- -Google “regalará” 200 dólares mensuales de uso a cada desarrollador que utilice las nuevas APIs de Google Maps.
- -El acceso a Google Maps se integra dentro de la plataforma Cloud de Google. Esto obliga a los desarrolladores a indicar un medio de pago para utilizar las APIs, aunque no vaya a exceder de los 200 dólares mensuales de crédito.
- -Las llamadas gratuitas a las APIs se han limitado de 25.000 peticiones diarias a 28.000 por mes.
- -Para poder utilizar el API de Google Maps el desarrollador deberá tener una llave válida, actualizada y su perfil de Google Cloud deberá incluir, como indicamos previamente, sus datos bancarios.

A cambio de lo anterior, podemos incluir propaganda en los mapas o incluso podemos usarlo en aplicaciones móviles de pago.

Obtención de una clave Google Maps

Para poder utilizar este servicio de Google, igual que como ocurre cuando se utiliza desde una página web, será necesario registrar la aplicación que lo utilizará. Tras registrar la aplicación se nos entregará una clave que tendremos que indicar en la aplicación.

Ejercicio: Obtención de una clave Google Maps

1. Para obtener la clave Google Maps entra en la siguiente página web <https://code.google.com/apis/console/>
2. Tendrás que introducir un usuario de Google que realiza la solicitud.

3. Crea un nuevo proyecto en esta consola. Para ello abre el desplegable de la parte superior y en la ventana emergente selecciona NUEVO PROYECTO. Introduce como nombre Ejemplo Google Maps y pulsa Crear. (el proceso tardará algunos segundos y debes tener en cuenta que no podrás modificar el nombre asignado al proyecto).
4. Una vez generado, selecciónalo en el desplegable superior y accede a la opción Ir a la visión general de las APIs, que encontrarás dentro de la tarjeta APIs.
5. En la nueva pantalla podrás ver algunos accesos directos a las APIs más comunes, entre las que se suele encontrar la opción de Maps SDK for Android. Si no es así, selecciona la opción HABILITAR APIs Y SERVICIOS en la parte superior, donde podrás acceder a Maps SDK for Android.
6. Una vez dentro de la opción de mapas podrás encontrar información relacionada con la documentación o el listado de precios[2]. Te recomendamos que leas detenidamente ambas informaciones. Una vez hecho, pulsa en Habilitar.
7. En la nueva ventana podremos consultar información relevante sobre las cuotas de uso y métricas de nuestra aplicación, una vez esté operativa. De momento, selecciona la pestaña de Credenciales. En la ventana emergente selecciona Crear Credenciales. Selecciona Clave de API.
8. En la ventana siguiente, copia al portapapeles la clave creada:

Clave de API creada

Para usar esta clave en tu aplicación, transfírela como un parámetro
key=API_KEY

Tu clave de API
AIzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8



 Restringe la clave para impedir el uso no autorizado en producción.

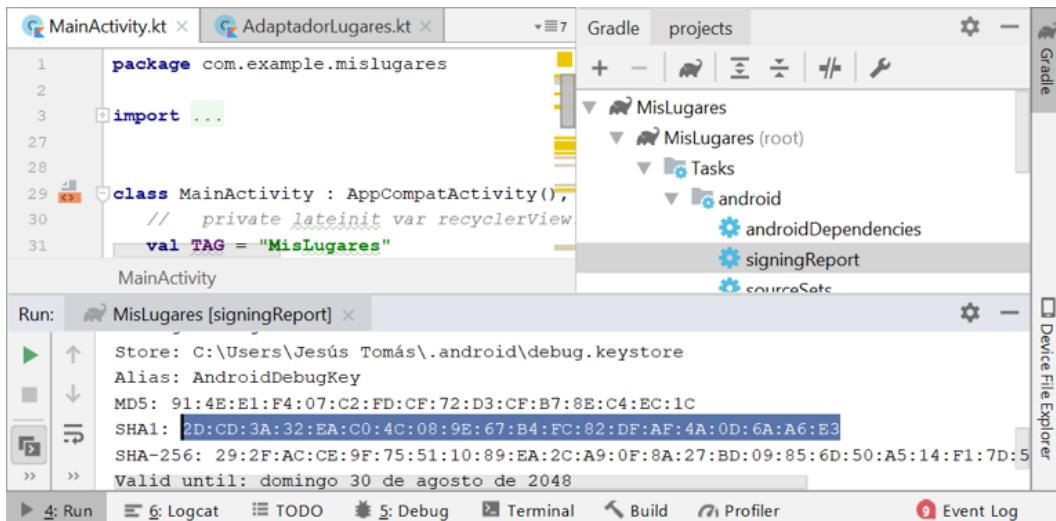
[CERRAR](#) [RESTRINGIR CLAVE](#)

Ejercicio: Restringir uso de la clave Google Maps

La clave que acabamos de crear podría caer en malas manos y ser usada desde otra aplicación Android, iOS o Web. Esto podría ser perjudicial para nosotros, al verse reducida la cuenta asignada a esta clave. Una forma de evitar usos no autorizados de esta clave consiste en indicar a Google que solo permita el uso de esta clave desde nuestra aplicación. Para conseguirlo, lo primero que necesitamos es la huella digital SHA1 del certificado digital con el que se ha firmado nuestra aplicación. En esta fase de desarrollo estamos usando el certificado digital de depuración. En la fase de publicación, el certificado será diferente y tendremos que volver a obtener la huella SHA1. Los pasos 1 al 4 permiten obtener la huella digital SHA1 desde Android Studio. Los pasos 5 al 9 realizan lo mismo, pero desde la línea de comando. Puedes utilizar la alternativa que prefieras.

1. Desde Android Studio, pulsa en el botón Gradle del panel de la derecha.
2. En el desplegable abre la ruta <Nombre del proyecto>/Task/android.
3. Haz doble clic en signingReport.

4. En la ventana Run, aparecerá la firma digital SHA1. Cópiala al portapapeles y pasa al punto 10.



5. El primer paso va a consistir en descubrir dónde está almacenado el certificado digital de depuración. Accede a la carpeta .android que encontrarás en la carpeta de tu usuario. Dentro se almacena el fichero debug.keystore con el certificado digital de depuración. En Windows, la ruta de este fichero podría ser C:\Users\<Usuario>\.android\debug.keystore. En Linux y Mac, la ruta es /.android/debug.keystore.

6. Copia esta ruta en el portapapeles.

7 Ahora necesitamos extraer la huella digital SHA1 de este fichero. Para extraer la huella digital puedes utilizar el programa keytool. En Windows, este programa se encuentra en la carpeta C:\Program Files\Java\jre7\bin\ o en una similar. Abre un intérprete de comandos (símbolo del sistema) y sitúate en la carpeta anterior (o similar).

```
cd C:\Archivos de programa\Java\jre7\bin
```

8. Ejecuta el siguiente comando reemplazando el nombre del fichero por el que acabas de copiar en el portapapeles.

```
keytool -v -list -keystore [ruta a debug.keystore]
```

En nuestro ejemplo:

```
keytool -v -list -keystore C:\android-sdk\.android\debug.keystore
```

```
C:\Archivos de programa\Java\jre7\bin>keytool -v -list -keystore c:\android-sdk\platforms\android-23\debug.keystore
Introduzca la contraseña del almacén de claves:
*****
WARNING WARNING WARNING
* La integridad de la información almacenada en el almacén de claves *
* No se ha comprobado. Para comprobar dicha integridad, *
* debe proporcionar la contraseña del almacén de claves. *
*****
Tipo de Almacén de Claves: JKS
Proveedor de Almacén de Claves: SUN
Su almacén de claves contiene 1 entrada

Nombre de Alias: androiddebugkey
Fecha de Creación: 20-jul-2012
Tipo de Entrada: PrivateKeyEntry
Longitud de la Cadena de Certificado: 1
Certificado[1]:
Propietario: CN=Android Debug, O=Android, C=US
Emisor: CN=Android Debug, O=Android, C=US
Número de serie: 7cd560d6
Válido desde: Fri Jul 20 21:32:46 CEST 2012 hasta: Sun Jul 13 21:32:46 CEST 2042
Huellas digitales del certificado:
MD5: AF:7C:FC:0F:6F:68:C8:F6:40:7E:74:E3:6C:0E:69:FD
SHA1: 9E:80:89:80:E0:54:45:AA:61:FD:38:75:E3:F5:64:08:DB:9F:83:B9
SHA256: 0A:DC:F3:67:D9:91:93:BB:1A:8A:5E:96:12:D0:15:22:5E:72:76:57:B2:
CD:74:BC:7C:97:D4:DE:F3:7E:74:54
Nombre del Algoritmo de Firma: SHA256withRSA
Versión: 3
```

NOTA: Si la ruta del fichero tiene espacios, introduce la ruta entre comillas.

El programa te solicitará una contraseña para proteger el almacén de claves. Deja la contraseña en blanco. De toda la información mostrada, nos interesa la huella digital del certificado en codificación SHA1. Como puedes ver en la captura anterior, para nuestro ejemplo está formada por los siguientes bytes:

9E:80:89:80:E0:54:45:AA:61:FD:38:75:E3:F5:64:08:DB:9F:83:B9

9. Copia en el portapapeles esta secuencia de dígitos. En Windows pulsa con el botón derecho sobre la barra superior de la venta y selecciona Marcar. Selecciona el área a copiar. Luego, en este mismo menú, selecciona Editar/Copiar.

10. Accede a la consola de administración de APIs de Google (<https://code.google.com/apis/console/>) y selecciona el proyecto creado.

11. En el menú de la izquierda selecciona Credenciales. En la lista de Claves de API, pulsa en el botón de editar (con forma de lápiz):

Claves de API				
<input type="checkbox"/>	Nombre	Fecha de creación	Restricción	Clave
<input checked="" type="checkbox"/>	Clave de API 1	17 abr. 2017	Ninguna	AlzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8



12. Selecciona Aplicación para Android y pulsa en Añadir nombre de paquete y huella digital. Aparecerán dos cuadros de entrada donde has de añadir el nombre de paquete de la aplicación y la huella digital obtenida al principio del ejercicio.

Restricción de clave

Si restringes una clave, puedes especificar qué sitios web, direcciones IP o aplicaciones pueden usarla. [Más información](#)

- Ninguna
- URLs de referencia HTTP (sitios web)
- Direcciones IP (servidores web, tareas cron, etc.)
- Aplicaciones para Android
- Aplicaciones para iOS

Restringir el uso a tus aplicaciones Android (Opcional)

Añade el nombre del paquete y la huella digital del certificado de firma SHA-1 para restringir el uso de tus aplicaciones de Android

Puedes encontrar el nombre del paquete en el archivo AndroidManifest.xml. A continuación, usa el comando siguiente para obtener la huella digital:

```
$ keytool -list -v -keystore mystore.keystore
```

Nombre de paquete	Huella digital de certificado SHA-1
org.example.ejemplogooglemaps	9E:80:89:80:E0:54:45:AA:61:FD:38:75:E3:F5:64:08:DB:9F:83:B9

+ Añadir nombre de paquete y huella digital

13. Finalmente, pulsa en Guardar.

Ejercicio: Un ejemplo simple con Google Maps

Veamos un sencillo ejemplo que nos permite visualizar un mapa centrado en las coordenadas geográficas detectadas por el sistema de posicionamiento.

1. Crea un nuevo proyecto con los siguientes datos:

Phone and Tablet / Empty Activity

Name: Ejemplo Google Maps

Package name: org.example.ejemplogooglemaps

Language: Java

Minimum API level: API 19 Android 4.4 (KitKat)

2. Abre *Android SDK Manager* y asegúrate de que los paquetes *Google Play Services* y *Google Repository* están instalados. Si hay una versión más reciente actualízala:

	Name	Version	Status
<input checked="" type="checkbox"/>	Google Play services	33	Installed
<input checked="" type="checkbox"/>	Google USB Driver, rev 11	11.0.0	Installed
<input type="checkbox"/>	Google Web Driver	2	Not installed
<input checked="" type="checkbox"/>	Intel x86 Emulator Accelerator (HAXM installer)	6.0.4	Installed
<input type="checkbox"/>	NDK	13.0.3315539	Not installed
<input type="checkbox"/>	Support Repository		
<input type="checkbox"/>	ConstraintLayout for Android		Update Available: 1
<input type="checkbox"/>	Solver for ConstraintLayout		Update Available: 1
<input checked="" type="checkbox"/>	Android Support Repository	38.0.0	Installed
<input checked="" type="checkbox"/>	Google Repository	36	Installed

3. Vamos a importar a nuestro proyecto el paquete de la librería de *Google Play Services*. Para ello añade en el graddle la siguiente línea.

```
implementation 'com.google.android.gms:play-services-maps:16.0.0'
```

4. *AndroidManifest.xml*, añade en siguiente permiso:

NOTA: Los permisos de localización no son necesarios para trabajar con Google Maps, pero sí los debemos especificar para trabajar con la funcionalidad MyLocation, la cual vamos a utilizar en nuestro ejemplo. **NOTA:** Este permiso incluye de forma implícita los permisos ACCESS_COARSE_LOCATION y NETWORK_PROVIDER.

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

5. Añade también las siguientes líneas dentro de la sección <application>:

```
<meta-data android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key" />
```

6. Crea el siguiente fichero de recurso *res/values/google_maps_api.xml*:

```
<resources>
    <string name="google_maps_key" templateMergeStrategy="preserve"
        translatable="false">
        AIzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8
    </string>
</resources>
```

Reemplaza los caracteres marcados ("Alza...") por la API Key obtenida en el ejercicio anterior.

7. Reemplaza el contenido del layout *activity_main.xml* por:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <fragment
        android:id="@+id/mapa"
        class="com.google.android.gms.maps.SupportMapFragment"/>
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

NOTA: La etiqueta <fragment> ha de escribirse en minúscula.

8. Abre MainActivity.class y haz que esta clase herede de FragmentActivity y que implemente la interfaz OnMapReadyCallback:

```
class MapaActivity: FragmentActivity(), OnMapReadyCallback {
9. Reemplaza el método onCreate por el siguiente:
JavaKotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    // Obtenemos el mapa de forma asíncrona (notificará cuando esté listo)
    val mapFragment = supportFragmentManager.findFragmentById(R.id.mapa) as
        SupportMapFragment
    mapFragment.getMapAsync(this)
}
```

10. Debemos implementar el método onMapReady que será llamado en el momento en que el mapa está disponible. Es en esta función donde podremos manipular el mapa. Una implementación mínima sería la siguiente:

```
override fun onMapReady(googleMap: GoogleMap) {
    val UPV = LatLng(39.481106, -0.340987) //Nos ubicamos en la UPV
    googleMap.addMarker(MarkerOptions().position(UPV).title("Marker UPV"))
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(UPV))
}
```

11. Ejecuta la aplicación. A continuación, se muestra el resultado:



NOTA: Si utilizas un emulador, asegúrate que disponga de servicios de Google Play.

Ejercicio: Introduciendo código en Google Maps

En el ejercicio anterior hemos visto un ejemplo muy básico, donde solo se mostraba un mapa con las opciones predeterminadas. En este ejercicio aprenderemos a configurarlo y añadir marcadores desde el código.

1. Abre el layout `activity_main.xml` y añade los siguientes tres botones dentro del `<ConstraintLayout>` (tras el `<fragment ...>`):

```

<Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="moveCamera"
        android:text="ir a UPV"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />
<Button android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="animateCamera"
        android:text="animar a UPV"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button3"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button1" />
<Button android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="addMarker"
        android:text="marcador"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button2" />

```

2. Sustituye el contenido de `MainActivity.java` por:

```

JavaKotlin
class MainActivity : FragmentActivity(), OnMapReadyCallback,
                    GoogleMap.OnMapClickListener {
    lateinit var mapa: GoogleMap
    val UPV = LatLng(39.481106, -0.340987)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val mapFragment = supportFragmentManager.findFragmentById(R.id.mapa)
                    as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        mapa = googleMap.apply {
            mapType = GoogleMap.MAP_TYPE_SATELLITE
            uiSettings.isZoomControlsEnabled = false
            moveCamera(CameraUpdateFactory.newLatLngZoom(UPV, 15f))
            addMarker(MarkerOptions().position(UPV).title("UPV")
                .snippet("Universidad Politécnica de Valencia")
                .icon(BitmapDescriptorFactory.fromResource(
                    android.R.drawable.ic_menu_compass)))
                .anchor(0.5f, 0.5f))
        }
        if (ActivityCompat.checkSelfPermission(this,
                android.Manifest.permission.ACCESS_FINE_LOCATION)
                == PackageManager.PERMISSION_GRANTED) {
            mapa.isMyLocationEnabled = true
            mapa.uiSettings.isCompassEnabled = true
        }
    }
}

```

```

        }
        mapa.setOnMapClickListener(this)
    }

    fun moveCamera(view: View) {
        mapa.moveCamera(CameraUpdateFactory.newLatLng(UPV))
    }

    fun animateCamera(view: View) {
        mapa.animateCamera(CameraUpdateFactory.newLatLng(UPV))
    }

    fun addMarker(view: View) {
        mapa.addMarker(MarkerOptions().position(mapa.cameraPosition.target))
    }

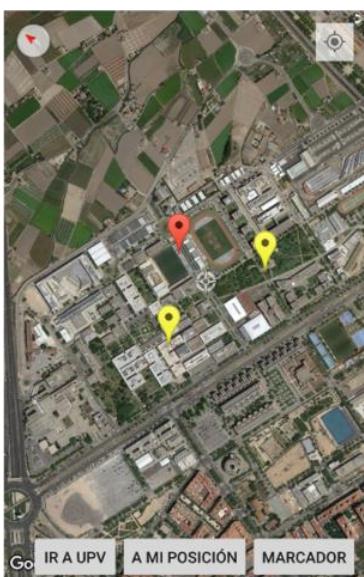
    override fun onMapClick(puntoPulsado: LatLng) {
        mapa.addMarker(MarkerOptions().position(puntoPulsado)
            .icon(BitmapDescriptorFactory.defaultMarker(
                BitmapDescriptorFactory.HUE_YELLOW)))
    }
}

```

Comenzamos declarando dos objetos: UPV, que hace referencia a la posición geográfica de la Universidad Politécnica de Valencia, y mapa, que nos permitirá acceder al objeto GoogleMap que hemos insertado en un fragment de nuestro layout. El mapa es cargado asíncronamente, por lo que es necesario implementar el método onMapReady de la interfaz OnMapReadyCallback, que sera llamado en el momento en que mapa esté listo. Será en este método cuándo podremos configurar el objeto GoogleMap que nos pasan como parámetro, para adaptarlo a nuestras necesidades. setMapType() permite seleccionar el tipo de mapa (normal, satélite, híbrido o relieve). Para averiguar las constantes correspondientes, te recomendamos que utilices la opción de autocompletar (escribe GoogleMap.y podrás seleccionar las constantes de esta clase). El método moveCamera() desplaza el área de visualización a una determinada posición (UPV), a la vez que define el nivel de zoom (15). El nivel de zoom ha de estar en un rango de 2 (continente) hasta 21 (calle). El método addMarker() permite añadir los típicos marcadores que habrás visto en muchos mapas. En este ejemplo se indica la posición (UPV), un título, una descripción, un ícono y el punto del ícono, que haremos coincidir con el punto exacto que queremos indicar en el mapa. Un valor de (0, 0) corresponde a la esquina superior izquierda del ícono y (1, 1), a la esquina inferior derecha. Como nuestro ícono tiene forma de círculo, hemos indicado el valor (0.5, 0.5) para que coincida con su centro. Finalmente, hemos registrado un escuchador de evento para detectar pulsaciones sobre la pantalla. El escuchador vamos a ser nosotros mismos (this), por lo que hemos implementado la interfaz OnMapClickListener y añadido el método onMapClick(). El método setMyLocationEnabled(true) activa la visualización de la posición del dispositivo por medio del típico círculo. Para dispositivos con versión 6 o superior hay que tener la precaución de verificar si tenemos permiso de localización antes de activar esta opción. Por defecto, al tratarse de un permiso catalogado como peligroso se encontrará desactivado. En este código no se solicita este permiso. Para activarlo manualmente debes usar en el dispositivo la opción Ajustes / Aplicaciones / Ejemplo Google Maps / Permisos. El método getUiSettings() permite configurar las acciones de la interfaz de usuario. En este ejemplo se han utilizado dos: desactivar los botones de zoom y visualizar una brújula. Estos métodos solo están disponibles si la capa LocationLayer está activa, por lo que es recomendable iniciarlos posteriormente al método setMyLocationEnabled(true). Puedes usar autocompletar para descubrir otras posibles

configuraciones. En caso de no tener permiso de localización, debemos deshabilitar el botón que nos lleva a nuestra posición.

A continuación, se incluyen los tres métodos que se ejecutarán al pulsar sobre los botones añadidos al layout. El primero, moveCamera(), desplaza el punto de visualización a la UPV. A diferencia del uso anterior, sin cambiar el nivel de zum que el usuario tenga seleccionado. El segundo, animateCamera(), nos desplaza también a la UPV por medio de una animación (similar a la que a veces utilizan en el Telediario para mostrar un punto en conflicto). El tercero, addMarker(), añade un nuevo marcador en el centro del mapa que estamos observando (getCameraPosition()). En este caso usaremos el marcador por defecto, sin información adicional. Como hemos indicado, se llamará a onMapClick() cuando se pulse sobre el mapa. Se pasa como parámetro las coordenadas del punto donde se ha pulsado, que utilizaremos para añadir un marcador. Esta vez el marcador será de color amarillo.



3. Ejecuta la aplicación. Se muestra el resultado.

[1] <https://cloud.google.com/maps-platform/pricing/sheet/?hl=es>

[2] <https://developers.google.com/maps/billing/understanding-cost-of-use?hl=es#maps-product>

Añadiendo mapas en Mis Lugares

Objetivos:

Describir como añadir mapas a la aplicación Mis Lugares.

Ejercicio: Añadiendo Google Maps en Mis Lugares

1. Realiza el ejercicio “*Obtención de una clave Google Maps*”, pero esta vez indica como nombre del proyecto Mis Lugares.
2. Añade la librería Google Maps y configura AndroidManifest.xml. Para ello sigue los puntos 2 al 6 del ejercicio “*Un ejemplo simple con Google Maps*”.
3. Crea un nuevo layout que se llame mapa.xml con el siguiente código:

```
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
    <fragment android:id="@+id/mapa"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        class="com.google.android.gms.maps.SupportMapFragment"/>  
</LinearLayout>
```

4. Crea una nueva clase para la actividad que mostrará el mapa:

```
public class MapaActivity extends FragmentActivity  
    implements OnMapReadyCallback {  
    private GoogleMap mapa;  
    private RepositorioLugares lugares;  
  
    @Override public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.mapa);  
        lugares = ((Aplicacion) getApplication()).lugares;  
        SupportMapFragment mapFragment = (SupportMapFragment)  
            getSupportFragmentManager().findFragmentById(R.id.mapa);  
        mapFragment.getMapAsync(this);  
    }  
  
    @Override public void onMapReady(GoogleMap googleMap) {  
        mapa = googleMap;  
        mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);  
        if (ActivityCompat.checkSelfPermission(this,  
            android.Manifest.permission.ACCESS_FINE_LOCATION) ==  
            PackageManager.PERMISSION_GRANTED) {  
            mapa.setMyLocationEnabled(true);  
            mapa.getUiSettings().setZoomControlsEnabled(true);  
            mapa.getUiSettings().setCompassEnabled(true);  
        }  
        if (lugares.tamaño() > 0) {  
            GeoPunto p = lugares.elemento(0).getPosicion();  
            mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(
```

```
        new LatLng(p.getLatitud(), p.getLongitud()), 12));
    }
    for (int n=0; n<lugares.tamaño(); n++) {
        Lugar lugar = lugares.elemento(n);
        GeoPunto p = lugar.getPosicion();
        if (p != null && p.getLatitud() != 0) {

            Bitmap iGrande = BitmapFactory.decodeResource(
                getResources(), lugar.getTipo().getRecurso());
            Bitmap icono = Bitmap.createScaledBitmap(iGrande,
                iGrande.getWidth() / 7, iGrande.getHeight() / 7, false);
            mapa.addMarker(new MarkerOptions()
                .position(new LatLng(p.getLatitud(), p.getLongitud()))
                .title(lugar.getNombre()).snippet(lugar.getDireccion())
                .icon(BitmapDescriptorFactory.fromBitmap(icono)));
        }
    }
}
```

El código utilizado es similar al utilizado en el ejercicio anterior. Una diferencia está en que el centro del mapa se sitúa (`moveCamera`) en el primer lugar de `listaLugares` siempre que tenga algún elemento. Luego se introduce un bucle donde añadiremos un marcador para cada lugar. Queremos utilizar los iconos utilizados en la aplicación. El problema es que su tamaño es excesivo. Para resolverlo los leemos como `Drawables` de los recursos, los convertimos en `Bitmap` y los escalamos dividiendo su anchura y altura entre siete.

NOTA: Desde un punto de vista de eficiencia, lo ideal sería añadir los nuevos recursos reescalados.

5. Registra esta actividad añadiendo la siguiente línea en AndroidManifest.xml dentro de la etiqueta <application>:

```
<activity android:name="presentacion.MapaActivity" />
```

6. Vamos a añadir en la actividad principal una nueva opción en el ActionBar para visualizar el mapa. Para ello edita el fichero res/menu/menu_main.xml y añade el siguiente ítem de menú:

```
<item android:title="Mapa"  
      android:id="@+id/menu_mapa"  
      android:icon="@android:drawable/ic_menu_myplaces"  
      android:orderInCategory="100"  
      app:showAsAction="always"/>
```

7. Abre la clase MainActivity y añade dentro del método onOptionsItemSelected() el siguiente código:

```
if (id==R.id.menu_mapa) {  
    Intent intent = new Intent(this, MapaActivity.class);  
    startActivity(intent);  
}
```

8. Ejecuta la aplicación en un dispositivo real y selecciona la opción que acabas de introducir. El resultado ha de ser similar al siguiente:



NOTA: Si aparece el error: `NoClassDefFoundError: Lorg/apache/http/ProtocolVersion consulta.`

- Si cambias de orientación el terminal, la actividad Mapa se reinicializará y el mapa volverá a la posición inicial. Si quieres evitarlo, bloquea la orientación de esta actividad. Para ello añade el siguiente atributo en la definición de la actividad dentro de `AndroidManifest.xml`.

Ejercicio: Añadiendo un escuchador en Google Maps

Si en el ejercicio anterior pulsas sobre un marcador, verás como se abre una ventana de información (`InfoWindow`). Queremos conseguir que cuando se pulse sobre esta ventana se abra la actividad que nos muestra información detallada.



- Vamos a introducir un escuchador que recoja el evento correspondiente cuando se pulse sobre esta ventana. Para ello, añade al final del método `onMapReady()` de la actividad `MapaActivity` la siguiente línea:

```
mapa.setOnInfoWindowClickListener(this);
```

- Aparecerá un error justo en la línea que acabas de introducir. Si sitúas el cursor de texto sobre el error, aparecerá una bombilla roja con opciones para resolver el error. Selecciona “*Make ‘MapaActivity’ implement ‘OnInfoWindowClickListener’*”. Observa como en la definición de la clase se añade esta interfaz:

```
public class MapaActivity extends FragmentActivity implements
    OnMapReadyCallback, GoogleMap.OnInfoWindowClickListener {
```

- Ahora aparecerá un nuevo error sobre `MapaActivity` dado que no implementa esta interface. Aparecerá la bombilla roja con opciones para resolver el error. Selecciona “*Implemented methods*” y luego el único método de la interfaz. Completa este código, tal y como se muestra a continuación:

JavaKotlin

```
@Override public void onInfoWindowClick(Marker marker) {
    for (int id=0; id<lugares.tamanyo(); id++){
        if (lugares.elemento(id).getNombre()
            .equals(marker.getTitle())){
            Intent intent = new Intent(this, VistaLugarActivity.class);
            intent.putExtra("pos", id);
```

```
        startActivity(intent);
        break;
    }
}
```

Se llamará a este método cuando se pulse sobre cualquier ventana de información. Para averiguar el marcador al que corresponde, se pasa el objeto Marker que se ha pulsado. En el marcador hemos introducido alguna información sobre el lugar (como el nombre); sin embargo, lo que necesitamos es el id del lugar. No resulta sencillo introducir este id en un objeto Marker. Para resolverlo hemos introducido un bucle donde se busca un lugar cuyo nombre coincide con el título de marcador. Cuando se encuentre una coincidencia, se creará una intención para lanzar la actividad correspondiente.

NOTA: Lo más correcto para resolverlo sería crear un descendiente de Marker que añada este id. Sin embargo, la clase Marker se ha marcado como final, por lo que no es posible crear descendientes.

Unidad 8:

Bases de datos y Fragments en Android

Bases de datos y Fragments en Android Introducción a la unidad

La aplicación desarrollada hasta este capítulo guarda la información en forma de variables. El problema de estas variables es que dejan de existir en el momento en que la aplicación es destruida. Muy frecuentemente vamos a necesitar almacenar información de manera permanente. Las alternativas más habituales para conservar esta información son los ficheros, las bases de datos o servicios a través de la red. Estas técnicas nos permiten mantener a buen recaudo los datos de la aplicación. De forma adicional, el sistema Android pone a nuestra disposición dos nuevos mecanismos para almacenar datos, las preferencias y *ContentProvider*.

Comenzaremos la unidad enumerando las alternativas para guardar información en Android. Luego se describirá el almacenamiento de datos usando bases de datos. Android incorpora la librería SQLite, que nos permitirá crear y manipular nuestras propias bases de datos de forma muy sencilla.

Al final de la unidad aprenderemos a usar *fragments*. Su utilización es fundamental, dado que el nuevo planteamiento de diseño de la interfaz de usuario en Android se basa en *fragments*. Se trata de elementos constructivos básicos que podremos combinar dentro del *layout* de una actividad.

Para poder seleccionar fechas y horas de forma cómoda en la aplicación Mis Lugares, aprenderemos a utilizar cuadros de diálogo con este fin.

Objetivos:

- -Repasar las alternativas para el almacenamiento de datos en Android.
- -Mostrar como desde Android podemos utilizar SQLite para trabajar con bases de datos.
- -Mostrar como usando *fragments* podemos diseñar elementos reutilizables de la IU.
- -Mostrar el uso de cuadros de diálogo para seleccionar fechas y horas.

Almacenamiento Datos en Android

video [Almacenamiento de datos en Android](#)

Objetivos:

Repasar las alternativas para el almacenamiento de datos permanentemente en Android.

Existen muchas alternativas para almacenar información de forma permanente en un sistema informático. A continuación mostramos una lista de las más habituales utilizadas en Android:

- **Preferencias:** Es un mecanismo liviano que permite almacenar y recuperar datos primitivos en la forma de pares clave/valor. Este mecanismo se suele utilizar para almacenar los parámetros de configuración de una aplicación.
- **Ficheros:** Puedes almacenar los ficheros en la memoria interna del dispositivo o en un medio de almacenamiento removible como una tarjeta SD. También puedes utilizar fichero añadidos a tu aplicación como recursos.
- **XML:** Se trata de un estándar fundamental para la representación de datos, en Internet y en muchos otros entornos (como en el Android SDK). En Android disponemos de las librerías SAX y DOM para manipular datos en XML.
- **JSON:** Es una alternativa a XML para almacenar información estructurada. Usa una representación simple y compacta, lo que la hace especialmente interesante para transacciones por Internet. En este capítulo se describen dos herramientas: GSON y org.json.
- **Base de datos:** Las APIs de Android contienen soporte para SQLite. Tu aplicación puede crear y usar base de datos SQLite de forma muy sencilla y con toda la potencia que nos da el lenguaje SQL.
- **Proveedores de contenido:** Un proveedor de contenido es un componente de una aplicación que expone el acceso de lectura / escritura de sus datos a otras aplicaciones.. Está sujeto a las restricciones de seguridad que quieras imponer. Los proveedores de contenido implementan una sintaxis estándar para acceder a sus datos mediante URI (Uniform Resource Identifiers) y un mecanismo de acceso para devolver los datos similar a SQL. Android provee algunos proveedores de contenido para tipos de datos estándar, tales como contactos personales, ficheros multimedia, etc.
- **Internet:** No te olvides que también puedes usar la nube para almacenar y recuperar datos.

Uso de base de datos en Android

video [Uso de bases de datos en Android](#)

Objetivos:

Mostrar como desde Android podemos utilizar SQLite para trabajar con bases de datos.

En los próximos ejercicios pasamos a demostrar cómo guardar los datos de la aplicación Mis Lugares en una base de datos. Esta estará formada por una única tabla (lugares). A continuación, se muestran las columnas que contendrán y las filas que se introducirán como ejemplo. Los valores que aparecen en las columnas `_id` y `fecha` no coincidirán con los valores reales:

<code>_id</code>	nombre	direccion	longitud	latitud	tipo	foto	telefono	url	Comentario	fecha	valoracion
1	Escuela	C/ Paran	-0.166	38.99	7		962849	ht	Uno de lo	2345	3.0
2	Al de	P. Industr	-0.190	38.92	2		636472	ht	No te pier	2345	3.0
4	android	ciberesp	0.0	0.0	7			ht	Amplia tu	2345	5.0
7	Barranc	Vía Verd	-0.295	38.86	9			ht	Espectacu	2345	4.0
5	La Vital	Avda. de	-0.172	38.97	6		962881	ht	El típico c	2345	2.0

Estructura de la tabla lugares de la base de datos lugares.

Creación de una base de datos

Ejercicio: Utilizando una base de datos en Mis Lugares.

1. Comenzamos haciendo una copia del proyecto, dado que en la nueva versión se eliminará parte del código desarrollado y es posible que queramos consultarla en un futuro. Abre en el explorador de ficheros la carpeta que contiene el proyecto. Para hacer esto puedes pulsar con el botón derecho sobre app en el explorador del proyecto y seleccionar *Show in Explorer*. Haz una copia de esta carpeta un nuevo nombre del proyecto.

2. Crea la clase `LugaresBD` en el proyecto y escribe el siguiente código:

```
public class LugaresBD extends SQLiteOpenHelper {

    Context contexto;

    public LugaresBD(Context contexto) {
        super(contexto, "lugares", null, 1);
    }
}
```

```

        this.contexto = contexto;
    }

@Override public void onCreate(SQLiteDatabase bd) {
    bd.execSQL("CREATE TABLE lugares (" +
        "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "nombre TEXT, " +
        "direccion TEXT, " +
        "longitud REAL, " +
        "latitud REAL, " +
        "tipo INTEGER, " +
        "foto TEXT, " +
        "telefono INTEGER, " +
        "url TEXT, " +
        "comentario TEXT, " +
        "fecha BIGINT, " +
        "valoracion REAL)");
    bd.execSQL("INSERT INTO lugares VALUES (null, "+
        "'Escuela Politécnica Superior de Gandía', "+
        "'C/ Paranimf, 1 46730 Gandia (SPAIN)', -0.166093, 38.995656, "+
        "TipoLugar.EDUCACION.ordinal() + '', 962849300, "+
        "'http://www.epsg.upv.es', "+
        "'Uno de los mejores lugares para formarse.', "+
        System.currentTimeMillis() +", 3.0)");
    bd.execSQL("INSERT INTO lugares VALUES (null, 'Al de siempre', "+
        "'P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)', "+
        "-0.190642, 38.925857, " + TipoLugar.BAR.ordinal() + '', "+
        "636472405, '', """No te pierdas el arroz en calabaza.', "+
        System.currentTimeMillis() +", 3.0)");
    bd.execSQL("INSERT INTO lugares VALUES (null, 'androidcurso.com', "+
        "'ciberespacio', 0.0, 0.0,"+TipoLugar.EDUCACION.ordinal() + "", "+
        "962849300, 'http://androidcurso.com', "+
        "'Amplia tus conocimientos sobre Android.', "+
        System.currentTimeMillis() +", 5.0)");
    bd.execSQL("INSERT INTO lugares VALUES (null, 'Barranco del Infierno', "+
        "'Vía Verde del río Serpis. Villalonga (Valencia)', -0.295058, "+
        "38.867180, "+TipoLugar.NATURALEZA.ordinal() + "", 0, "+
        "'http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-verde-del-"+
        "rio.html', 'Espectacular ruta para bici o andar', "+
        System.currentTimeMillis() +", 4.0)");
    bd.execSQL("INSERT INTO lugares VALUES (null, 'La Vital', "+
        "'Avda. La Vital, 0 46701 Gandia (Valencia)', -0.1720092, 38.9705949, "+
        "TipoLugar.COMPRAS.ordinal() + '', 962881070, "+
        "'http://www.lavital.es', 'El típico centro comercial', "+
        System.currentTimeMillis() +", 2.0)");
}

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

```

El constructor de la clase se limita a llamar al constructor heredado. Los parámetros se describen a continuación:

contexto: Contexto usado para abrir o crear la base de datos.

nombre: Nombre de la base de datos que se creará. En nuestro caso, “puntuaciones”.

cursor: Se utiliza para crear un objeto de tipo cursor. En nuestro caso no lo necesitamos.

version: Número de versión de la base de datos empezando desde 1. En el caso de que la base de datos actual tenga una versión más antigua se llamará a onUpgrade() para que actualice la base de datos.

El método onCreate() se invocará cuando sea necesario crear la base de datos. Como parámetro se nos pasa una instancia de la base de datos que se acaba de crear. Este es el momento de crear las tablas que contendrán información. El primer campo tiene por nombre _id y será un entero usado como clave principal. Su valor será introducido de forma automática por el sistema, de forma que dos registros no tengan nunca el mismo valor.

En nuestra aplicación necesitamos solo la tabla lugares, que es creada por medio del comando SQL CREATE TABLE lugares... La primera columna tiene por nombre _id y será un entero usado como clave principal. Su valor será introducido automáticamente por el sistema, de forma que dos filas no tengan nunca el mismo valor de _id.

Las siguientes líneas introducen nuevas filas en la tabla utilizando el comando SQL INSERT INTO lugares VALUES (, , ...). Los valores deben introducirse en el mismo orden que las columnas. La primera columna se deja como null dado que corresponde al _id y es el sistema quien ha de averiguar el valor correspondiente. Los valores de tipo TEXT deben introducirse entre comillas, pudiendo utilizar comillas dobles o simples. Como en Java se utilizan comillas dobles, en SQL utilizaremos comillas sencillas. El valor TipoLugar.EDUCACION.ordinal() corresponde a un entero según el orden en la definición de este enumerado y System.currentTimeMillis() corresponde a la fecha actual representada como número de milisegundos transcurridos desde 1970. El resto de los valores son sencillos de interpretar. Ha de quedar claro que este constructor solo creará una base de datos (llamando a onCreate()) siestá todavía no existe. Si ya fue creada en una ejecución anterior, nos devolverá la base de datos existente.

El método onUpgrade() está vacío. Si más adelante, en una segunda versión de Mis Lugares, decidiéramos crear una nueva estructura para la base de datos, tendríamos que indicar un número de versión superior, por ejemplo la 2. Cuando se ejecute el código sobre un sistema que disponga de una base de datos con la versión 1, se invocará el método onUpgrade(). En él tendremos que escribir los comandos necesarios para transformar la antigua base de datos en la nueva, tratando de conservar la información de la versión anterior.

Lectura de datos de una base de datos

3. Para acceder a los datos de la aplicación se definió la interfaz Lugares. Vamos a implementar esta interfaz para que los cambios sean los mínimos posibles. Añade el texto subrayado a la clase:

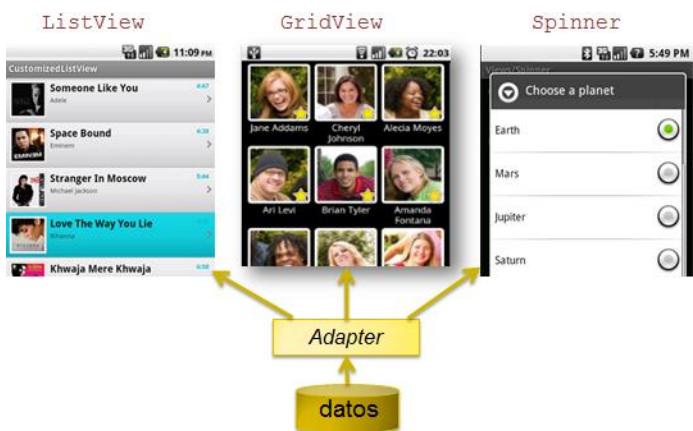
```
public class LugaresBD extends SQLiteOpenHelper  
    implements RepositorioLugares {
```

Aparecerá un error justo en la línea que acabas de introducir. Si sitúas el cursor de texto sobre el error, aparecerá una bombilla roja con opciones para resolver el error. Pulsa en “*Implement methods*”, selecciona todos los métodos y pulsa OK. Observa como en la clase se añaden todos los métodos de esta interfaz. De momento vamos a dejar estos métodos sin implementar. En la sección “Operaciones con bases de datos en Mis Lugares” aprenderemos a realizar las operaciones básicas cuando trabajamos con datos: altas, bajas, modificaciones y consultas.

- No ejecutes todavía la aplicación. Hasta que no hagamos el siguiente ejercicio no funcionará correctamente.

Adaptadores para bases de datos

Un adaptador (Adapter) es un mecanismo de Android que hace de puente entre nuestros datos y las vistas contenidas en un RecyclerView, ListView, GridView o Spinner.



En el siguiente ejercicio vamos a crear un adaptador que toma la información de la base de datos que acabamos de crear y se la muestra a un RecyclerView. Realmente podríamos usar el adaptador AdaptadorLugares que ya tenemos creado. Este adaptador toma la información de un objeto que sigue la interfaz Lugares, restricción que cumple la clase LugaresBD. No obstante, vamos a realizar una implementación alternativa. La razón es que la implementación actual de AdaptadorLugares necesitaría una consultas a la base de datos cada vez que requiera una información de Lugares. (Veremos más adelante que cada llamada a elemento(), añade(), nuevo(), ...) va a suponer una acceso a la base de datos).

El nuevo adaptador, AdaptadorLugaresBD, va a trabajar de una forma más eficiente. Vamos a realizar una consulta de los elementos a listar y los va a guardar en un objeto de la clase Cursor. Mantendrá esta información mientras no cambie la información a listar, por lo que solo va a necesitar una consulta a la base de datos.

Ejercicio: Un Adaptador para base de datos en Mis Lugares

- Crea la clase AdaptadorLugaresBD con el siguiente código:

```
public class AdaptadorLugaresBD extends AdaptadorLugares {

    protected Cursor cursor;

    public AdaptadorLugaresBD(RepositorioLugares
                               lugares, Cursor cursor) {
        super(lugares);
        this.cursor = cursor;
    }

    public Cursor getCursor() {
        return cursor;
    }

    public void setCursor(Cursor cursor) {
        this.cursor = cursor;
    }
}
```

```
public Lugar lugarPosicion(int posicion) {  
    cursor.moveToPosition(posicion);  
    return LugaresBD.extraeLugar(cursor);  
}  
  
public int idPosicion(int posicion) {  
    cursor.moveToPosition(posicion);  
    if (cursor.getCount() > 0) return cursor.getInt(0);  
    else return -1;  
  
    return cursor.getInt(0);  
}  
  
@Override  
public void onBindViewHolder(ViewHolder holder, int posicion) {  
    Lugar lugar = lugarPosicion(posicion);  
    holder.personaliza(lugar);  
    holder.itemView.setTag(new Integer(posicion));  
}  
  
@Override public int getItemCount() {  
    return cursor.getCount();  
}
```

Esta clase extiende AdaptadorLugares; de esta forma aprovechamos la mayor parte del código del adaptador y solo tenemos que indicar las diferencias. La más importante es que ahora el constructor tiene un nuevo parámetro de tipo Cursor, que es el resultado de una consulta en la base de datos. Realmente es aquí donde vamos a buscar los elementos a listar en el RecyclerView. Esta forma de trabajar es mucho más versátil que utilizar un array, podemos listar un tipo de lugar o que cumplan una determinada condición sin más que realizar un pequeño cambio en la consulta SQL. Además, podremos ordenarlos por valoración o cualquier otro criterio, porque se mostrarán en el mismo orden como aparecen en el Cursor. Por otra parte, resulta muy eficiente dado que se realiza solo una consulta a la base de datos, dejando el resultado almacenado en la variable de tipo Cursor.

En Java el constructor de la clase se limita a llamar al super() y a almacenar el nuevo parámetro en una variable global. **En Kotlin** este proceso se indica en la declaración. **En Java** se han añadido los métodos *getter* y *setter* que permiten acceder al Cursor desde fuera de la clase.

Con el método `lugarPosicion()` vamos a poder acceder a un lugar, indicar su posición en Cursor o, lo que es lo mismo, su posición en el listado. Para ello, movemos el cursor a la posición indicada y extraemos el lugar en esta posición, utilizando un método estático de `LugarBD`.

Cuando queramos realizar una operación de borrado o edición de un registro de la base de datos, vamos a identificar el lugar a modificar por medio de la columna `_id`. Recuerda que esta columna ha sido definida en la posición 0. Para obtener el id de un lugar conociendo la posición que ocupa en el listado, se ha definido el método `lugarPosicion()`.

Los dos últimos métodos ya existen en la clase que extendemos, pero los vamos a reemplazar; por esta razón tienen la etiqueta de override. El primero es `onBindViewHolder()` que se utilizaba

para personalizar la vista ViewHolder en una determinada posición. La gran diferencia entre el nuevo método es que ahora el lugar lo obtenemos del cursor, mientras que en el método anterior se obtenía de lugares. Esto supondría una nueva consulta en la base de datos por cada llamada a onBindViewHolder(), lo que sería muy poco eficiente. En el método getItemCount() pasa algo similar, obtener el número de elementos directamente del cursor es más eficiente que hacer una nueva consulta.

Observa la última línea de onBindViewHolder() (holder.view.tag = posicion). El atributo Tag permite asociar a una vista cualquier objeto con información extra. La idea es asociar a cada vista del RecyclerView la posición que ocupa en el listado. La idea es que cuando asociamos un onClickListener este nos indica la vista pulsada, pero no la posición. De esta forma, sabiendo la vista conoceremos su posición. En la implementación anterior usábamos un método alternativo: posición = recyclerView.getChildAdapterPosition(vista). Pero tiene el inconveniente de necesitar el recyclerView. Y ahora no vamos a disponer de él.

En Kotlin tanto las clase como los atributos son por defecto cerrados . Por lo tanto, aparece un error al intentar heredar de AdaptadorLugares. Para resolverlo pulsa sobre la bombilla roja y selecciona *Make AdaptadorLugares Open*.

2. Añade a la clase LugaresBD el siguiente método estático:

```
public static Lugar extraeLugar(Cursor cursor) {
    Lugar lugar = new Lugar();
    lugar.setNombre(cursor.getString(1));
    lugar.setDireccion(cursor.getString(2));
    lugar.setPosicion(new GeoPunto(cursor.getDouble(3),
                                    cursor.getDouble(4)));
    lugar.setTipo(TipoLugar.values()[cursor.getInt(5)]);
    lugar.setFoto(cursor.getString(6));
    lugar.setTelefono(cursor.getInt(7));
    lugar.setUrl(cursor.getString(8));
    lugar.setComentario(cursor.getString(9));
    lugar.setFecha(cursor.getLong(10));
    lugar.setValoracion(cursor.getFloat(11));
    return lugar;
}

public Cursor extraeCursor() {
    String consulta = "SELECT * FROM lugares";
    SQLiteDatabase bd = getReadableDatabase();
    return bd.rawQuery(consulta, null);
}
```

El primer método crea un nuevo lugar con los datos de la posición actual de un Cursor. El segundo nos devuelve un cursor que contiene todos los datos de la tabla.

3. Abre la clase Aplicacion y reemplaza la declaración de la variable lugares y adaptador:

```
public LugaresBD lugares;
public AdaptadorLugaresBD adaptador;

@Override public void onCreate() {
    super.onCreate();
    lugares = new LugaresBD(this);
    adaptador= new AdaptadorLugaresBD(lugares, lugares.extraeCursor())
}
```

Hemos pasado adaptador dentro de Aplicacion para que sea accesible desde cualquier clase. Ahora la lista de lugares que el usuario a buscado en la base de datos estará almacenada dentro de adaptador. Y queremos acceder a esta lista desde varios sitios. En Kotlin la inicialización de las propiedades se recomienda realizarla en su declaración. Observa como para adaptador se utiliza by lazy, para indicar que la inicialización se realice cuando vallamos a utilizar la variable. De hacerlo inmediatamente corremos el peligro de que la base de datos no esté creada.

4. Añade en MainActivity la siguiente propiedad:

```
private RepositorioLugaresBD lugares;
private AdaptadorLugaresBD adaptador;

@Override protected void onCreate(Bundle savedInstanceState) {
    ...
    adaptador = ((Aplicacion) getApplication()).adaptador;
}
```

5. Reemplaza en MainActivity dentro de onCreate() en código subrayado:

JavaKotlin

```
adaptador.setOnItemClickListener(new View.OnClickListener() {

    @Override public void onClick(View v) {
        int pos =(Integer)(v.getTag());
        usoLugar.mostrar(pos);
    }
});
```

6. Ejecuta la aplicación y verifica que las listas se muestran correctamente.

7. Si pulsas sobre un elemento del RecyclerView posiblemente se producirá un error. Para solucionarlo abre la clase VistaLugarActivity y reemplaza:

JavaKotlin

```
lugar = lugares.elementoadaptador.lugarPosicion(pos);
```

Recuerda que el método elemento() todavía no ha sido implementado. Además, como ya hemos comentado, resulta más eficiente acceder a este lugar usando la lista almacenada en adaptador.

8. En la clase EdicionLugarActivity realiza la misma operación.

9. En la clase CasosUsoLugar pasaremos el adaptador al constructor:

```
private RepositorioLugaresBD lugares;
private AdaptadorLugaresBD adaptador;

public CasosUsoLugar(Activity actividad,
                      RepositorioLugares_LugaresBD lugares,
                      AdaptadorLugaresBD adaptador) {
    ...
    this.adaptador = adaptador;
}
```

10. Añade el nuevo parámetro en las llamadas a este constructor.

11. Ejecuta la aplicación y verifica que funciona. Puedes seleccionar un lugar e incluso editarlo, aunque si guardas una edición no se almacenarán los cambios. Lo arreglaremos más adelante.

Ejercicio: *Adaptando la actividad del Mapa al nuevo adaptador*

En este ejercicio adaptaremos la actividad MapaActivity para que use adecuadamente el nuevo adaptador basado en Cursor. El proceso es el mismo que acabamos de realizar: Reemplazaremos los accesos a Aplicacion.lugares por Aplicacion.adaptador.

1. Reemplaza inicialización de la variable lugares por adaptador:

```
private RepositorioLugares lugares;  
private AdaptadorLugaresBD adaptador;
```

```
@Override public void onCreate(Bundle savedInstanceState) {  
    ...  
    lugares = ((Aplicacion) getApplication()).lugares;  
adaptador = ((Aplicacion) getApplication()).adaptador;
```

2. Reemplaza el código del método onMapReady():

```
if (lugares.tamanyo() adaptador.getItemCount() > 0) {  
    GeoPunto p= lugares.elemento adaptador.lugarPosicion(0).getPosicion();  
    ...  
    for (int n=0; n< lugares.tamanyo() adaptador.getItemCount(); n++) {  
        Lugar lugar = lugares.elemento adaptador.lugarPosicion(n);
```

3. En el método onInfoWindowClick() reemplaza:

```
for (int pos=0; pos< lugares.tamaño() adaptador.getItemCount(); pos++){  
    if (lugares.elemento adaptador.lugarPosicion(pos).getNombre()
```

Como hemos indicado, la ventaja de este cambio es que no realizaremos nuevos accesos a la base de datos una vez obtenido el Cursor.

4. Verifica el funcionamiento de la actividad MapaActivity.

Práctica: *Probando consultas en Mis Lugares*

1. En el método extraeCursor() de la clase LugaresBD reemplaza el comando SELECT * FROM lugares por SELECT * FROM lugares WHERE valoracion>1.0 ORDER BY nombre LIMIT 4. Ejecuta la aplicación y verifica la nueva lista.
2. Realiza otras consultas similares. Si tienes dudas, puedes consultar en Internet la sintaxis del comando SQL SELECT.
3. Si quieres practicar el uso del método query(), puedes tratar de realizar una consulta utilizando este método.

Práctica: Añadir criterios de ordenación y máximo en Preferencias

1. Modifica el método extraeCursor() para que el criterio de ordenación y el máximo de lugares a mostrar corresponda con los valores que el usuario ha indicado en las preferencias.
2. Si el usuario escoge el primer criterio de ordenación has de dejar la consulta original sin introducir la cláusula “ORDER BY”.
3. Si escoge el orden por valoración este ha de ser descendiente, de más valorados a menos. Puedes usar la cláusula “ORDER BY valoracion DESC”.
4. Para ordenar por distancia puedes usar la siguiente consulta SQL:

```
"SELECT * FROM lugares ORDER BY " +
    "(" + lon + "-longitud)*(" + lon + "-longitud) + " +
    "(" + lat + "-latitud )*(" + lat + "-latitud )"
```

Donde las variables lon y lat han de corresponder con la posición actual del dispositivo. Esta ecuación es una simplificación que no tiene en cuenta que los polos están achitados, pero funciona de forma adecuada.

5. Si no actualizamos el cursor con la lista el cambio de preferencias no será efectivo hasta que salgas de aplicación y vuelvas a entrar. Para evitar este inconveniente, llama a la actividad PreferenciasActivity mediante startActivityForResult(). En el método onActivityResult() has de actualizar el cursor de adaptado e indicar todos los elementos han de redibujarse. Para esta última acción puedes utilizar adaptador.notifyDataSetChanged().

Solución:

1. Reemplaza en lugaresBD el siguiente método:

```
public Cursor extraeCursor() {
    SharedPreferences pref =
        PreferenceManager.getDefaultSharedPreferences(contexto);
    String consulta;
    switch (pref.getString("orden", "0")) {
        case "0":
            consulta = "SELECT * FROM lugares ";
            break;
        case "1":
            consulta = "SELECT * FROM lugares ORDER BY valoracion DESC";
            break;
        default:
            double lon = ((Aplicacion) contexto.getApplicationContext())
                .posicionActual.getLongitud();
            double lat = ((Aplicacion) contexto.getApplicationContext())
                .posicionActual.getLatitude();
            consulta = "SELECT * FROM lugares ORDER BY " +
                "(" + lon + "-longitud)*(" + lon + "-longitud) + " +
                "(" + lat + "-latitud )*(" + lat + "-latitud )";
            break;
    }
}
```

```
consulta += " LIMIT "+pref.getString("maximo","12");
SQLiteDatabase bd = getReadableDatabase();
return bd.rawQuery(consulta, null);
}
2. En MainActivity añade:
JavaKotlin
static final int RESULTADO_PREFERENCIAS = 0;

public void lanzarPreferencias(View view) {
    Intent i = new Intent(this, PreferenciasActivity.class);
    startActivityForResult(i, RESULTADO_PREFERENCIAS);
}

@Override protected void onActivityResult(int requestCode, int resultCode,
                                         Intent data) {
    if (requestCode == RESULTADO_PREFERENCIAS) {
        adaptador.setCursor(lugares.extraeCursor());
        adaptador.notifyDataSetChanged();
    }
}
```

Operaciones con bases de datos en Mis Lugares

Objetivos:

Aplicar las operaciones básicas con bases de datos en la aplicación Mis Lugares.

En los apartados anteriores hemos aprendido a crear una base de datos y a realizar consultas en una tabla. En este apartado vamos a continuar aprendiendo las operaciones básicas cuando trabajamos con datos. Estas son: altas, bajas y modificaciones y consultas.

Ejercicio: Consulta de un elemento en Mis Lugares

1. Reemplaza en la clase LugaresBD el método elemento() con el siguiente código. Su finalidad es buscar el lugar correspondiente a un id y devolverlo.

```
@Override public Lugar elemento(int id) {  
    Cursor cursor = getReadableDatabase().rawQuery(  
        "SELECT * FROM lugares WHERE _id = "+id, null);  
    try {  
        if (cursor.moveToFirst())  
            return extraeLugar(cursor);  
        else  
            throw new SQLException("Error al acceder al elemento _id = "+id);  
    } catch (Exception e) {  
        throw e;  
    } finally {  
        if (cursor!=null) cursor.close();  
    }  
}
```

Comenzamos inicializando el valor del lugar a devolver a null, para que corresponda con el valor devuelto en caso de no encontrarse el id. Luego se obtiene el objeto bd llamando a getReadableDatabase(). Este objeto nos permitirá hacer consultas en la base de datos. Por medio del método rawQuery() realiza una consulta en la tabla lugares usando el comando SQL SELECT * FROM lugares WHERE _id =... . Este comando podría interpretarse como, selecciona todos los campos de la tabla lugares, para el registro con el id indicado. El resultado de una consulta es un Cursor con el registro, si es encontrado, o en caso contrario, un Cursor vacío.

En la siguiente línea llamamos cursor.moveToNext() para que el cursor pase a la siguiente fila encontrada. Como es la primera llamada estamos hablando del primer elemento. Devuelve true si lo encuentra y false si no. En caso de encontrarlo llamamos a extraeLugar() para actualizar todos los atributos de lugar con los valores de la fila apuntada por el cursor. Si no lo encontramos lanzamos una excepción.

Es importante cerrar lo antes posibles el consumo de memoria. Lo hacemos en la sección finally para asegurarnos que se realiza siempre, haya habido una excepción o no.

NOTA: Este método no es usado por la aplicación. Ha sido añadido por pertenecer a la interfaz Lugares.

Ejercicio: Modificación de un lugar

Si tratas de modificar cualquiera de los lugares observarás que los cambios no tienen efecto. Para que la base de datos sea actualizada, realiza el siguiente ejercicio:

1. Añade en la clase LugaresBD en el método actualizaLugar() el siguiente código. Su finalidad es reemplazar el lugar correspondiente al id indicado por un nuevo lugar.

```
@Override public void actualiza(int id, Lugar lugar) {  
    getWritableDatabase().execSQL("UPDATE lugares SET" +  
        " nombre = '" + lugar.getNombre() +  
        "', direccion = '" + lugar.getDireccion() +  
        "', longitud = '" + lugar.getPosicion().getLongitud() +  
        ", latitud = '" + lugar.getPosicion().getLatitud() +  
        ", tipo = '" + lugar.getTipo().ordinal() +  
        ", foto = '" + lugar.getFoto() +  
        "', telefono = '" + lugar.getTelefono() +  
        ", url = '" + lugar.getUrl() +  
        "', comentario = '" + lugar.getComentario() +  
        "', fecha = '" + lugar.getFecha() +  
        ", valoracion = '" + lugar.getValoracion() +  
        " WHERE _id = " + id);  
}
```

2. En la clase EdicionLugarActivity, en el método onOptionsItemSelected() añade el código subrayado y elimina el tachado:

```
int _id = adaptador.idPosicion(pos);  
usoLugar.guardar(pos _id, lugar);
```

La variable pos corresponde a un indicador de posición dentro de la lista. Para utilizar correctamente el método actualiza() de LugaresBD, hemos de obtener el _id correspondiente a la primera columna de la tabla. Este cambio lo realiza el método idPosicion() de adaptador.

3. Ejecuta la aplicación y trata de modificar algún lugar. Observa que, cuando realizas un cambio en un lugar, estos parecen que no se almacenan. Realmente si que se han almacenado, el problema está en que el metodo adaptador no se ha actualizado. Para verificar que los cambios si que se han almacenado, has de salir de la aplicación y volver a lanzarla.

4. Para resolver el refresco de MainActivity has de añadir la siguiente línea al final del método guardar() de CasosUsoLugar:

JavaKotlin

```
adaptador.setCursor(lugares.extraeCursor());  
adaptador.notifyDataSetChanged();
```

Asignamos un nuevo cursor al adaptador y le indicamos que los datos han cambiado para que vuelva a crear las vistas correspondiente del RecyclerView.

5. Ejecuta de nuevo la aplicación. Tras editar un lugar, los cambios se reflejan en MainActivity pero no en VistaLugarActivity.

6. Para resolver este nuevo problema has de añadir las siguientes líneas en VistaLugarActivity:

```
public int _id = -1;

@Override public void onActivityCreated(Bundle state) {
    ...
    if (extras != null) pos = extras.getInt("pos", 0);
    else                pos = 0;
    _id = adaptador.idPosicion(pos);
    ...

@Override public void onActivityResult(int requestCode, int resultCode,...
    if (requestCode == RESULTADO_EDITAR) {
        lugar = lugares.elemento(_id);
        pos = adaptador.posicionId(_id);
        actualizaVistas();
    }
}
```

Necesitamos actualizar la variable lugar dado que esta acaba de ser modificada. Extraerla según su posición en el listado es potencialmente peligroso, dado que esta posición puede cambiar dinámicamente. Por ejemplo, si ordenamos los lugares por orden alfabético y modificamos su inicial, posiblemente cambie su posición. Por el contrario, el _id de un lugar nunca puede cambiar. Hemos obtenido el _id al crear la actividad. Tras la edición del lugar, con este _id, obtenemos los nuevos valores para lugar y buscamos la nueva posición a partir de _id.

7. Para hacer la última acción añade en AdaptadorLugaresBD la siguiente función:

```
public int posicionId(int id) {
    int pos = 0;
    while (pos < getItemCount()) return -1;
    else                  return pos;
}
```

Como ves se recorren todos los elementos del adaptador hasta encontrar uno con el mismo id. Si no es encontrado devolvemos -1.

8. Ejecuta la aplicación y verifica el nuevo funcionamiento.

Ejercicio: Modificación valoración y fotografía de un lugar

1. Algunos de los campos de un lugar no se modifica en la actividad EdicionLugarActivity, si no que se hacen directamente en VistaLugarActivity. En concreto la valoración, la fotografía y más adelante añadiremos fecha y hora. Cuando se modifiquen estos campos, también habrá que almacenarlos de forma permanente en la base de datos. Empezaremos por la valoración. Añade en el método actualizaVistas() de VistaLugarActivity el código subrayado.

```
@Override public void onRatingChanged(RatingBar ratingBar,
                                         float valor, boolean fromUser) {
    lugar.setValoracion(valor);
    usoLugar.actualizaPosLugar(pos, lugar);
```

```
pos = adaptador.posicionId( id);
}
```

Cuando el usuario cambie la valoración de un lugar se llamará a onRatingChanged() donde actualizamos la valoración y llamamos a actualizaLugares(). Esta función llamará a actualizaVistas(), donde cambiamos raingBar, lo que provocará una llamada al escuchador, y así sucesivamente entrando en bucle. Para evitarlo antes de cambiar el valor desactivamos el escuchador. Si tenemos seleccionada la ordenación por valoración, al cambiarla puede cambiar la posición del lugar en la lista. Por si ha cambiado, volvemos a obtener la variable pos.

2. Añade la siguiente función a CasosUsoLugar:

```
public void actualizaPosLugar(int pos, Lugar lugar) {
    int id = adaptador.idPosicion(pos);
    guardar(id, lugar); //
```

Primero obtenemos en la variable id el identificador del lugar. Para ello, vamos a usar la posición que el lugar ocupa en el listado. Con el id, ya podemos actualizar la base de datos. Como hemos visto, siempre que cambie algún contenido es importante que el adaptador actualice el Cursor, esto ya lo hace la función guardar().

3. Para que los cambios en las fotografías se actualicen también has obtener el lugar de forma adecuada y llamar a actualizaPosLugar():

```
public void ponerFoto(int pos, String uri, ImageView imageView) {
    Lugar lugar = lugares.elemento adaptador.lugarPosicion(pos);
    lugar.setFoto(uri);
    visualizarFoto(lugar, imageView);
    actualizaPosLugar(pos, lugar);
}
```

4. Verifica que tanto los cambios de valoración como de fotografía se almacenan correctamente.

Ejercicio: Alta de un lugar

En este ejercicio aprenderemos a añadir nuevos registros a la base de datos.

1. Reemplaza en la clase LugaresBD el método nuevo() por el siguiente. Su finalidad es crear un nuevo lugar en blanco y devolver el id del nuevo lugar.

```
@Override public int nuevo() {
int _id = -1;
Lugar lugar = new Lugar();
getWritableDatabase().execSQL("INSERT INTO lugares (nombre, " +
    "direccion, longitud, latitud, tipo, foto, telefono, url, " +
    "comentario, fecha, valoracion) VALUES ('', '', '' +
    lugar.getPosicion().getLongitud() + "", +
    lugar.getPosicion().getLatitud() + "", "+ lugar.getTipo().ordinal()+
    ", '', 0, '', '' , " + lugar.getFecha() + ", 0)");
Cursor c = getReadableDatabase().rawQuery(
    "SELECT _id FROM lugares WHERE fecha = " + lugar.getFecha(), null);
if (c.moveToNext()) _id = c.getInt(0);
```

```

        c.close();
        return _id;
    }
}

```

Comenzamos inicializando el valor del `_id` a devolver a -1. De esta manera, si hay algún problema este será el valor devuelto. Luego se crea un nuevo objeto Lugar. Si consultas el constructor de la clase, observarás que solo se inicializan posicion, tipo y fecha. El resto de los valores serán una cadena vacía para String y 0 para valores numéricos. Acto seguido, se crea una nueva fila con esta información. Los valores de texto y numéricos tampoco se indican, al inicializarse de la misma manera.

El método ha de devolver el `_id` del elemento añadido. Para conseguirlo se realiza una consulta buscando una fila con la misma fecha que acabamos de introducir.

2. Para la acción de añadir vamos a utilizar el botón flotante que tenemos desde la primera versión de la aplicación. Abre el fichero `res/layout/activity_main.xml` y reemplaza el ícono aplicado a este botón:

```

<android.support.design.widget.FloatingActionButton
    ...
    android:src="@android:drawable/ic_input_add"
    ... />

```

3. Abre la clase `MainActivity` y dentro de `onCreate()` comenta el código tachado y añade el subrayado, para que se ejecute al pulsar el botón flotante:

```

fab.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
            .setAction("Action", null).show();
        usoLugar.nuevo();
    }
});

```

4. Añade el siguiente caso de uso en `CasoUsoLugar`:

```

public void nuevo() {
    int id = lugares.nuevo();
    GeoPunto posicion = ((Aplicacion) actividad.getApplication())
        .posicionActual;
    if (!posicion.equals(GeoPunto.SIN_POSICION)) {
        Lugar lugar = lugares.elemento(id);
        lugar.setPosicion(posicion);
        lugares.actualiza(id, lugar);
    }
    Intent i = new Intent(actividad, EdicionLugarActivity.class);
    i.putExtra("_id", id);
    actividad.startActivity(i);
}

```

Comenzamos creando un nuevo lugar en la base e datos cuyo identificador va a ser `_id`. La siguiente línea obtiene la posición actual. Si el dispositivo está localizado, obtenemos el lugar recién creado, cambiamos su posición y lo volvemos a guardar. A continuación vamos a lanzar la actividad `EdicionLugarActivity` para que el usuario rellene los datos del lugar. Hasta ahora hemos utilizado el extra "pos""id" para indicar la posición en la lista del objeto a editar. Pero ahora esto no es posible, dado que este nuevo lugar no ha sido añadido a la lista. Para resolver

el problema vamos a crear un nuevo extra, “_id”, que usaremos para identificar el lugar a editar por medio de su campo _id.

5. En la clase EdicionLugarActivity añade el código subrayado:

```
private int _id;

@Override protected void onCreate(Bundle savedInstanceState) {
    ...
    Bundle extras = getIntent().getExtras();
    pos = extras.getInt("pos", -1);
    _id = extras.getInt("_id",-1);
    if ( id!= -1) lugar = lugares.elemento( id);
    else         lugar = adaptador.lugarPosicion(pos);
    actualizaVistas();
}
```

Esta actividad va a poder ser llamada de dos formas alternativas: usando el extra “pos” “id” para indicar que el lugar a modificar ha de extraerse de una posición del adaptador; o usando “_id” en este caso el lugar será extraído de la base de datos usando su identificador. Observa como se han definido dos variables globales, id e “pos” e _id. Aunque solo una se va a inicializar y la otra valdrá -1.

6. Cuando el usuario pulse la opción guardar se llamará al método onOptionsItemSelected(). Para almacenar la información tendremos que verificar cual de las dos variables ha sido inicializada. Añade el código subrayado en este método:

```
case R.id.accion_guardar:
    ...
    if ( id== -1) int _id = adaptador.idPosicion(pos);
    usoLugar.guardar(_id, lugar);
    finish();
    return true;
```

El primer if es añadido dado que si nos han pasado el identificador _id ya no tiene sentido obtenerlo a partir de la posición.

7. Verifica que los cambios introducidos funcionan correctamente.

Ejercicio: Baja de un lugar

En este ejercicio aprenderemos a eliminar registros a la base de datos.

1. Reemplaza en la clase LugaresBD el método borrar() por el siguiente. Su finalidad es eliminar el lugar correspondiente al id indicado.

```
public void borrar(int id) {
    getWritableDatabase().execSQL("DELETE FROM lugares WHERE _id = " + id);
}
```

2. Añade en la clase VistaLugarActivity, dentro del método onOptionsItemSelected(), el código subrayado:

```
case R.id.accion_borrar:
    int _id = adaptador.idPosicion(pos);
    usoLugar.borrar(pos _id)
    return true;
```

3. En CasosUsoLugares, dentro de borrarLugar(), añade las dos líneas subrayadas para actualizar el cursor y notificar al adaptador que los datos han cambiado:

```
lugares.borrar(id);
adaptador.setCursor(lugares.extraeCursor());
adaptador.notifyDataSetChanged();
actividad.finish();
```

4. Ejecuta la aplicación y trata de dar de baja algún lugar.

Ejercicio: *Opción CANCELAR en el alta de un lugar*

Si seleccionas la opción nuevo y en la actividad EdicionLugarActivity seleccionas la opción CANCELAR, puedes verificar que esta opción funciona mal. Los datos introducidos no se guardarán; sin embargo, se creará un nuevo lugar con todos sus datos en blanco. Para verificarlo has de salir de la aplicación para que se recargue el adaptador.

Para evitar este comportamiento, borra el elemento nuevo cuando se seleccione la opción CANCELAR. Pero este comportamiento ha de ser diferente cuando el usuario entró en la actividad EdicionLugarActivity para editar un lugar ya existente. Para diferenciar estas dos situaciones puedes utilizar los extras _id y pos.

Solución:

Añade en el método onOptionsItemSelected() en la opción accion_cancelar:

```
if (_id!=-1) lugares.borrar(_id)
```

Preguntas de repaso: [SQLite II](#)

[1] <http://youtu.be/ekn9j2J9sos>

Los Fragments

video [Los Fragments en Android.](#)

Objetivos:

Mostrar como usando fragments podemos diseñar elementos reutilizables de la IU

Con la popularización de las tabletas surgió el problema de desarrollar simultáneamente una aplicación para ser ejecutada tanto en un móvil como en una tableta. En otros sistemas, como iOS, se decidió que el desarrollador tenía que implementar dos aplicaciones diferentes. Android siguió con la estrategia de usar recursos alternativos para adaptarse a los diferentes tamaños de pantalla. Las herramientas vistas hasta ahora no resultan suficientes: cuando se diseña una interfaz de usuario específica para una tableta, no solo es preciso adaptar el tamaño de letra o los márgenes, sino que también es necesario reestructurar cómo se muestra la información en pantalla. En una tableta pueden caber muchos elementos de diseño al mismo tiempo, mientras que en un móvil estamos más limitados. Por ejemplo, podríamos diseñar dos elementos de la interfaz de usuario: uno que nos permitiera elegir entre una lista de lugares y otro que mostrara los detalles de uno de esos lugares. En una tableta se podrían mostrar ambos elementos a la vez, mientras que en un móvil tendríamos que mostrar primero uno y luego el otro.

Para resolver este problema, en la versión 3.0 de Android se introdujeron los *fragments*. Los *fragments* son bloques de interfaz de usuario que pueden utilizarse en diferentes sitios, simplificando así la composición de una interfaz de usuario. Los *fragments* nos permiten diseñar y crear cada uno de los elementos de nuestra aplicación por separado. Luego, dependiendo del tamaño de pantalla disponible, mostraremos uno solo o más de uno a la vez.



Figura 1. Uso de fragments en tableta y móvil.

Es importante resaltar que no cambia el papel de las actividades. Sigue siendo el elemento básico que representa cada pantalla de una aplicación y nos permite navegar por ella. La novedad introducida es que cuando diseñemos una actividad, esta puede estar formada por uno o más *fragments*.

Cuando diseñemos un *fragment*, este ha de gestionarse a sí mismo, recibiendo eventos de entrada y modificando su vista sin necesidad de que la actividad que lo contiene intervenga. De esta forma, el *fragment* se podrá utilizar en diferentes actividades sin tener que modificar el código.

fragments. El problema es que esta característica aparece en una versión que todavía no está disponible en muchos dispositivos. Para resolver este problema se ha creado una librería de compatibilidad para poder utilizar *fragments* en versiones anteriores a la 3.0. Esta librería se incluye de manera automática a un proyecto, siempre que el requerimiento mínimo de SDK sea inferior al nivel 11 (3.0); pero lo desarrollemos con una versión superior a la 3.0 (*Target SDK*). Para verificar esto, abre el proyecto creado en el ejercicio anterior. Observa cómo esta librería se incluye en *libs/android-support-v4.jar*.

Cada *fragment* ha de implementarse en una clase diferente. Esta clase tiene una estructura similar a la de una actividad, pero con algunas diferencias. La primera es que esta clase tiene que extender Fragment. El ciclo de vida es muy parecido al de una actividad; sin embargo, dispone de unos cuantos eventos más, que le indican cambios en su estado con respecto a la actividad que lo contiene. El ciclo de vida de un *fragment* va asociado al de la actividad que lo contiene (por ejemplo, si la actividad es destruida, todos los *fragments* que contiene son destruidos); pero también es posible destruir un *fragment* sin modificar el estado de la actividad.

Los *fragments* suelen mostrar una vista (aunque esto no es imprescindible). Es recomendable definir esta vista en un fichero XML de recursos. Por lo tanto, para crear un *fragment* usaremos una clase Java para definir su comportamiento y un fichero XML para definir su apariencia.

Los *fragments* se pueden introducir en una actividad de dos formas diferentes: por código o desde XML. Ambas formas tienen sus ventajas y sus inconvenientes. Introducir un *fragment* desde XML es más sencillo. Además, el diseño queda diferenciado del código, simplificando el trabajo del diseñador. Sin embargo, trabajar de esta forma tiene un inconveniente: una vez introducido ya no podremos reemplazar el *fragment* por otro. Por lo tanto, un *fragment* añadido desde XML será siempre estático. Si lo añadimos desde código, ganamos la posibilidad de intercambiar el *fragment* por otro. En los siguientes ejercicios veremos cómo añadir *fragments* desde XML.

Uso de Fragments en Mis Lugares

Objetivos:

Mejorar el interfaz de usuario de la aplicación Mis Lugares cuando se ejecuta en una tableta.

Ejercicio: Un primer fragment.

En este ejercicio modificaremos la aplicación Mis Lugares, pero ahora trabajando con un *fragment*. Su funcionalidad será idéntica. La ventaja de definir *fragments* en vez de actividades es que podemos mostrar varios *fragments* a la vez en la pantalla, pero no varias actividades.

1. En este apartado vamos a realizar un número importante de modificaciones y es posible que algo salga mal. Puede ser un buen momento para realizar una copia del proyecto actual. Así, siempre dispondremos de una versión operativa. Para ello, desde el explorador de ficheros de tu sistema operativo, realiza una copia de la carpeta que contiene el proyecto. Para acceder rápidamente a esta carpeta desde el explorador del proyecto, pulsa en *app* con el botón derecho y selecciona *Show in Explorer*.
2. En este ejercicio vamos a mostrar en un fragment lo que antes se mostraba en *MainActivity*. Por lo tanto, podemos reutilizar su *layout* en XML para nuestro *fragment*. Copia el fichero *content_main.xml* en *fragment_selector.xml*. Desde el explorador del proyecto usa *Ctrl-C* y *Ctrl-V*.
3. Ahora nos falta definir la clase para el *fragment*. Crea una nueva clase llamada *SelectorFragment* y rellénala con el siguiente código:

```
public class SelectorFragment extends Fragment {  
    private LugaresBD lugares;  
    private AdaptadorLugaresBD adaptador;  
    private CasosUsoLugar usoLugar;  
    private RecyclerView recyclerView;  
  
    @Override  
    public View onCreateView(LayoutInflater inflador, ViewGroup contenedor,  
                             Bundle savedInstanceState) {  
        View vista = inflador.inflate(R.layout.fragment_selector,  
                                     contenedor, false);  
        recyclerView = vista.findViewById(R.id.recyclerView);  
        return vista;  
    }  
}
```

```

@Override
public void onActivityCreated(Bundle state) {
    super.onActivityCreated(state);
    lugares = ((Aplicacion) getActivity()).getApplication().lugares;
    adaptador = ((Aplicacion) getActivity()).getApplication().adaptador;
    usoLugar = new CasosUsoLugar(getActivity(), lugares, adaptador);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
    recyclerView.setAdapter(adaptador);
    adaptador.setOnItemClickListener(new View.OnClickListener() {
        @Override public void onClick(View v) {
            int pos = (Integer)(v.getTag());
            usoLugar.mostrar(pos);
        }
    });
}
}

```

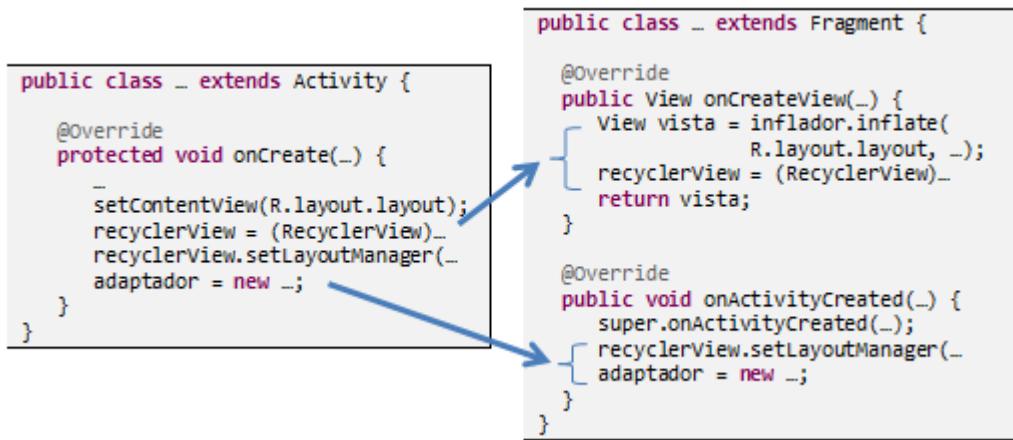
NOTA: Tras incluir nuevas clases tendrás que indicar los imports adecuados. Para que Android Studio lo haga automáticamente pulsa **Alt-Intro**. La clase Fragment aparece en dos paquetes, por lo que te pedirá que selecciones uno de los dos. Utiliza el segundo, que corresponde a la librería de compatibilidad:

android.app.Fragment
 android.support.v4.app.Fragment

El código de esta clase es similar al que teníamos antes en `MainActivity`, salvo que ahora extendemos a `Fragment` en vez de a `AppCompatActivity`, y que los métodos del ciclo de vida son diferentes.

Al igual que en una actividad, un *fragment* también tiene una vista asociada. En la actividad asociábamos la vista en el método `onCreate()`, llamando a `setContentView()`. En un *fragment* también disponemos del método `onCreate()`, pero no es aquí donde hay que asociar la vista. Se ha creado un nuevo método en el ciclo de vida, `onCreateView()`, con la finalidad de asociar su vista. En este método se nos pasan tres parámetros: un `LayoutInflater` que nos permite crear una vista a partir de un layout XML, el contenedor donde será insertado el *fragment* (en el punto siguiente veremos que se trata de un `LinearLayout`) y posibles valores guardados de una instancia anterior^[1]. El método `onCreateView()` ha de devolver la vista ya creada. El hecho de disponer de este método va a resultar muy interesante, dado que nos va a permitir cambiar la vista de un *fragment* sin tener que volverlo a crear.

Por otra parte `onActivityCreated()` es llamado cuando la actividad que contiene el *fragment* termina de crearse. Aprovecharemos este método para realizar tareas de inicialización, como por ejemplo crear el adaptador y asociarlo al `RecyclerView`. Observa como la forma de trabajar con un `RecyclerView` es diferente cuando lo hacemos desde una actividad que extiende `Fragment`. Aunque, ha de quedar claro, que en el fondo se realiza la misma tarea. En el siguiente esquema se compara cómo asociar el *layout*, el `RecyclerView` tanto en una actividad como en un *fragment*.



4. La actividad MainActivity va a visualizar el *layout content_main.xml*. Reemplaza su contenido por el siguiente código:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">
    <fragment
        android:id="@+id/selector_fragment"
        android:name= "com.example.mislugares.presentacion.SelectorFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />
</LinearLayout>

```

Como puedes ver, introducir el *fragment* desde un XML es muy sencillo. Simplemente añadimos una etiqueta *<fragment>* y en el atributo name indicamos el nombre de la clase del *fragment*. Este *fragment* es introducido en un *LinearLayout* que actuará de contenedor. Es habitual usar un contenedor para poder añadir nuevos *fragments* o reemplazarlos. Es importante incluir el atributo *layout_behavior* para su correcto funcionamiento dentro del *CoordinatorLayout*.

5. Elimina en MainActivity la declaración de las variables globales recyclerView. En el método onCreate() elimina las líneas que inicializan recyclerView y la asignación del evento onClick para adaptador.

6. Ejecuta el proyecto. Verifica que la aplicación tiene la misma funcionalidad que antes.

NOTA: Puede parecer que no hemos conseguido gran cosa, dado que al final el funcionamiento es idéntico. Aunque resulta más complejo trabajar con fragments, Google recomienda que siempre diseñemos los elementos del IU basados en fragments, en lugar de en actividades. De esta forma, tendremos la posibilidad de mostrar varios elementos del IU a la vez en pantalla.

Ejercicio: Implementando un segundo fragment.

Recordemos que la aplicación que queremos hacer tiene que mostrar una serie de lugares, y que al pulsar sobre uno de ellos, nos muestre la información detallada sobre él. En este ejercicio crearemos un segundo *fragment* para mostrar la información de un lugar, utilizando como base la actividad VistaLugarActivity. Haremos también que MainActivity muestre simultáneamente los dos *fragments* que hemos creado.

1. Desde el explorador del proyecto copia la actividad VistaLugarActivity (Ctrl-C) y realiza una copia (Ctrl-V) con nombre VistaLugarFragment.
2. Haz que la nueva clase herede de Fragment en lugar de AppCompatActivity.

Nota: Importa esta clase del paquete android.support.v4.app.

3. Elimina el método onCreate() y distribuye su código entre los siguientes:

```
@Override public View onCreateView(LayoutInflater inflador,  
        ViewGroup contenedor, Bundle savedInstanceState) {  
    setHasOptionsMenu(true);  
    View vista = inflador.inflate(R.layout.vista_lugar, contenedor, false);  
    return vista;  
}  
  
@Override public void onActivityCreated(Bundle state) {  
    super.onActivityCreated(state);  
    Bundle extras = getActivity().getIntent().getExtras();  
    if (extras != null) {  
        pos = extras.getInt("pos", 0);  
        actualizaVistas();  
    }  
}
```

El layout que visualizará el fragment es el mismo que usábamos en la actividad pero ahora es asignado en el método onCreateView(). La recogida de parámetros y la inicialización se realiza en onActivityCreated().

4. **En Java** añade la variable global:

```
private View v;
```

De esta forma, cada vez que queramos acceder a la vista del *fragment*, podremos usar esta variable en lugar del método getView().

6. **En Java** al principio de actualizaVistas() añade:

```
View v = getView();
```

Reemplaza todas las apariciones de findViewById() por v.findViewById(). Este método no está en la clase Fragment, pero si que está en la clase View.

7. En Java cambia el modificador de onActivityResult() de protected a public. Para poder sobreescribir un método es imprescindible que uses los mismos modificadores, y para el método en cuestión son diferentes en la clase Activity que en Fragment.

8. Reemplaza el método onCreateOptionsMenu() por el siguiente:

```
@Override
```

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {  
    inflater.inflate(R.menu.vista_lugar, menu);  
    super.onCreateOptionsMenu(menu, inflater);
```

Desde un *fragment* también podemos añadir ítems de menú a la actividad. El procedimiento es muy parecido, solo cambia el perfil del método.

9. Ya no estamos en una actividad has de reemplazar las apariciones de *this* por una referencia a la actividad del fragment:

```
Toast.makeText(this getActivity(), "...
```

También:

```
getActivity().finish();
```

10. Modifica *content_main.xml* para que muestre ambos *fragments*. Para ello añade el siguiente elemento al final del LinearLayout:

```
<fragment  
    android:id="@+id/vista_lugar_fragment"  
    android:name="com.example.mislugares.presentacion.VistaLugarFragment"  
    android:layout_width="0dp"  
    android:layout_height="match_parent"  
    android:layout_weight="1" />
```

11. Abre la clase *MainActivity* y al final del método *onOptionsItemSelected()* asegúrate que el valor devuelto de la siguiente forma:

```
return super.onOptionsItemSelected(item);
```

De esta manera permitimos que el sistema pregunte a los *fragments* si tienen que procesar la selección de un ítem de menú.

12. Ejecuta la aplicación. Podrás ver como se muestran los dos *fragments* uno al lado del otro. De momento, el *fragment* de la derecha no muestra información de ningún lugar concreto.

Ejercicio: Modificar el contenido de un *fragment* desde otro.

La aplicación creada hasta ahora no funciona correctamente cuando se visualizan los dos *fragments*. En este ejercicio vamos a conseguir que cuando se pulse sobre un elemento de la lista, el *fragment* de la derecha visualice la información del lugar seleccionado.

1. Reemplaza en *CasosUsoLugares* el siguiente método:

```
public void mostrar(int pos) {  
    VistaLugarFragment fragmentVista = obtenerFragmentVista();  
    if (fragmentVista != null) {  
        fragmentVista.pos = pos;  
        fragmentVista._id = adaptador.idPosicion(pos);  
        fragmentVista.actualizaVistas();  
    } else {  
        Intent intent = new Intent(actividad, VistaLugarActivity.class);  
        intent.putExtra("pos", pos);  
        actividad.startActivityForResult(intent, 0);  
    }
```

```

}

public VistaLugarFragment obtenerFragmentVista() {
    FragmentManager manejador = actividad.getSupportFragmentManager();
    return (VistaLugarFragment)
        manejador.findFragmentById(R.id.vista_lugar_fragment);
}

```

Este método ha de visualizar el lugar solicitado. Comenzamos obteniendo una referencia al fragment con id `vista_lugar_fragment`. Si existe este fragmento quiere decir que está ahora en pantalla y no es necesario crear una nueva actividad. Simplemente cambiando pos e `_id`, y llamando al método `actualizarVistas()` conseguimos que se muestre la información en el *fragment* ya existente. En caso de que este fragment no exista (esto podrá pasar tras hacer uno de los próximos ejercicios), creamos una nueva actividad para mostrar la información.

2. En Java para poder acceder a la propiedad `pos` de `VistaLugarFragment` cambia el modificador `private` por `public`.

3. Observa como aparece un error al tratar de obtener `supportFragmentManager`. Estamos trabajando con una variable de tipo `Activity`, pero este método solo está disponible en su descendiente `FragmentActivity`. Para resolverlo cambia el tipo de la propiedad.

```

public class CasosUsoLugar {
    private FragmentActivity actividad;
    ...
    public CasosUsoLugar(FragmentActivity actividad, LugaresBD lugares ...

```

Este cambio implica que ya solo podremos usar estos casos de uso desde una actividad de la clase `FragmentActivity`. Nosotros lo estábamos haciendo desde `AppCompatActivity`. No vamos a tener problemas al tratarse de un descendiente de `FragmentActivity`.

4. Ejecuta la aplicación y verifica que puedes cambiar el fragment de la derecha.

5. Si vas a preferencias y cambias el criterio de ordenación y acto seguido modificas la valoración del lugar en pantalla. Es posible que este se duplique en la lista de la izquierda. El problema se debe a que las variables `pos` y `_id` de `VistaLugarFragment` no tenían el valor correcto tras alterar el orden de la lista.

6. Para arreglarlo añade en `MainActivity` dentro de `onActivityResult()`:

```

if (requestCode == RESULTADO_PREFERENCIAS) {
    adaptador.cursor = lugares.extraeCursor();
    adaptador.notifyDataSetChanged();
    if (usoLugar.obtenerFragmentVista() != null)
        usoLugar.mostrar(0);
}

```

En Kotlin el uso de ; es opcional. Lo que hacemos es averiguar si estamos visualizando dos fragments y en ese caso mostraremos en el fragment de la derecha el primer lugar de la lista. En caso contrario, solo se visualiza la lista y no existe VistaLugarFragment.

Ejercicio: Adaptar CasosUsoLugares a fragments

La clase CasosUsoLugares estaba pensada para ser usada desde una actividad. De hecho, era uno de los parámetros que se pasaban en el constructor. Gracias a este parámetro no solo se extraía el contexto, sino que también se usaba para invocar a startActivityForResult() para arrancar nuevas actividades y que luego se devuelva información a la actividad adecuada.

Pero ahora la situación ha cambiado, estos casos de uso no solo pueden ser utilizados por actividades, sino también por fragments. En este ejercicio vamos a introducir los cambios necesarios para que la respuesta de startActivityForResult() sea recogida por la actividad o fragment que está utilizando la clase.

1. Añade el siguiente atributo en CasosUsoLugares:

```
public class CasosUsoLugar {  
    protected Fragment fragment;  
    ...  
    public CasosUsoLugar(FragmentActivity actividad, Fragment fragment,  
                         LugaresBD lugares, AdaptadorLugaresBD adaptador) {  
        this.fragment = fragment;  
        ...  
    }  
}
```

La idea es que cuando se use desde una actividad el parámetro fragment se pase como null, y si es desde un fragment se pasarán tanto el parámetro actividad como fragment.

2. Reemplaza, todas las apariciones de actividad.startActivityForResult por:

```
if (fragment != null) fragment.startActivityForResult(...);  
else actividad.startActivityForResult(...);
```

Si nos han indicado un fragment llamamos desde este para que nos devuelva el resultado a este. En caso contrario lo hacemos desde la actividad.

3. En MainActivity, VistaLugarActivity y EdicionLugarActivity añade como nuevo parámetro null:

```
usoLugar = new CasosUsoLugar(this, null, lugares, adaptador);
```

4. En VistaLugarFragment y SelectorFragment añade como nuevo parámetro this:

```
usoLugar = new CasosUsoLugar(getActivity(), this, lugares, adaptador);
```

5. Para que onActivityResult() se llame en la actividad y en los fragments has de llamar al super en MainActivity, al principio del método:

```
super.onActivityResult(requestCode, resultCode, data);
```

Ejercicio: Introducir escuchadores manualmente en el fragment.

Cuando definimos el *layout vista_lugar.xml* utilizamos el atributo `onClick` en varias vistas para asociar métodos que se ejecutan al pulsar sobre la vista. El problema es que estos métodos solo pueden ser definidos en una actividad y no en un fragment. Cuando diseñamos un *fragment* hemos de conseguir que sea reutilizable, por lo que todo su comportamiento ha definirse en la clase del *fragment*. Para resolver este problema, vamos a programar los escuchadores manualmente, en lugar de utilizar el atributo `onClick`.

Más información sobre `onClick` y escuchadores de eventos en[\[2\]](#)

1. Abre el *layout vista_lugar.xml* y localiza el siguiente fragmento de código. Elimina la línea tachada y asegúrate que coincida el id:

```
<LinearLayout  
    android:id="@+id/barra_url"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:onClick="verPgWeb"  
    android:orientation="horizontal" >
```

2. Abre la clase *VistaLugarFragment* y añade en el método `onActivityCreated()` el siguiente código. **En Java** tras obtener v:

```
v.findViewById(R.id.barra_url).setOnClickListener(new OnClickListener () {  
    public void onClick(View view) { usoLugar.verPgWeb(lugar); } });
```

NOTA: Cuando pulses **Alt-Intro** para incluir los imports de las nuevas clases, selecciona el paquete marcado.

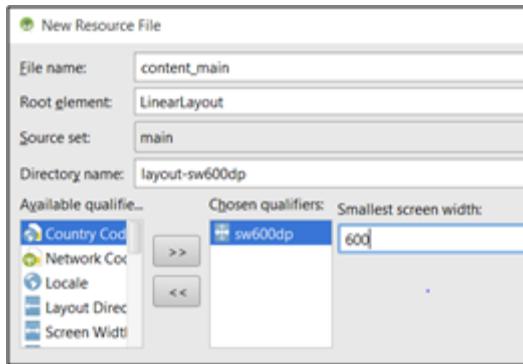
3. Ejecuta la aplicación y verifica que al pulsar en la vista de un lugar, sobre la url o su ícono se abre la página web correspondiente.

4. Repite esta operación para todas las vistas del *layout* donde se haya utilizado el atributo `onClick`.

Ejercicio: Mostrar dos fragments solo con pantallas grandes.

Cuando ejecutamos la aplicación en una pantalla pequeña, como la de un teléfono, no tienen ningún sentido mostrar dos *fragments* simultáneamente. Esto solo nos interesa en una tableta. Para conseguir este doble funcionamiento vamos a trabajar con dos *layouts* diferentes. Cargaremos uno u otro aprovechando los recursos alternativos de Android.

1. En el explorador del proyecto, pulsa con el botón derecho sobre la carpeta *res/layout*. y selecciona *New > Layout Resource File*. En *File name:* introduce *content_main*; en *Available qualifiers:* selecciona *Smallest Screen Width*; y el valor *600*:



Se creará la carpeta `res/layout-sw600dp`. Los recursos de esta carpeta se cargarán cuando la aplicación se ejecute en una pantalla de 7' o más.

2. Realiza una copia del contenido de `content_main.xml` por defecto al nuevo recurso que acabas de crear.
3. Elimina en el `content_main.xml` por defecto el segundo de los dos fragments que contiene.
4. Ejecuta la aplicación en un dispositivo de pantalla pequeña y en uno de más de 7'. Observa cómo se muestra uno o dos fragments según el tamaño de pantalla.

Práctica: Simplificación de la actividad `VistaLugar`.

Si comparas el código de la actividad `VistaLugarActivity` con el de `VistaLugarFragment` verás que son casi idénticos. Dejar el mismo código en dos clases diferentes es un grave error de programación. Trata de modificar la actividad `VistaLugarActivity` para que se limite a visualizar el fragment `VistaLugarFragment` en su interior.

Solución:

Crea el Layout `activity_vista_lugar.xml`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <fragment
        android:id="@+id/vista_lugar_fragment"
        android:name="com.example.mislugares.presentacion.VistaLugarFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:clickable="true"/>
</LinearLayout>
En VistaLugarActivity: solo ha de estar el método onCreate() que cargue el layout:
setContentView(R.layout.activity_vista_lugar);
```

Ejercicio: Ajustando comportamiento al borrar un lugar con fragments.

Dentro de la clase CasosUsoLugar, el código para borrar un lugar termina con:

```
actividad.finish();
```

Si trabajamos solo en una actividad, el funcionamiento es correcto. Tras borrar el lugar cerramos la actividad dado que no tiene sentido mostrar un lugar que ya no existe. Pero si trabajamos con dos fragments visualizándose juntos, al cerrar la actividad se cerrarán los dos y saldremos de la aplicación. En el presente ejercicio corregiremos este comportamiento no deseado.

1. En el método borrar() reemplaza actividad.finish() por el código siguiente:

```
if (obtenerFragmentSelector() == null) {  
} else {  
  
    mostrar(0);  
}
```

Tras borrar el lugar trataremos de averiguar si estamos en una configuración con dos fragments. Esto ocurrirá cuando podemos obtener una referencia de selector_fragment. Si no existe, realizamos la misma acción de antes. Si existe, hacer que se muestre un lugar con código diferente del borrado, en este caso se muestra el primero de la lista.

2. Añade la siguiente función:

```
fun obtenerFragmentSelector(): SelectorFragment? {  
    val manejador = actividad.supportFragmentManager  
    return manejador.findFragmentById(R.id.selector_fragment) as  
        SelectorFragment?  
}
```

3. Ejecuta la aplicación y verifica el resultado.

4. Si borrar todos los lugares de la lista, trabajando con una tableta, verás que se produce un error. Ni siquiera podrás volver a arrancar la aplicación.

5. Para resolverlo añade en VistaLugarFragment al comienzo de actualizaVistas:

JavaKotlin

```
if (adaptador.getItemCount() == 0) return
```

6. Ejecuta la aplicación y verifica el resultado.

Dialogos de selección de fecha y hora

Objetivos:

Mostrar el uso de cuadros de diálogo para seleccionar fechas y horas.

Los cuadros de dialogo fueron introducidos en el ejercicio *Un cuadro de dialogo para indicar el id de lugar*. En esa ocasión aprendimos a realizar un cuadro de dialogo personalizado. En este apartado aprenderemos a utilizar cuadros de diálogo específicos para trabajar con fechas y horas. Empezaremos introduciendo algunos conceptos y clases que nos ayudarán a trabajar con este tipo de información.

Clases para trabajar con fechas en Java:

Clase Date[\[1\]](#)

La clase Date representa un instante en el tiempo con una precisión de milisegundos. Se utiliza un sistema de medición del tiempo independiente de la zona horaria, conocido como UTC (Tiempo Universal Coordinado). El estándar de medición de tiempo UTC utiliza el tiempo en el meridiano de Greenwich independientemente de dónde nos encontremos. De esta forma, se evitan los problemas que aparecen cuando se comunican dos sistemas con mediciones locales de tiempo diferentes.

Para representar un instante de tiempo se suele utilizar la codificación conocida como “Tiempo Unix”. Esta codificación consiste en medir el número de milisegundos transcurridos desde el 1 de enero de 1970. Para almacenar este valor se utiliza un entero de 64 bits, en Java la palabra reservada long representa a un entero de este tipo. Si quieras en Android obtener el tiempo actual en este formato utiliza el método currentTimeMillis() de la clase System.

```
long ahora = System.currentTimeMillis();
Date fecha = new Date(ahora);
```

Clase DateFormat[\[2\]](#)

La clase Date está pensada para contar el tiempo de forma Universal en toda la tierra, de forma que sea sencilla de manipular por una máquina. Sin embargo, las personas utilizamos una medición del tiempo que depende de la zona horaria donde estemos o incluso dependerá de si el país donde estemos utiliza el horario de verano. Cuando tengas que mostrar o solicitar una fecha a una persona, deberás utilizar la representación del tiempo a la que está acostumbrada. En este caso la clase abstracta DateFormat o su descendiente SimpleDateFormat[\[3\]](#) te serán de gran ayuda para este propósito.

A continuación se muestra un ejemplo sencillo:

```
DateFormat df = new SimpleDateFormat("dd/MM/yy");
String salida = df.format(fecha);
```

Clase Calendar[4]

Como hemos comentado, la clase Date utiliza internamente un simple entero para representar un instante de tiempo. Por el contrario, los humanos nos complicamos algo más dado que usamos la combinación de varios campos: como año, mes, día, hora, minuto y milisegundo. Utiliza la clase Calendar para obtener estos campos desde un objeto Date. A diferencia de la clase Date, la clase Calendar depende de la configuración local del dispositivo (locale). Para obtener, la fecha actual según la representación local del dispositivo utiliza el método getInstance():

```
Calendar calendario = Calendar.getInstance();
calendario.setTimeInMillis(ahora);
int hora = calendario.get(Calendar.HOUR_OF_DAY);
int minuto = calendario.get(Calendar.MINUTE);
```

La clase Calendar es una clase abstracta, que en principio te permitiría trabajar con cualquier clase de calendario (como el calendario maya o el musulmán). No obstante, el calendario usado oficialmente en casi todo el mundo es el calendario Gregoriano, definido en la clase [GregorianCalendar](#).

Ejercicio: Añadiendo un dialogo de selección para cambiar la hora.

Un cuadro de diálogo es un tipo de ventana emergente que solicita al usuario de la aplicación algún tipo de información, antes de realizar algún proceso. Este tipo de ventanas no suele ocupar la totalidad de la pantalla. En la aplicación Mis Lugares hemos utilizado diálogos en dos ocasiones: para indicar el id a mostrar y para confirmar el borrado de un lugar. En este ejercicio aprenderemos a hacer un dialogo más complejo, que permite modificar la hora y los minutos.



1. Abre el *layout vista_lugar.xml* y localiza el <imageView> que indica la hora asociada al lugar. Añade el atributo marcado:

```
<ImageView
    android:id="@+id/icono_hora"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:contentDescription="logo de la hora"
    android:src="@android:drawable/ic_menu_recent_history" />
```

- 2.** Abre la clase VistaLugarFragment y añade en el método onActivityCreated() el siguiente código. Si no has realizado el ejercicio con *fragments* añadelo en la clase VistaLugar dentro de onCreate(), pero serán necesarias ligeras modificaciones en los siguientes puntos.

```
vista.findViewById(R.id.icono_hora).setOnClickListener(
    new OnClickListener() {
        public void onClick(View view) { usoLugar.cambiarHora(pos); } });
```

NOTA: Selecciona el paquete android.view.OnClickListener. Repite la operación, pero cambiando R.id.icono_hora por R.id.hora.

- 3.** Como acabas de ver vamos a crear un nuevo caso de uso para cambiar la hora de un lugar. La clase CasosUsoLugar empieza a ser demasiado grande. Podría ser interesante dividirla en tres partes: Operaciones de tipo CRUD (altas, bajas, modificaciones...), fotografías y de fecha y hora. De momento vamos a añadir las operaciones de fecha y hora en la clase CasosUsoLugarFecha:

```
public class CasosUsoLugarFecha extends CasosUsoLugar {

    public CasosUsoLugarFecha(FragmentActivity actividad, Fragment fragment,
                               LugaresBD lugares, AdaptadorLugaresBD adaptador) {
        super(actividad, lugares, adaptador);
    }
}
```

Vamos a heredar de CasoUsoLugar al necesitar funciones definidas en esta clase (en concreto actualizaPosLugar(pos, lugar))

- 4. En Kotlin** aparecerán errores dado que por defecto las clases y las propiedades son cerradas. Si utilizas el desplegable de la bobilla podrás corregir rápidamente estos errores. El resultado final ha de ser:

En Java para poder utilizar propiedades o métodos de la clase padre, estos no pueden tener el modificador private. Cambia los que vayas a utilizar a protected.

```
private protected FragmentActivity actividad;
private protected Fragment fragment;
private protected AdaptadorLugaresBD adaptador;
private protected void actualizaPosLugar(int pos, Lugar lugar) {
```

- 5.** Añade en la nueva clase:

```
int pos = -1;
Lugar lugar;

public void cambiarHora(int pos) {
    lugar = adaptador.lugarPosicion(pos);
    this.pos = pos;
    DialogoSelectorHora dialogo = new DialogoSelectorHora();
```

```

        dialogo.setOnTimeSetListener(this);
        Bundle args = new Bundle();
        args.putLong("fecha", lugar.getFecha());
        dialogo.setArguments(args);
        dialogo.show(actividad.getSupportFragmentManager(), "selectorHora");
    }
}

```

Este método se ejecutará cuando se pulse sobre el ícono de hora. Su objetivo es mostrar un cuadro de diálogo para que el usuario pueda modificar la hora asociada al lugar. Los parámetros son: el pos del lugar a modificar y el TextView donde escribiremos la nueva hora. Comenzamos tres variables que usaremos para recordar la información tras volver del diálogo. Continuamos creando un nuevo diálogo y luego le asignamos el escuchador a nuestra propia clase. De esta forma, cuando el usuario cambie la hora se llamará a un método de nuestra clase. Este método lo crearemos en uno de los puntos siguientes. A este diálogo le pasamos como argumento la fecha del lugar en un long. Finalmente, mostramos el diálogo llamando al método show(). Este método utiliza dos parámetros: el manejador de fragments y una etiqueta que identificará el cuadro de diálogo.

6. Crea la siguiente clase

```

public class DialogoSelectorHora extends DialogFragment {

    private OnTimeSetListener escuchador;

    public void setOnTimeSetListener(OnTimeSetListener escuchador) {
        this.escuchador = escuchador;
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        Calendar calendario = Calendar.getInstance();
        Bundle args = this getArguments();
        if (args != null) {
            long fecha = args.getLong("fecha");
            calendario.setTimeInMillis(fecha);
        }
        int hora = calendario.get(Calendar.HOUR_OF_DAY);
        int minuto = calendario.get(Calendar.MINUTE);
        return new TimePickerDialog(getActivity(), escuchador, hora,
                minuto, DateFormat.is24HourFormat(getActivity()));
    }
}

```

Pulsa **Alt-Intro** para añadir los imports automáticamente. Algunas clases se encuentran en varios paquetes, por lo que te preguntará. Utiliza los que se muestran marcados:

java.text.DateFormat	android.app.DialogFragment
android.text.format.DateFormat	android.support.v4.app.DialogFragment

Esta clase extiende DialogFragment, que define un fragment que muestra una ventana de diálogo flotante sobre la actividad. El control del cuadro de diálogo debe hacerse siempre a través de los métodos del API, nunca directamente.

Para definir un nuevo DialogFragment se puede sobreescibir onCreateView() para indicar el contenido del diálogo. Alternativamente, se puede sobreescibir onCreateDialog() para crear un diálogo totalmente personalizado, como hacemos en este ejercicio. En este método hay que devolver un objeto Dialog que se mostrará.

Creamos un objeto Calendar y si nos han pasado una fecha se la asignamos. En caso contrario, la fecha corresponderá con la actual. Luego extraemos la hora y los minutos del calendario.

Finalmente creamos un nuevo dialogo de la clase TimePickerDialog. Se trata de un tipo de dialogo definido en el sistema que nos permite seleccionar horas y minutos. En su constructor indicamos cuatro parámetros: el contexto, un escuchador al que llamará cuando se seleccione la hora, la hora y minutos que se mostrarán al inicio y un valor booleano que indica si trabajamos con formato de 24 horas o de 12. En el código se usa el valor definido en nuestro contexto.

7. Haz que CasosUsoLugarFecha implemente la siguiente interfaz:

```
public class CasosUsoLugarFecha extends CasosUsoLugar
    implements TimePickerDialog.OnTimeSetListener {
```

8. Añade la siguiente función:

```
@Override public void onTimeSet(TimePicker vista, int hora, int minuto) {
    Calendar calendario = Calendar.getInstance();
    calendario.setTimeInMillis(lugar.getFecha());
    calendario.set(Calendar.HOUR_OF_DAY, hora);
    calendario.set(Calendar.MINUTE, minuto);
    lugar.setFecha(calendario.getTimeInMillis());
    actualizaPosLugar(pos, lugar);
    TextView textView = actividad.findViewById(R.id.hora);
    textView.setText.DateFormat.getTimeInstance().format(
        new Date(lugar.getFecha())));
}
```

En el punto anterior hemos indicado que nuestra clase actuaría como escuchador, cuando se seleccionara una hora en el cuadro de diálogo. Como consecuencia este método será llamado. Se nos pasan tres parámetros. En este caso nos interesa la hora y los minutos seleccionados. Para cambiar esta información en la fecha asociada al lugar, comenzamos creando un objeto Calendar y lo inicializamos con la fecha que tiene el lugar. Luego, le modificamos la hora y los minutos según los parámetros que nos han indicado. Hay que aclarar que el resto de la fecha, como el día o el mes, no van a modificarse. La nueva fecha es introducida en el objeto lugar y a continuación actualizamos la base de datos.

Para modificar el TextView de la hora, comenzamos creando un formato de fecha, donde se visualizará la hora y los minutos separado por dos puntos. Para convertir la fecha correctamente hay que conocer la zona horaria definida en el sistema. Esto se consigue con java.util.Locale.getDefault(). Finalmente usamos este formato sobre un objeto Date para cambiara el contenido del TextView.

9. En VistaLugarActivity utiliza la clase CasosUsoLugarFecha en lugar de CasosUsoLugar para la variable usoLugar.

9. Ejecuta la aplicación y verifica el resultado.

10.

Práctica: Añadiendo un dialogo de selección para cambiar la fecha.

Podrías crear un cuadro de dialogo para modificar la fecha asociada al lugar (día, mes y año). Has de realizar los mismos pasos que en el ejercicio anterior, pero ahora se basará en el diálogo siguiente.



En este caso tendrás que usar un diálogo de la clase DatePickerDialog:

```
DatePickerDialog(actividad , escuchador, año, mes, dia);
```

El escuchador ha de implementar el interface OnDateSetListener y este interface define el siguiente método:

```
@Override
```

```
public void onDateSet(DatePicker view, int anyo, int mes, int dia) {...}
```

Finalmente utiliza el siguiente formato para representarlo:

```
DateFormat formato = DateFormat.getDateInstance();
```

En este caso no se define un formato concreto como en el ejercicio anterior, si no que se selecciona el definido en el sistema para representar una fecha. De esta forma, el formato será el que ha configurado el usuario en el dispositivo.

Solución:

Clase VistaLugarFragment dentro de onActivityCreated():

```
vista.findViewById(R.id.icono_fecha).setOnClickListener(
    new OnClickListener() {
        public void onClick(View view) { usoLugar.cambiarFecha(pos); } });
```

En la clase DialogoSelectorFecha añade la interfaz y las dos funciones indicadas.

```
public class CasosUsoLugarFecha extends CasosUsoLugar implements
    TimePickerDialog.OnTimeSetListener , DatePickerDialog.OnDateSetListener {

    public void cambiarFecha(int pos) {
        lugar = adaptador.lugarPosicion(pos);
        this.pos = pos;
        DialogoSelectorFecha dialogo = new DialogoSelectorFecha();
        dialogo.setOnDateSetListener(this);
        Bundle args = new Bundle();
        args.putLong("fecha", lugar.getFecha());
        dialogo.setArguments(args);
        dialogo.show(actividad.getSupportFragmentManager(),"selectorFecha");
    }

    @Override
    public void onDateSet(DatePicker view, int año, int mes, int dia) {
        Calendar calendario = Calendar.getInstance();
        calendario.setTimeInMillis(lugar.getFecha());
        calendario.set(Calendar.YEAR, año);
        calendario.set(Calendar.MONTH, mes);
        calendario.set(Calendar.DAY_OF_MONTH, dia);
        lugar.setFecha(calendario.getTimeInMillis());
        actualizaPosLugar(pos, lugar);
        TextView textView = actividad.findViewById(R.id.fecha);
        textView.setText.DateFormat.getDateInstance().format(
            new Date(lugar.getFecha())));
    }
}
```

Clase DialogoSelectorFecha:

```
public class DialogoSelectorFecha extends DialogFragment {

    private OnDateSetListener escuchador;

    public void setOnDateSetListener(OnDateSetListener escuchador) {
        this.escuchador = escuchador;
    }
    @Override public Dialog onCreateDialog(Bundle savedInstanceState) {
        Calendar calendario = Calendar.getInstance();
        Bundle args = this getArguments();
        if (args != null) {
            long fecha = args.getLong("fecha");
            calendario.setTimeInMillis(fecha);
        }
        int año = calendario.get(Calendar.YEAR);
        int mes = calendario.get(Calendar.MONTH);
        int dia = calendario.get(Calendar.DAY_OF_MONTH);
        return new DatePickerDialog(getActivity(),escuchador,año,mes,dia);
    }
}
```

Práctica: Separando casos de uso en varias clases

Acabamos de crear la clase CasosUsoLugarFecha que es una ampliación de uso de CasosUsoLugar. Se trata de una estrategia adecuada en muchas ocasiones, pero también existe otra alternativa que nos permitiría dividir los casos de uso en diferentes clases. En esta práctica se describe cómo hacerlo. Crea una clase abstracta con nombre CasosUsoLugarBase. Que solo contenga la función actualizaPosLugar(). Crea tres clases que extiendan de esta con nombres: CasosUsoLugarOperacion, CasosUsoLugarFoto, CasosUsoLugarFecha. Distribuye las funciones entre las tres clases. Desde las actividades o fragmentos que utilicen estos casos de uso tendrás que crear variables para cada una de las clases necesarias. Al hacer este cambio estamos complicando el código, necesitamos más clase y variables. Pero tenemos algunas ventajas: La responsabilidad de cada clase es más pequeña, lo que las hace más fáciles de entender y mantener. El código es más reutilizable. Por ejemplo, cuando necesites añadir fotografías en una nueva aplicación, será fácil localizar el código necesario y adaptarlo con el mínimo número de cambios.

[1] <http://developer.android.com/reference/java/util/Date.html>

[2] <http://developer.android.com/reference/java/text/DateFormat.html>

[3] <http://developer.android.com/reference/java/text/SimpleDateFormat.html>

[4] <http://developer.android.com/reference/java/util/Calendar.html>