

# Programació avançada i de comunicacions

Eduard García Sacristán, Joan Climent Balaguer

Programació multimèdia i dispositius mòbils



# Índex

<b>Introducció</b>	<b>5</b>
<b>Resultats d'aprenentatge</b>	<b>7</b>
<b>1 Programació avançada</b>	<b>9</b>
1.1 Imports a l'Android Studio	9
1.2 Persistència	10
1.3 Treballant amb bases de dades	11
1.3.1 Classe DBInterface	12
1.3.2 Fent servir la classe "DBInterface"	17
1.4 Proveïdors de continguts	26
1.4.1 Accedint al proveïdor de continguts	27
1.4.2 Inserint dades	32
1.5 Publicant aplicacions	32
1.5.1 Preparar l'aplicació	33
1.5.2 Distribuir aplicacions	36
<b>2 Programació de comunicacions</b>	<b>39</b>
2.1 Comunicacions en Android	39
2.2 Mostrar pàgines web amb un 'widget'	41
2.3 Model dels fils	43
2.4 Connexions HTTP	46
2.4.1 Comunicacions segures amb HTTPS	48
2.4.2 Pujant dades a un servidor	48
2.5 Missatgeria	48
2.5.1 Enviament d'SMS	49
2.5.2 Rebre SMS	51
2.5.3 Enviant un SMS mitjançant un 'Intent'	54
2.5.4 Seguretat	55
2.5.5 Enviant un MMS mitjançant 'Intent'	56
2.6 Analitzant codi XML	57



## Introducció

La programació d'aplicacions per dispositius mòbils, en general, consta de més funcionalitats que unes simples pantalles que s'envien entre elles informació introduïda per l'usuari. Una vegada tenim clar l'esquema de l'aplicació i la transició entre pantalles, és el moment d'anar més enllà i atribuir-li un seguit de particularitats específiques.

Android ens proporciona una API i unes llibreries de classes que donen al programador l'oportunitat de crear aplicacions completes. A banda de la programació de la interfície gràfica de l'aplicació (amb totes les seves complexitats), existeixen altres elements que poden enriquir la vostra aplicació, com ara la persistència de dades i la comunicació de la vostra aplicació amb Internet.

En l'apartat “Programació avançada” veureu tècniques avançades de programació en Android. S'hi tractaran diferents formes d'obtenir persistència de dades, mitjançant bases de dades i proveïdors de continguts. També veureu la manera de preparar les aplicacions per a la seva publicació i distribució.

En l'apartat “Programació de comunicacions” veureu com treballar amb les diferents llibreries de comunicacions que ofereix Android per a la programació de les seves aplicacions. Atès que Android és un sistema dissenyat per a dispositius mòbils, la connexió contínua és un dels seus aspectes fonamentals. Vegeu la comunicació web i els serveis de missatgeria instantània i multimèdia.

Per seguir els continguts d'aquest mòdul, és convenient anar fent les activitats i els exercicis d'autoavaluació i llegir els annexos (si n'hi ha). Tot i que les unitats formatives tenen un contingut important des del punt de vista conceptual, sempre s'ha procurat donar-los un enfocament pràctic en les activitats proposades.



## Resultats d'aprenentatge

En finalitzar aquesta unitat, l'alumne/a:

1. Desenvolupa aplicacions per a dispositius mòbils analitzant i fent servir les tecnologies i llibreries específiques.
  - Utilitza les classes necessàries per a la connexió i comunicació amb dispositius sense fils.
  - Utilitza les classes necessàries per a l'intercanvi de missatges text i multimèdia.
  - Utilitza les classes necessàries per establir connexions i comunicacions HTTP i HTTPS.
  - Utilitza les classes necessàries per establir connexions amb magatzems de dades garantint la persistència.
  - Realitza proves d'interacció usuari-aplicació per optimitzar les aplicacions desenvolupades a partir d'emuladors.
  - Empaqueta i desplega les aplicacions desenvolupades en dispositius mòbils reals.
  - Documenta els processos necessaris per al desenvolupament de les aplicacions.





## 1. Programació avançada

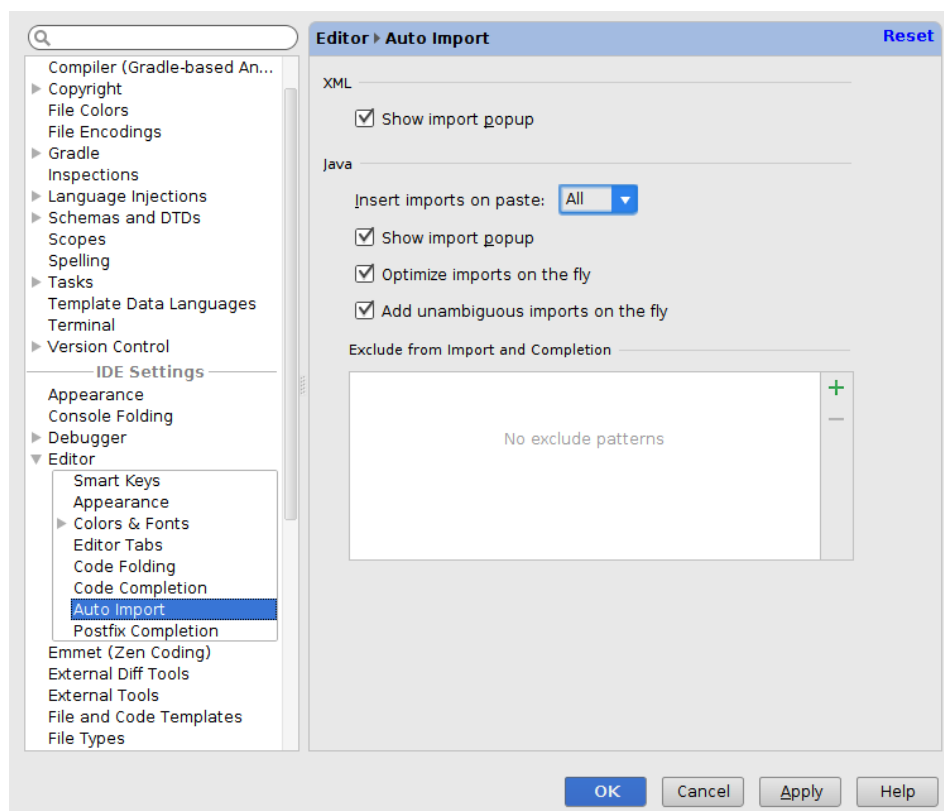
Quan hàgiu creat l'esquema d'una senzilla aplicació, és el moment d'afegir funcionalitats avançades, com una interfície d'usuari avançada, l'accés a les dades de l'aplicació (mitjançant bases de dades o proveïdors de continguts) o la persistència de les dades de la mateixa. Així mateix, quan penseu que teniu l'aplicació suficientment desenvolupada, és el moment de publicar l'aplicació en qualsevol de les diferents formes que teniu per fer-ho.

### 1.1 Imports a l'Android Studio

Per facilitar la programació amb l'Android Studio modificarem les preferències per automatitzar els *imports* a mesura que anem escrivint o que enganxem codi, així obtindrem una major agilitat a l'hora d'escriure codi.

Accedirem a les preferències des de *File/Settings* i al submenú *Editor/Auto Import* haurem de marcar els *checkboxes* *Optimize imports on the fly* i *Add unambiguous imports on the fly*. A més, podem modificar el valor d'*Insert imports on paste* d'*Ask* a *All* (vegeu la figura 1.1).

FIGURA 1.1



Amb aquestes modificacions l'Android Studio farà tots els *imports* automàticament, és per això que sempre haurem de comprovar que l'import sigui el correcte per evitar comportaments no desitjats.

## 1.2 Persistència

És probable que vulgueu que la vostra aplicació pugui desar algunes dades entre les diferents execucions de l'aplicació. Per exemple, podeu voler desar les preferències de l'aplicació per tal que la propera vegada que l'executeu tingui la mateixa aparença que la darrera vegada que es va executar. Existeixen diferents formes d'obtenir persistència en les dades de l'aplicació en diferents execucions:

- Un mecanisme lleuger anomenat **preferències compartides** (*shared preferences*) per desar petites quantitats de dades.
- El sistema de fitxers tradicional.
- Una base de dades relacional SQLite.

Per desar poca informació, la millor forma de fer-ho és amb les preferències compartides. Android incorpora l'objecte `SharedPreferences`, que serveix per desar i llegir dades persistents en la forma clau-valor de dades primitives. Podeu fer servir `SharedPreferences` per desar qualsevol tipus de dades primitives: *boolean*, *float*, *int*, *long* i *string*. Aquestes dades es mantindran entre sessions, encara que la vostra aplicació s'hagi tancat. A més, seran desades automàticament en un fitxer XML.

Per obtenir un objecte `SharedPreferences` a la vostra aplicació podeu fer servir el mètode `getSharedPreferences()` amb dos arguments: el nom del fitxer de preferències i el mode d'operació. Per exemple:

```
1 private SharedPreferences prefs;  
2 //Obtenir l'objecte SharedPreferences  
3 prefs = getSharedPreferences("FitxerPreferences", MODE_PRIVATE);
```

En el cas que vulgueu fer servir un únic fitxer de preferències, podeu cridar el mètode `getPreferences()`, on no cal especificar el nom del fitxer.

Per escriure valors al fitxer de preferències:

1. Crideu el mètode `edit()` per obtenir un objecte `SharedPreferences.Editor`.
2. Afegiu valors amb els mètodes que us permeten escriure valors primitius, com `putBoolean()` o `putString()`. Aquests mètodes tenen dos arguments. El primer és un *string* que defineix la clau, i el segon és el valor que es vol desar.

### 3. Confirmeu els valors amb `commit()`.

Per exemple:

```
1 SharedPreferences.Editor editor = prefs.edit();
2
3 editor.putInt("Edat", 23);
4 editor.putString("NomUsuari", "Fidel");
```

Per llegir els valors de preferències podeu fer servir els mètodes anàlegs `getBoolean()`, `getString()` o `getInt()` de la classe `SharedPreferences`. Aquests mètodes tenen dos arguments: el primer és el nom de la clau que esteu cercant al fitxer de preferències. El segon és el valor per defecte que es farà servir en cas que no es trobi la clau en el fitxer de preferències. Per exemple:

```
1 //Carregarlespreferències
2 SharedPreferences prefs = getSharedPreferences("FitxerConfiguracio",
3     MODE_PRIVATE);
4
5 int Edat = prefs.getInt("Edat", 18);
6 String nom = prefs.getString("NomUsari", "Usuari");
```

El lloc més adequat per carregar i desar les preferències de l'aplicació serien els mètodes `onCreate()` i `onStop()` (el moment en què l'aplicació s'engega i el moment en què es tanca).

## 1.3 Treballant amb bases de dades

Android proporciona un sistema de bases de dades relacionals basat en `SQLite` que podeu fer servir a les vostres aplicacions. Quan la quantitat de dades a desar és important, o bé es volen fer cerques sobre les dades, o la informació està relacionada entre si, és més adequat tenir-la estructurada en forma d'una base de dades. Per exemple, podríeu tenir una base de dades amb informació de diferents fabricants relacionada amb els diferents productes que aquests produeixen. Fent servir bases de dades, podeu assegurar la integritat de les dades especificant relacions entre els diferents conjunts de dades.

Android fa servir el sistema de bases de dades `SQLite`. La base de dades que podeu crear per a la vostra aplicació únicament estarà disponible per a la pròpia aplicació. La resta d'aplicacions del dispositiu no hi podran accedir, per tant no podeu fer servir la base de dades per compartir dades entre aplicacions.

Treballar amb bases de dades a Android pot ser complicat. Per aquest motiu, heu de crear una classe que servirà per encapsular l'accés a les bases de dades i, així, simplificar el codi de la vostra aplicació (que tindrà un accés a les dades transparent a la seva implementació, a través d'aquesta classe).

---

Diem que una aplicació té un accés transparent a les dades quan podem accedir a la informació a través de mètodes sense haver de conèixer la seva implementació. Aquesta manera de treballar ens permet modificar l'estructura interna de la informació sense que les aplicacions que la fan servir hagin de modificar el seu codi per funcionar correctament.

---

Podeu descarregar el codi corresponent a aquesta activitat en l'annex anomenat "Bases de dades" de la secció "Annexos".

### 1.3.1 Classe DBInterface

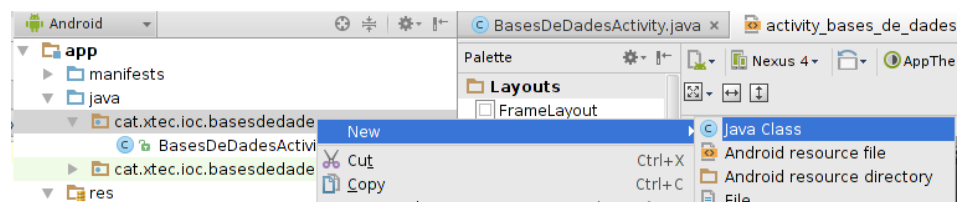
Creeu un nou projecte amb les següents dades:

- **Application name:** BasesDeDades
- **Company domain:** cat.xtec.ioc
- **Blank Activity:** BasesDeDadesActivity

Deixeu la resta d'opcions amb els valors per defecte.

Ara creareu una classe que us servirà d'interfície amb l'accés a les bases de dades de l'aplicació. Aquesta classe tindrà el nom DBInterface. Creeu una aplicació, i dins d'aquesta, creeu una nova classe fent clic amb el botó dret dins del paquet del vostre projecte i seleccionant *New/Class*, com es pot veure a la figura 1.2.

**FIGURA 1.2.** Creant una nova classe



Aquesta classe crearà, obrirà, farà servir i tancarà una base de dades SQLite. Creareu una base de dades anomenada BDClints que contindrà una única taula, *contactes*. Aquesta taula tindrà únicament tres camps: *\_id*, *nom* i *email*, tal com es pot veure a la taula 1.1.

**TAULA 1.1.** Taula de contactes

<i>_id</i>	<i>nom</i>	<i>email</i>
1	John	jcoltrane@atlantic.com
2	Miles	mdavis@bluenote.com

En primer lloc, cal que definiu una sèrie de constants de text que serviran per establir alguns identificadors i camps (i així no caldrà haver-los de repetir per tot el codi, amb el perill d'equivocar-se en algun moment).

```

1 package ioc.xtec.cat.basesdedades;
2
3 import android.content.ContentValues;
4 import android.content.Context;
5 import android.database.Cursor;
6 import android.database.SQLException;
7 import android.database.sqlite.SQLiteDatabase;
8 import android.database.sqlite.SQLiteOpenHelper;
9 import android.util.Log;
10
11
12 public class DBInterface {
13     //Constants

```

```

14 public static final String CLAU_ID = "_id";
15 public static final String CLAU_NOM = "nom";
16 public static final String CLAU_EMAIL = "email";
17
18 public static final String TAG = "DBInterface";
19
20 public static final String BD_NOM = "BDClients";
21 public static final String BD_TAULA = "contactes";
22 public static final int VERSIO = 1;
23
24 public static final String BD_CREATE =
25     "create table " + BD_TAULA + "( " + CLAU_ID + " integer primary key
26         autoincrement, " +
27         CLAU_NOM + " text not null, " + CLAU_EMAIL + " text not
28         null);";
29
30 private final Context context;
31 private AjudaBD ajuda;
32 private SQLiteDatabase bd;
33 }

```

</newcontent>

Concretament, la constant BD\_CREATE conté la cadena que es farà servir per crear la taula *contactes* dins de la base de dades.

La classe AjudaBD, la creareu més tard. El constructor de la nostra classe, l'únic que fa és crear un objecte AjudaBD i guardar en una variable el context en què s'està executant la classe:

```

1 public DBInterface(Context con)
2 {
3     this.context = con;
4     ajuda = new AjudaBD(context);
5 }

```

Context és una classe implementada pel sistema Android que dóna accés a recursos i classes específics de l'aplicació.

A Android existeix la classe SQLiteOpenHelper, que és una classe que serveix d'ajuda per gestionar la creació de bases de dades i gestió de versions. Creareu la classe AjudaBD que hereta d'aquesta:

```

1 private static class AjudaBD extends SQLiteOpenHelper {
2     AjudaBD(Context con) {
3         super(con, BD_NOM, null, VERSIO);
4     }
5
6     @Override
7     public void onCreate(SQLiteDatabase db) {
8         try {
9             db.execSQL(BD_CREATE);
10        } catch (SQLException e) {
11            e.printStackTrace();
12        }
13    }
14
15    @Override
16    public void onUpgrade(SQLiteDatabase db, int VersioAntiga, int
        VersioNova) {
17        Log.w(TAG, "Actualitzant Base de dades de la versió" + VersioAntiga
            + " a " + VersioNova + ". Destruirà totes les dades");
18        db.execSQL("DROP TABLE IF EXISTS " + BD_TAULA);
19    }

```

```
20         onCreate(db);  
21     }  
22 }
```

El constructor d'AjudaDB crida el constructor d'SQLiteOpenHelper, el qual crea un objecte d'ajuda per crear, obrir i gestionar la base de dades. Mireu la documentació d'SQLiteOpenHelper per obtenir ajuda sobre els seus mètodes. Aquesta classe s'ocupa d'obrir la base de dades si aquesta existeix o crear-la en cas contrari, i actualitzar-la si és necessari.

El mètode onCreate() crea una nova base de dades. El mètode onUpgrade() és cridat quan s'ha d'actualitzar la base de dades. El que fa és eliminar-la (fer un *drop* de la taula) i tornar-la a crear.

De tornada a la classe DBInterface, el següent pas és definir els diferents mètodes per obrir i tancar la base de dades.

```
1 //Obre la BD  
2  
3 public DBInterface obre() throws SQLException {  
4     bd = ajuda.getWritableDatabase();  
5     return this;  
6 }  
7  
8 //Tanca la BD  
9  
10 public void tanca() {  
11     ajuda.close();  
12 }
```

El mètode getWritableDatabase() crea i/o obre una base de dades. La primera vegada que es crida s'obre la base de dades i es crida onCreate(). Aquí és on es crea la base de dades, cridant execSQL() amb la cadena de creació de la base de dades. Una vegada creada, la base de dades queda en *cache*, i per tant es pot cridar aquest mètode per obrir-la.

A continuació definireu els mètodes per modificar la base de dades. Per inserir un contacte fareu servir el mètode insert(String table, String nullColumnHack, ContentValues values). Els arguments són aquests:

- String table: taula on es vol inserir un element.
- String nullColumnHack: un argument opcional que deixarem a *null*. Podeu consultar la *Guia del desenvolupador d'Android* per consultar el seu ús.
- ContentValues values: un objecte de la classe ContentValues que serveix per emmagatzemar valors que poden ser processats per un ContentResolver.

Aquest mètode retorna l'ID de la fila que s'ha inserit o un -1 si hi ha hagut una errada. El nostre mètode per inserir un contacte crearà un ContentValues amb els valors de la fila a inserir i farà la crida a insert().

```
1 //Insereix un contacte
2
3 public long insereixContacte(String nom, String email
4 {
5     ContentValues initialValues = new ContentValues();
6     initialValues.put(CLAU_NOM, nom);
7     initialValues.put(CLAU_EMAIL, email);
8     return bd.insert(BD_TAULA, null, initialValues);
9
10 }
```

Per esborrar un element de la taula fareu servir el mètode `delete` (`String table`, `String whereClause`, `String[] whereArgs`). El significat dels arguments és el següent:

- `String table`: taula de la qual s'esborrarà el registre.
- `String whereClause`: la clàusula `WHERE` que s'aplicarà per esborrar de la base de dades. Si se li passa `null`, esborrarà totes les files de la taula.
- `String[] whereArgs`: arguments de la clàusula `WHERE`. Aquest mètode retorna la quantitat de files afectades per la clàusula `WHERE`.

```
1 //Esborra un contacte
2
3 public boolean esborraContacte(long IDFila)
4 {
5     return bd.delete(BD_TAULA, CLAU_ID + " = " + IDFila, null) > 0;
6 }
```

El vostre mètode retornarà un valor booleà que indica si s'ha esborrat algun element o no.

Per retornar un contacte fareu servir el mètode `query()`, que té els següents arguments:

- `boolean distinct`: serà `true` si voleu que cada fila sigui única, o `false` en cas contrari.
- `String table`: defineix la taula respecte a la qual voleu executar la sentència de *query*.
- `String[] columns`: admet una llista de les columnes de la taula que retornarà el mètode.
- `String selection`: estableix un filtre que defineix quines files retornar, amb el format de clàusula d'SQL, `WHERE` (sense incloure la paraula `WHERE` a l'*string*).
- `String[] selectionArgs`: permet afegir els arguments de la selecció, si no els heu introduït directament a la cadena.
- `String groupBy`: estableix un filtre que defineix com s'agrupen les files. Té el mateix format que la clàusula d'SQL: `GROUP BY` (sense incloure les paraules `GROUP BY`). Si se li passa un `null`, les files que es retornin no estaran agrupades.

- **String having:** estableix un filtre que declara quins grups de files incloure al cursor. Té el mateix format que la clàusula d'SQL: **HAVING** (sense incloure la paraula **HAVING**). Si se li passa un *null*, s'inclouran tots els grups.
- **String orderBy:** indica com ordenar les files. Té el mateix format que la clàusula d'SQL: **ORDER BY** (sense incloure les paraules **ORDER BY**). Si se li passa un *null* retorna les files amb l'ordre per defecte.
- **String limit:** especifica el límit de files retornades pel *query*, amb el format de la clàusula d'SQL: **LIMIT**. Si se li passa un *null*, no existeix límit.

Aquest mètode retorna un objecte de la classe **Cursor**, que proporciona accés de lectura i escriptura al resultat retornat per una consulta (un *query*) a la base de dades.

```
1 //Retorna un contacte
2
3 public Cursor obtenirContacte(long IDFil) throws SQLException {
4     Cursor mCursor = bd.query(true, BD_TAULA, new String[] {CLAU_ID, CLAU_NOM,
5         CLAU_EMAIL}, CLAU_ID + " = " + IDFil, null, null, null, null, null);
6
7     if(mCursor != null) {
8         mCursor.moveToFirst();
9     }
10
11     return mCursor;
12 }
```

Per obtenir tots els contactes, feu servir una altra versió de *query* que no inclogui el primer booleà.

```
1 //Retorna tots els contactes
2
3 public Cursor obtenirTotsElsContactes()
4 {
5     return bd.query(BD_TAULA, new String[] {CLAU_ID, CLAU_NOM, CLAU_EMAIL}, null,
6         null, null, null, null);
7 }
```

Fixeu-vos que Android fa servir un objecte de la classe **Cursor** per valor de retorn de les consultes a la base de dades (els *queries*). Penseu en el **Cursor** com un apuntador al conjunt de resultats obtinguts de la consulta a la base de dades. L'ús d'un **Cursor** permet a Android gestionar d'una forma més eficient les files i les columnes.

Finalment, per actualitzar un registre de la taula fareu servir el mètode **update()** amb els següents arguments:

- **String table:** estableix la taula a actualitzar.
- **ContentValues values:** introdueix un objecte de la classe **ContentValues** amb el valor de les columnes a actualitzar.
- **String whereClause:** inclou la clàusula **WHERE** opcional que s'ha d'aplicar quan es fa l'actualització. Si se li passa *null* actualitzarà totes les files.



- `String[] whereArgs`: estableix els arguments del `WHERE`.

El codi resultant és el següent:

```

1 //Modifica un contacte
2
3 public boolean actualitzarContacte(long IDFila, String nom, String email) {
4     ContentValues args = new ContentValues();
5     args.put(CLAU_NOM, nom);
6     args.put(CLAU_EMAIL, email);
7     return bd.update(BD_TAULA, args, CLAU_ID + " = " + IDFila, null) > 0;
8 }

```

### 1.3.2 Fent servir la classe "DBInterface"

Quan hàgiu creat la classe que us servirà d'ajuda per treballar amb les bases de dades, és el moment de fer-la servir. El següent exemple mostra una aplicació que fa servir la classe d'interfície de base de dades. És molt senzilla, però serveix per il·lustrar la forma d'ús d'aquesta classe.

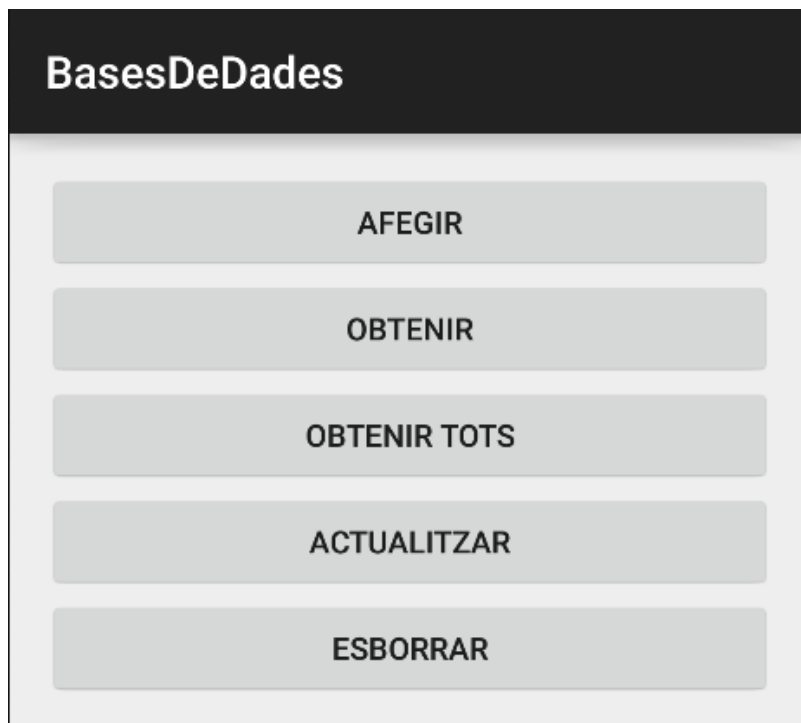
L'activitat principal té un *layout* amb cinc botons, un per a cada opció de l'aplicació, tal com es pot veure a la figura 1.3.

---

Alguns dels *layouts* s'han omès, els podreu trobar al codi font del projecte.

---

FIGURA 1.3. Layout de l'aplicació



El codi que correspon a aquest *layout* és el següent:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"

```

```

5      android:layout_height="match_parent"
6      android:orientation="vertical"
7      android:paddingBottom="@dimen/activity_vertical_margin"
8      android:paddingLeft="@dimen/activity_horizontal_margin"
9      android:paddingRight="@dimen/activity_horizontal_margin"
10     android:paddingTop="@dimen/activity_vertical_margin"
11     tools:context=".MainActivity">
12
13     <Button
14         android:id="@+id/btnAfegir"
15         android:layout_width="fill_parent"
16         android:layout_height="wrap_content"
17         android:text="Afegir" />
18
19     <Button
20         android:id="@+id/btnObtenir"
21         android:layout_width="fill_parent"
22         android:layout_height="wrap_content"
23         android:text="Obtenir" />
24
25     <Button
26         android:id="@+id/btnObtenirTots"
27         android:layout_width="fill_parent"
28         android:layout_height="wrap_content"
29         android:text="Obtenir Tots" />
30
31     <Button
32         android:id="@+id/btnActualitzar"
33         android:layout_width="fill_parent"
34         android:layout_height="wrap_content"
35         android:text="Actualitzar" />
36
37     <Button
38         android:id="@+id/btnEsborrar"
39         android:layout_width="fill_parent"
40         android:layout_height="wrap_content"
41         android:text="Esborrar" />
42
43 </LinearLayout>

```

L'aplicació activarà una activitat per a cadascun dels botons, equivalent a les diferents “pantalles” de l'aplicació. Aquestes diferents activitats han d'estar definides a l'AndroidManifest.xml. Recordeu que heu de definir al fitxer de *manifest* els filtres d'*intents* per a cada activitat, per tal que les diferents activitats s'activin quan es llancin els *intents* adequats per a cada una d'elles. Aquest és el document de *manifest* de l'aplicació:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="ioc.xtec.cat.basesdedades" >
4
5     <application
6         android:allowBackup="true"
7         android:icon="@drawable/ic_launcher"
8         android:label="@string/app_name"
9         android:theme="@style/AppTheme" >
10         <activity
11             android:name=".BaseDeDadesActivity"
12             android:label="@string/app_name" >
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19         <activity
20             android:name=".Afegir"

```

```

21         android:label="@string/title_activity_afegir" >
22         <intent-filter>
23             <action android:name="cat.xtec.ioc.AFEGIR" />
24
25             <category android:name="android.intent.category.DEFAULT" />
26         </intent-filter>
27     </activity>
28     <activity
29         android:name=".Obtenir"
30         android:label="@string/title_activity_obtenir" >
31         <intent-filter>
32             <action android:name="cat.xtec.ioc.OBTENIR" />
33
34             <category android:name="android.intent.category.DEFAULT" />
35         </intent-filter>
36     </activity>
37     <activity
38         android:name=".Esborrar"
39         android:label="@string/title_activity_esborrar" >
40         <intent-filter>
41             <action android:name="cat.xtec.ioc.ESBORRAR" />
42
43             <category android:name="android.intent.category.DEFAULT" />
44         </intent-filter>
45     </activity>
46     <activity
47         android:name=".Actualitzar"
48         android:label="@string/title_activity_actualitzar" >
49         <intent-filter>
50             <action android:name="cat.xtec.ioc.ACTUALITZAR" />
51
52             <category android:name="android.intent.category.DEFAULT" />
53         </intent-filter>
54     </activity>
55 </application>
56
57 </manifest>

```

L'activitat principal implementa la interfície `OnClickListener` per gestionar les accions que es deriven quan l'usuari prem els botons. Tot seguit teniu el codi parcial que correspon a la definició de la classe i de les variables que farem servir a l'activitat:

```

1  public class BasesDeDadesActivity extends Activity implements OnClickListener {
2
3      Button btnAfegir, btnObtenir, btnObtenirTots, btnActualitzar, btnEsborrar;
4      DBInterface bd;

```

El mètode `onCreate()` senzillament carrega el *layout*, crea l'objecte d'interfície de la base de dades i crea els *listeners* dels botons.

```

1  public void onCreate(Bundle savedInstanceState) {
2      super.onCreate(savedInstanceState);
3      setContentView(R.layout.main);
4
5      bd = new DBInterface(this);
6
7      //Listeners dels botons
8      btnAfegir = (Button) findViewById(R.id.btnAfegir);
9      btnAfegir.setOnClickListener(this);
10
11      btnObtenirTots = (Button) findViewById(R.id.btnObtenirTots);
12      btnObtenirTots.setOnClickListener(this);
13
14      btnObtenir = (Button) findViewById(R.id.btnObtenir);
15      btnObtenir.setOnClickListener(this);

```

```

16
17     btnEsborrar = (Button) findViewById(R.id.btnEsborrar);
18     btnEsborrar.setOnClickListener(this);
19
20     btnActualitzar = (Button) findViewById(R.id.btnActualitzar);
21     btnActualitzar.setOnClickListener(this);
22
23 }

```

A continuació implementarem cadascuna de les accions que corresponen als botons:

- Afegir
- Obtenir
- Obtenir tots
- Actualitzar
- Esborrar

### Afegir

En primer lloc, cal definir el *listener* del botó:

```

1 //Afegir
2 if (v == btnAfegir) {
3     startActivity(new Intent("cat.xtec.ioc.AFEGIR"));
4 }

```

Això llançarà un *intent* per a l'activitat que permetrà afegir elements a la base de dades. Aquesta activitat té un disseny molt senzill, incorpora tan sols els *widgets* necessaris per afegir un nou element, com es pot veure a la figura 1.4.

FIGURA 1.4. Activitat d'afegir contactes

Aquest és el *listener* del botó *Afegir*:

```

1 @Override
2 public void onClick(View v) {
3     if(v == btnAfegir){

```

```

4      //Obrim la base de dades
5      bd = new DBInterface(this);
6      bd.obre();
7
8      //Inserim el contacte
9      if(bd.insereixContacte(editNom.getText().toString(), editEmail.
10         getText().toString()) != -1) {
11         Toast.makeText(this, "Afegit correctament", Toast.LENGTH_SHORT)
12            .show();
13     } else {
14         Toast.makeText(this, "Error a l'afegir", Toast.LENGTH_SHORT).
15            show();
16     }
17     bd.tanca();
18     finish();
19 }

```

Bàsicament, el que es fa és:

1. Obrir la base de dades.
2. Inserir un element amb les dades que obtingui de les caixes de text. Fixeu-vos com comprova el valor de retorn d'`insereixContacte()` per saber si s'ha pogut inserir el contacte nou correctament o no.
3. Tancar la base de dades.
4. Tancar l'activitat.

## Obtenir

Tot seguit definirem un nou *listener*, aquest cop pel botó d'*Obtenir*.

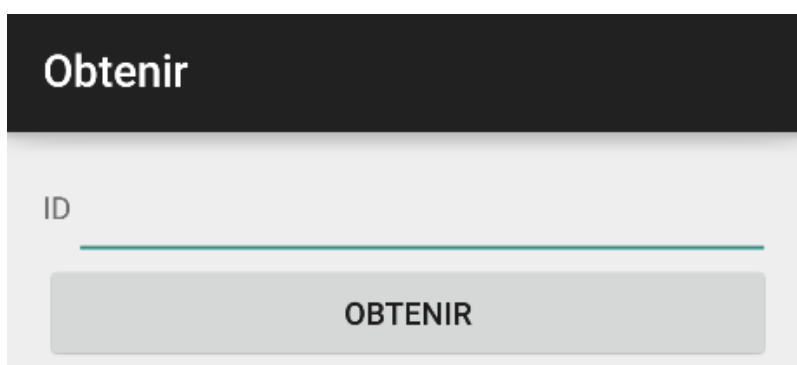
```

1  btnObtenir = (Button) findViewById(R.id.btnObtenir);
2  btnObtenir.setOnClickListener(this);
3
4  ...
5
6  else if (v == btnObtenir) {
7      startActivity(new Intent("cat.xtec.ioc.OBTENIR"));
8  }

```

Això us permetrà llançar una activitat en la que els únics *widgets* seran una caixa de text i el botó per obtenir el contacte, com es pot veure a la figura 1.5.

FIGURA 1.5. Activitat per obtenir un contacte



Al codi de l'activitat l'única cosa que es programa és el *listener* del botó *Obtenir*. El codi és el següent:

```
1  @Override
2      public void onClick(View v) {
3          //Botó obtenir
4          if (v == btnObtenir) {
5              Cursor c;
6
7              //Obrim la base de dades
8              bd = new DBInterface(this);
9              bd.obre();
10
11              //Aquest és l'identificador que està escrit a la caixa de text
12              long id = Long.parseLong(editID.getText().toString());
13
14              //Crida a la BD
15              c = bd.obtenirContacte(id);
16
17              // Comprovem si hi ha hagut algun resultat
18              if (c.getCount() != 0) {
19                  //Mostrem el contacte
20                  Toast.makeText(this, "id: " + c.getString(0) + "\n" + "Nom: " +
21                      c.getString(1) + "\n Email: " + c.getString(2), Toast.
22                      LENGTH_SHORT).show();
23              } else {
24                  Toast.makeText(this, "id inexistent!", Toast.LENGTH_SHORT).show
25                      ();
26              }
27              //Tanquem la BD
28              bd.tanca();
29              //Tanca l'activitat
30              finish();
31          }
32      }
```

El que hem fet és:

1. Obrir la base de dades
2. Obtenir l'identificador que està escrit a la caixa de text
3. Cridar la base de dades
4. Comprovar si hi ha hagut algun resultat
5. Mostrar el contacte
6. Tancar la base de dades
7. Tancar l'activitat

El més normal seria mostrar el contacte a través d'una altra activitat dissenyada per mostrar contactes, però per fer el programa més senzill hem optat per mostrar el contacte amb un missatge per pantalla.

### Obtenir tots

L'opció *Obtenir tots* mostrarà tots els contactes per pantalla, un a un. En una aplicació de veritat el més habitual seria introduir els contactes en un `ListView`

o un `ContentProvider`, però en aquest exemple volem mostrar únicament el funcionament de les bases de dades, per tant qualsevol forma de mostrar els contactes ens serà suficient. Atès que el programa no necessita cap altra informació extra per mostrar tots els contactes (i que, per tant, no es demanarà cap informació a l'usuari), aquesta opció no obrirà una altra activitat, sinó que mostrarà els contactes un a un per pantalla. Dins del *listener* del botó obtenim tots els contactes de la base de dades i fem un recorregut amb el cursor que se'ns retorna per anar mostrant tots els contactes.

```
1  else if (v == btnObtenirTots) {
2      //Obrir base de dades
3      bd.obre();
4
5      //Crida la BD per obtenir tots els contactes
6      Cursor c = bd.obtenirTotsElsContactes();
7
8      //Movem el cursor a la primera posició
9      if (c.moveToFirst()) {
10         do {
11             //Mostrem contactes...
12             MostraContacte(c);
13             //... mentre puguem passar al següent contacte
14         } while (c.moveToNext());
15     }
16     //Tanquem la BD
17     bd.tanca();
18
19     Toast.makeText(this, "Tots els contactes mostrats", Toast.
20                     LENGTH_SHORT).show();
21 }
```

El que hem fet és:

1. Obrir la base de dades
2. Cridar la BD per obtenir tots els contactes
3. Moure el cursor a la primera posició
4. Mostrar els contactes mentre es pugui
5. Tancar la base de dades
6. Mostrar un missatge per pantalla quan s'hagi acabat de mostrar els contactes.

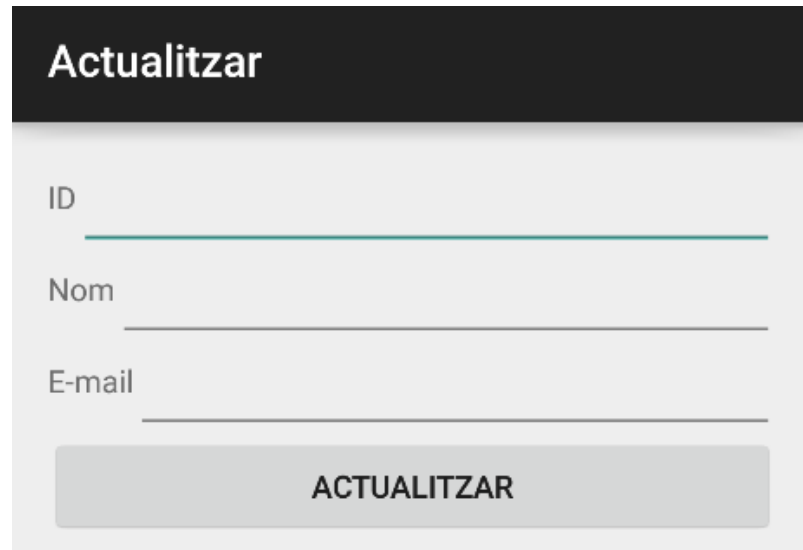
El mètode `MostraContacte()` simplement fa una crida a `Toast` per mostrar el contacte per pantalla.

```
1  public void MostraContacte(Cursor c)
2  {
3      Toast.makeText(this, "id: " + c.getString(0) + "\n" + "Nom: " + c.getString(1)
4                     + "\n Email: " + c.getString(2), Toast.LENGTH_SHORT).show();
5  }
```

## Actualitzar

L'opció d'actualitzar permet modificar les dades d'un contacte a partir del seu identificador. Quan hàgiu creat el *listener*, es mostrarà una activitat com la que es pot veure a la figura 1.6.

FIGURA 1.6. Activitat actualitzar



```
1 @Override
2 public void onClick(View v) {
3     if (v == btnActualitzar) {
4         long id;
5
6         //Obtenim la BD
7         bd = new DBInterface(this);
8         bd.obre();
9
10        //Identificador de la caixa de text
11        id = Long.parseLong(editID.getText().toString());
12
13        //Crida a la BD
14        boolean result = bd.actualitzarContacte(id, editNom.getText().
15            toString(), editEmail.getText().toString());
16
17        //Comprovem el resultat, si s'ha pogut actualitzar la BD o no
18        if (result)
19            Toast.makeText(this, "Element modificat", Toast.LENGTH_SHORT).
20                show();
21        else
22            Toast.makeText(this, "No s'ha pogut modificar l'element", Toast
23                .LENGTH_SHORT).show();
24
25        //Tanquem la BD
26        bd.tanca();
27
28        //Tanca l'activitat.
29        finish();
30    }
31 }
```



El que hem fet és:

1. Obrir la base de dades
2. Obtenir l'identificador de la caixa de text
3. Cridar la BD
4. Comprovar el resultat per determinar si s'ha pogut actualitzar la BD o no
5. Tancar la base de dades
6. Tancar l'activitat

El funcionament del codi és molt semblant. Obteniu les dades que necessiteu des dels *widgets* de l'activitat (les caixes de text) i feu la crida a la base de dades. En aquest cas, a més a més, es comprova el valor de retorn de la funció per saber si s'ha pogut actualitzar la base de dades correctament.

## Esborrar

La funció d'esborrar té una estructura molt similar a les altres funcions. Obté d'una caixa de text l'ID del contacte que es vol esborrar i fa una crida a la base de dades per esborrar el contingut corresponent. Es comprova el valor de retorn per saber si el contacte s'ha eliminat correctament.

```
1  @Override
2      public void onClick(View v) {
3          if (v == btnEsborrar) {
4              //Obrim la BD
5              bd = new DBInterface(this);
6              bd.obre();
7
8              //Obtenim l'ID de la caixa de text
9              long id = Long.parseLong(editID.getText().toString());
10
11             //Cridem la BD
12             boolean result = bd.esborraContacte(id);
13
14             //Comprovem el resultat de l'operació
15             if (result)
16                 Toast.makeText(this, "Element esborrat", Toast.LENGTH_SHORT).
17                     show();
18             else
19                 Toast.makeText(this, "No s'ha pogut esborrar l'element", Toast.
20                     LENGTH_SHORT).show();
21             //Tanquem la BD
22             bd.tanca();
23
24             //Tanquem l'activitat
25             finish();
26         }
27     }
```

El que hem fet és:

1. Obrir la base de dades

2. Obtenir l'identificador de la caixa de text
3. Cridar la BD
4. Comprovar el resultat de l'operació
5. Tancar la base de dades
6. Tancar l'activitat

## 1.4 Proveïdors de continguts

Els *content providers* (proveïdors de continguts) donen accés a una sèrie estructurada de dades i serveixen d'interfície de dades estàndard per connectar dades d'un procés amb el codi que s'està executant en un altre procés. Els proveïdors de continguts són la forma recomanada de compartir dades entre aplicacions. Són magatzems de dades que serveixen per compartir dades entre aplicacions. Es comporten de forma similar a una base de dades (podeu fer consultes, editar el contingut, afegir i esborrar), però a diferència d'aquestes fan servir diferents formes per emmagatzemar les seves dades. Aquestes poden estar en una base de dades, en fitxers o fins i tot a la xarxa, però per al procés que les fa servir això és transparent (i ens resulta indiferent).

Android fa servir una sèrie de proveïdors de continguts, estàndards del sistema, que poden fer servir la resta d'aplicacions, per exemple:

- *Browser*: emmagatzema dades com els marcadors del navegador, l'historial de navegació, etc.
- *CallLog*: emmagatzema dades com crides perdudes, detall de les trucades, etc.
- *ContactsContract*: emmagatzema informació dels contactes: nom, *email*, telèfon, foto, etc.
- *MediaStore*: emmagatzema dades multimèdia com imatges, àudio i vídeo.
- *Settings*: emmagatzema dades de configuració i preferències del dispositiu com el *bluetooth*, wifi, etc.

Podeu trobar una llista de proveïdors (del paquet `android.provider`) a la *Guia del desenvolupador* d'Android, que podeu trobar aquí:

<http://developer.android.com/reference/android/provider/package-summary.html>.

A banda d'aquests proveïdors de continguts, podeu crear els vostres propis.

Quan vulgueu accedir a les dades d'un proveïdor de continguts haureu de servir un objecte `ContentResolver` en el context de la vostra aplicació. Aquest objecte treballa com a client amb un objecte proveïdor, que treballa com a servidor:

un objecte `ContentProvider`. El proveïdor rep les dades dels clients, realitza la tasca i retorna els resultats. Heu de crear un `ContentProvider` si voleu compartir dades de la vostra aplicació amb altres, però per fer servir dades d'altres aplicacions simplement necessiteu un `ContentResolver`.

### 1.4.1 Accedint al proveïdor de continguts

El `ContentProvider` presenta dades a les aplicacions, com una o més taules similars a les que trobem a les bases de dades relacionals. Cada fila representa un element del proveïdor de continguts, i cada columna una dada concreta del mateix element de la fila.

L'aplicació accedeix a les dades del proveïdor a través de l'objecte client `ContentResolver`. Aquest objecte conté mètodes que criden altres mètodes (per cert, amb el mateix nom) a la classe `ContentProvider`. Per exemple, per obtenir una llista de contactes podeu cridar el mètode `ContentResolver.query()`, i aquest quest cridarà el mètode `query()` de `ContentProvider`. El mètode `query()` té una sèrie d'arguments que serveixen per definir la consulta que es vol fer al proveïdor. Aquests arguments tenen un paral·lelisme amb les opcions d'una consulta *query* a una base de dades:

- `Uri uri`: URI que representa la taula d'on s'obtidran les dades.
- `String[] projection`: llista de les columnes que es retornaran per cada fila de la taula.
- `String selection`: filtre que indica quines files retornar amb el mateix format que la clàusula `WHERE` d'SQL (sense la paraula "WHERE"). Si es passa un *null* retornarà totes les files de l'URI.
- `String[] selectionArgs`: en l'argument anterior (*selection*), en lloc d'incloure el valor de les columnes, directament podeu indicar "=" en la selecció. Els "?" seran substituïts pels valors de l'*array* `selectionArgs` en l'ordre en què apareixen a *selection*.
- `String sortOrder`: estableix com s'ordenen les files, amb el mateix format que la clàusula d'SQL `ORDER BY` (excloent les paraules "ORDER BY"). Si es passa *null* es farà servir l'ordre per defecte.

Es poden consultar els arguments i el seu equivalent en un `SELECT` d'SQL a la taula 1.2.

**TAULA 1.2.** Equivalència d'arguments entre `query()` i una consulta `SELECT` d'SQL

Argument de <code>query()</code>	Equivalent en <code>SELECT</code> d'SQL
<code>Uri uri</code>	<code>FROM taula</code>
<code>String[] projection</code>	<code>columna, columna, columna</code>
<code>String selection</code>	<code>WHERE col = valor</code>
. . . . .	

**TAULA 1.2** (continuació)

Argument de query()	Equivalent en SELECTd'SQL
String[] selectionArgs	No existeix un equivalent
String sortOrder	ORDER BY col, col,...

El mètode retorna un objecte de la classe `Cursor`, posicionat abans de la primera entrada o *null* en cas de trobar algun problema.

L'URI especifica les dades en el proveïdor. Els URI inclouen el nom del proveïdor (anomenat *authority*, 'autoritat') i un nom que apunta a una taula (*path*). El `ContentProvider` fa servir el *path* de l'URI per escollir la taula a la qual accedirà (té un *path* per a cada taula). L'estructura d'un URI és aquesta:

```
1 <prefix>://<authority>/<path>/<id>
```

on:

- El *prefix* per als proveïdors de continguts sempre és: `content://`.
- *authority* especifica el nom del proveïdor de continguts. Per als estàndards del sistema, per exemple: `media`, `call_log`, `browser`. Per a proveïdors de tercers, millor introduir un nom de domini complet com: `com.android.ioc`.
- El *path* especifica la taula dins del proveïdor.
- L'*id* és un identificador únic per a una fila concreta de la taula especificada al *path*.

A la taula 1.3 es poden veure alguns proveïdors de continguts del sistema.

**TAULA 1.3.** Alguns exemples de proveïdors de continguts del sistema.

String	Descripció
<code>content://media/internal/images</code>	Llista de les imatges en la memòria del dispositiu
<code>content://media/external/images</code>	Llista de les imatges en la memòria externa del dispositiu (per exemple, en la targeta SD)
<code>content://call_log/calls</code>	Llista de trucades fetes amb el dispositiu
<code>content://browser/bookmarks</code>	Llista dels marcadors del navegador web

D'altra banda, molts proveïdors de continguts tenen una constant amb l'URI d'accés a ells mateixos. Per exemple, en lloc de crear vosaltres l'URI per accedir al diccionari de paraules del dispositiu podeu fer servir el que ell mateix té definit: `UserDictionary.Words.CONTENT_URI`. O per accedir a la llista de contactes, `ContactsContract.Contacts.CONTENT_URI`.

Per obtenir dades d'un proveïdor, la vostra aplicació necessita d'un permís de lectura per part del proveïdor. No es pot demanar aquest permís durant l'execució, així que per obtenir-lo caldrà que feu servir l'element `<uses-permission>` en el vostre fitxer de *manifest* especificant el permís que voleu obtenir del proveïdor. En quedar definit al *manifest*, quan l'usuari instal·la aquesta aplicació li està

donant els permisos implícitament. Per saber exactament de quins permisos consta el proveïdor així com seu nom, consulteu la documentació del proveïdor. Per exemple, per demanar que la vostra aplicació tingui permisos de lectura sobre el proveïdor de contactes del dispositiu heu d'indicar-ho al fitxer de *manifest* de la següent manera, abans de l'etiqueta `<application>`:

```
1 <uses-permission android:name="android.permission.READ_CONTACTS">
2 </uses-permission>
```

El següent pas és realitzar una consulta al proveïdor per obtenir les seves dades. El següent codi mostra com fer una consulta per obtenir tots els contactes de la llista de contactes:

```
1 //Les columnes que volem obtenir per a cada element.
2 String[] projection = new String[] {
3     ContactsContract.Contacts._ID,
4     ContactsContract.Contacts.DISPLAY_NAME,
5     ContactsContract.Contacts.HAS_PHONE_NUMBER
6 };
7
8 //Condició: volem obtenir totes les files (per això és null).
9 String where = null;
10 String[] whereArgs = null;
11
12 //Ordre: que estiguin ordenats de forma ascendent.
13 String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " COLLATE LOCALIZED
14     ASC";
15
16 Cursor c = getContentResolver().query
17 (
18     ContactsContract.Contacts.CONTENT_URI,
19     projection, // Columnes per obtenir de cada fila
20     where,      // Criteri de selecció
21     whereArgs, // Criteri de selecció
22     sortOrder  // Ordre
23 );
```

Aquest codi obtindrà tota la llista de contactes i retornarà un cursor al resultat, la variable `c`. Quan hàgiu obtingut el cursor, heu de mirar el seu valor per saber si hi ha hagut un error, o si el cursor té dades o no. Ho podeu comprovar de la següent manera:

```
1 //Si hi ha hagut un error
2 if (c == null){
3     //Codi per tractar l'error, escriure logs, etc.
4
5 }
6 //Si el cursor està buit, el proveïdor no ha trobat resultats.
7 elseif (c.getCount() < 1) {
8     //El cursor està buit, el content provider no té elements.
9     Toast.makeText(this, "No hi ha dades", Toast.LENGTH_SHORT).show();
10 } else {
11     //Dades obtingudes
12     Toast.makeText(this, "OK", Toast.LENGTH_SHORT).show();
13 }
```

En aquests moments teniu un cursor a les dades obtingudes del proveïdor, així que podríeu recórrer les dades amb el cursor i treballar-hi. Per exemple, el següent codi mostra com obtenir l'identificador i el nom, i com saber si el contacte té número de telèfon:

```

1 //Mentre tenim un nou element al cursor
2 while(c.moveToNext()) {
3     //Obtenir ID
4     String contactId = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID
5     ));
6
7     //Obtenir nom
8     String nomContacte = c.getString(c.getColumnIndex(ContactsContract.Contacts.
9     DISPLAY_NAME));
10
11     //Saber si té telèfon
12     String hasPhone = c.getString(c.getColumnIndex(ContactsContract.Contacts.
13     HAS_PHONE_NUMBER));
14
15     //Mostrar
16     Toast.makeText(this, "id: " + contactId + "\n" + "Nom: " + nomContacte + "\n Té
17     telèfon: " + hasPhone , Toast.LENGTH_SHORT).show();
18 }
19 c.close();

```

Per obtenir una columna a partir del cursor heu de fer servir el mètode `Cursor.getString(int ColumnIndex)`, al qual se li passa l'índex de la columna que voleu obtenir. Per obtenir el número de columna a partir de l'identificador del camp que voleu, feu servir `Cursor.getColumnIndex(String columnName)`. Tot això ho podeu fer a la vegada:

```

1 String nomContacte = c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));

```

El codi anterior us mostra per pantalla la informació de tots els contactes del proveïdor tal com estan al cursor. Però, què passa si voleu filtrar aquesta informació? Treballant amb SQL ho podeu fer amb una sentència *WHERE columna = valor*. Això ho podeu definir al mètode `query()` amb els arguments `where` i `whereArgs`. En realitat, podríeu obtenir fàcilment la mateixa sentència *WHERE* modificant el valor de l'argument `where`. Si voleu obtenir al cursor únicament els contactes que tinguin telèfon, per exemple, podeu definir l'argument com:

```

1 String where = ContactsContract.Contacts.HAS_PHONE_NUMBER + "= 1";

```

O, per obtenir únicament els contactes amb nom IOC:

```

1 String where = ContactsContract.Contacts.DISPLAY_NAME + "= 'IOC'";

```

Per qüestions de seguretat i per evitar que l'usuari pugui introduir codi SQL maliciós, es pot fer servir el caràcter “?” a la variable `where` dins de la sentència. Els caràcters “?” seran substituïts amb els valors de l'array d'*strings* `whereArgs`. Per exemple, la sentència anterior seria equivalent a:

```

1 String where = ContactsContract.Contacts.DISPLAY_NAME + "= ?";
2 String[] whereArgs = {"IOC"};

```

### SQL Injection

L'objectiu de `whereArgs` i de la substitució dels “?” és impedir la inclusió de codi SQL maliciós dins de la sentència (vegeu aquest enllaç a la [Wikipedia](#)). Imagineu que creeu un codi per accedir a tots els contactes que tinguin un nom igual a una variable que

introdueix l'usuari ("WHERE name = valor\_variable"). I aquest usuari, en lloc d'introduir el nom d'una persona, introdueix: "nothing; DROP TABLE \*". La sentència SQL resultant, en sers executada: "WHERE name = nothing. DROP TABLE \*" aconseguiria esborrar totes les taules de la base de dades (si tingués permís per fer-ho).

Fins ara heu mostrat el nom, l'identificador i una variable que diu si el contacte té o no telèfon. Però obtenir el telèfon i el correu electrònic és una mica més complicat. Atès que a la llista un contacte pot tenir diversos números de telèfon i diverses adreces de correu electrònic, aquestes no estan emmagatzemades com variables estàtiques sinó com un proveïdor de continguts dins del proveïdor de continguts dels contactes. Per tant, per obtenir-les s'ha de fer una altra consulta i obtenir un cursor a la llista de telèfons i correus electrònics, respectivament. El següent codi recorre la llista de telèfons i correus i els va desant a una variable de tipus String (es podrien mostrar o fer qualsevol altra cosa amb ells). Al final, la variable es queda amb l'últim telèfon i correu que serà el que es mostri del contacte (encara que es podria mostrar el primer, o tots).

```
1 String telefon = null;
2 String email = null;
3
4 if (hasPhone.compareTo("1") == 0) {
5     // Obtenim els telèfons
6     Cursor telefons = getContentResolver().query(
7         ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
8         null,
9         ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = " + contactId,
10        null,
11        null);
12
13    //Recorrem els telèfons
14    while (telefons.moveToNext()) {
15        telefon = phones.getString(phones.getColumnIndex( ContactsContract.
16            CommonDataKinds.Phone.NUMBER));
17    }
18    //Tanquem el cursor
19    telefons.close();
20 }
21
22 //Obtenir cursor correus
23 Cursor emails = getContentResolver().query(ContactsContract.CommonDataKinds.
24     Email.CONTENT_URI, null, ContactsContract.CommonDataKinds.Email.CONTACT_ID
25     + " = " + contactId, null, null);
26
27 //Recorrem els correus
28 while (emails.moveToNext()) {
29     email = emails.getString(emails.getColumnIndex(ContactsContract.
30         CommonDataKinds.Email.DATA));
31 }
32 //Tanquem el cursor
33 emails.close();
34
35 //Mostrar
36 Toast.makeText(this, "id: " + contactId + "\n" + "Nom: " + nomContacte + "\n
37     Telefon: " + telefon + "\n email: " + email, Toast.LENGTH_SHORT).show();
```

### 1.4.2 Inserint dades

La forma d’inserir dades a un proveïdor de continguts és similar a la consulta. Per fer-ho, existeix el mètode `ContentResolver.insert()`. Aquest mètode insereix una nova fila al proveïdor de continguts i retorna l’URI de la fila creada. Els valors de la fila queden definits amb un objecte de la classe `ContentValues`. Per exemple, el següent codi serveix per inserir una nova paraula al diccionari personal de l’usuari del dispositiu. Perquè funcioni, s’han de donar permisos d’escriptura al diccionari de dades del dispositiu inserint la següent sentència al document de *manifest*:

```
1 <uses-permission android:name="android.permission.WRITE_USER_DICTIONARY">
2 </uses-permission>
```

El següent fragment de codi mostra com s’insereix un nou valor al proveïdor de continguts:

```
1 Uri UriNou;
2
3 ContentValues Valors = new ContentValues();
4
5 //Creem el valor de la nova entrada
6 Valors.put(UserDictionary.Words.APP_ID, "com.android.ioc");
7 Valors.put(UserDictionary.Words.LOCALE, "es_ES");
8 Valors.put(UserDictionary.Words.WORD, "Hospitalet");
9 Valors.put(UserDictionary.Words.FREQUENCY, "100");
10
11 //Inserim
12 UriNou = getContentResolver().insert(
13     UserDictionary.Words.CONTENT_URI,
14     Valors
15 );
```

## 1.5 Publicant aplicacions

Publicar és el procés de fer les vostres aplicacions d’Android disponibles per als usuaris. La publicació consta principalment de dues tasques:

- Preparar l’aplicació per a la distribució, compilant una versió de l’aplicació per a distribució.
- Distribuir l’aplicació als usuaris, on es publicita, ven i distribueix la versió del programa als usuaris.

El procés de publicació es realitza després d’haver comprovat el funcionament de l’aplicació en un entorn de depuració.

Preparar l’aplicació consta d’una sèrie de passos:



- Compileu i signeu la versió de distribució de l'aplicació. L'Android Studio proporciona tot el necessari per fer-ho.
- Traieu les crides a Log. També, quan feu la generació del fitxer .apk assegureu-vos de seleccionar a *build type* l'opció *release* i no *debug*. A més, afegiu o modifiqueu els valors dels atributs `versionCode` i `versionName` que estan a l'element `defaultConfig` del `build.gradle`
- Abans de distribuir l'aplicació, heu de comprovar la versió de distribució d'aquesta. Idealment, l'hauríeu de comprovar almenys en un telèfon i una tauleta.
- Assegureu-vos que tots els recursos (imatges, vídeos...) de l'aplicació estan actualitzats.
- Prepareu els servidors remots i els serveis si és que la vostra aplicació depèn d'aquests.
- Creeu una icona per a l'aplicació.
- Si voleu, podeu preparar un EULA (*End User License Agreement*, acord de llicència d'usuari final) per protegir la vostra propietat intel·lectual i la vostra persona.

Quan acabeu de preparar l'aplicació, creareu un fitxer signat .apk que podreu distribuir als usuaris.

### 1.5.1 Preparar l'aplicació

Per publicar una aplicació heu de crear un paquet que els usuaris puguin instal·lar i executar en els seus dispositius Android (versió *release*). El paquet de publicació de la versió conté els mateixos components que el fitxer .apk de depuració (codi font compilat, recursos, fitxer de *manifest...*) i es construeix amb les mateixes eines. Però el fitxer .apk que es publica està signat amb el vostre certificat i està optimitzat.

Per preparar l'aplicació per ser publicada, generalment es realitzen les següents tasques:

1. Preparació de materials i recursos.
2. Configuració de l'aplicació per a la publicació.
3. Compilació de l'aplicació per a la publicació.
4. Preparació dels servidors externs i recursos
5. Comprovació de l'aplicació per a la publicació.

## Claus criptogràfiques

El sistema Android requereix que cada aplicació instal·lada estigui signada digitalment amb un certificat propietat del seu desenvolupador (un certificat del qual el desenvolupador en tingui la clau privada). Android fa servir aquest certificat per identificar l'autor i establir relacions de confiança entre aplicacions.

---

Un requeriment per publicar a Google Play és que l'aplicació estigui signada amb una clau criptogràfica amb un període de validesa que acabi després del 22 d'octubre de 2033.

---

## Icona de l'aplicació

La vostra aplicació ha de tenir una icona i ha de seguir les recomanacions sobre com han de ser aquestes, tal com podeu trobar a la *Guia del Desenvolupador* d'Android:

[http://developer.android.com/guide/practices/ui\\_guidelines/icon\\_design\\_launcher.html](http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html)

La icona de l'aplicació serveix per identificar-la i pot aparèixer a la pantalla principal, al *launcher* del dispositiu, a *les meves descàrregues*, quan es gestionen les aplicacions instal·lades... A més a més, si publiqueu l'aplicació a Google Play es mostrarà la icona als usuaris (en aquest cas també heu de publicar una versió d'alta resolució de la icona).

## Altres continguts

Potser voleu preparar un EULA (*End User License Agreement*, acord de llicència d'usuari final). També, si voleu publicar la vostra aplicació a Google Play, heu de generar un text explicatiu del que fa i algunes captures de pantalla.

## Configurar l'aplicació per a alliberament

Aquestes són recomanacions de configuracions de l'aplicació:

- Escolliu un bon nom per al *package* de l'aplicació: no es podrà modificar després d'haver-la desplegat. Es pot configurar des del fitxer de *manifest*.
- Netegeu els directoris del projecte de fitxers innecessaris.
- Actualitzeu les configuracions del fitxer de *manifest*. Doneu valor als atributs `versionCode` i `versionName` del build.gradle tal com està explicat a la *Guia del desenvolupador* d'Android (<http://developer.android.com/tools/building/configuring-gradle.html>).
- Trebal·leu en la compatibilitat de l'aplicació. Afegiu suport per a diferents pantalles i versions d'Android.
- Actualitzeu les URL dels servidors i els serveis.

## Construir l'aplicació signada

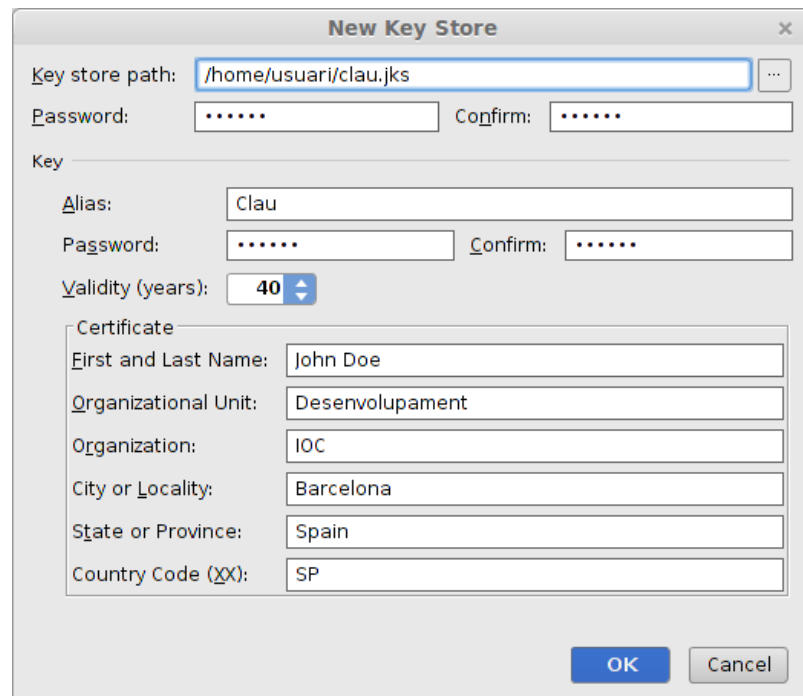
Si esteu fent servir Android Studio, podeu fer servir l'assistent d'exportació per crear una clau i exportar un fitxer .apk signat. Una clau privada adequada per signar l'aplicació ha de complir les següents característiques:

- És de la vostra possessió
- Representa la persona, corporació o organització que s'ha d'identificar amb l'aplicació
- Té un període de validesa que supera el temps de vida esperat de l'aplicació. Es recomana un període de validesa d'almenys 25 anys
- La clau no és la clau per defecte creada per les eines de l'SDK

Si no disposeu d'una clau, en podeu generar una amb l'ordre *keytool*. Tant si ho feu amb *keytool* o amb l'AndroidStudio heu de proporcionar alguns arguments:

- *Alias*: un nom o àlies per a la clau.
- *Password*: contrasenya de la clau.
- *Validity*: temps de validesa. En Android Studio està expressat en anys, i en *keytool* en dies.
- *First and Last Name*: nom i cognoms de la persona que signa.
- *Organization Unit*: organització.
- *City*: ciutat.
- *State or Province*: estat o província.
- *Country Code*: codi de país.

Per crear una clau i signar-la amb Android Studio seleccioneu *Build/Generate Signed APK...*, i ompliu les diferents opcions que us demana, tal com es pot veure a la figura 1.7. Podeu escollir una clau que ja tingueu desada al disc o crear-ne una de nova.

**FIGURA 1.7.** Signant una aplicació

En el cas que feu servir llibreries de tercers (per exemple, la llibreria externa de Google Maps), és possible que necessiteu altres claus. En el cas de Google Maps, heu de registrar la vostra aplicació al servei de Google Maps i obtindreu una clau de l'API de Maps.

### Comprovació de funcionament

Feu una comprovació del funcionament de l'aplicació, idealment en un dispositiu real (preferiblement un telèfon i una tauleta). A la *Guia del desenvolupador* d'Android hi trobareu consells de les coses que es poden comprovar de les aplicacions abans de distribuir-les: [http://developer.android.com/guide/topics/testing/what\\_to\\_test.html](http://developer.android.com/guide/topics/testing/what_to_test.html).

## 1.5.2 Distribuir aplicacions

Podeu distribuir la vostra aplicació de diferents maneres. La forma habitual de distribuir una aplicació és a través d'un mercat d'aplicacions (abans anomenat Android Market, ara Google Play). Però també podeu distribuir les aplicacions directament.

### Distribuïnt l'aplicació mitjançant Google Play

Google Play ha substituït el Google Market com la forma més habitual de distribuir aplicacions d'Android. És una plataforma que us permet publicitar, vendre i

distribuir la vostra aplicació d'Android a usuaris de tot el món. Quan publiqueu a través de Google Play teniu accés a una sèrie d'eines de desenvolupador que us permeten analitzar les vendes i les tendències del mercat, així com controlar a qui es distribuirà la vostra aplicació. Per distribuir una aplicació a Google Play cal que seguiu aquests tres passos:

1. Prepareu materials promocionals per vendre l'aplicació com captures de pantalla, vídeos, gràfics i un text explicatiu.
2. Configureu les opcions i pugeu els continguts. Configurant alguns ajustaments de Google Play podeu escollir en quins països voleu que estigui disponible la vostra aplicació, el llistat de llenguatges que voleu fer servir i el preu que voleu cobrar en cada país. També podeu configurar detalls de l'aplicació com el tipus d'aplicació, la categoria i el tipus de contingut. Quan l'hàgiu acabat, podeu pujar els continguts promocionals de la vostra aplicació com a esborrany.
3. Publiqueu la versió definitiva de l'aplicació. Si l'esborrany pujat és correcte, podeu fer clic a *Publicar* i en uns minuts la vostra aplicació estarà disponible per descarregar arreu del món.

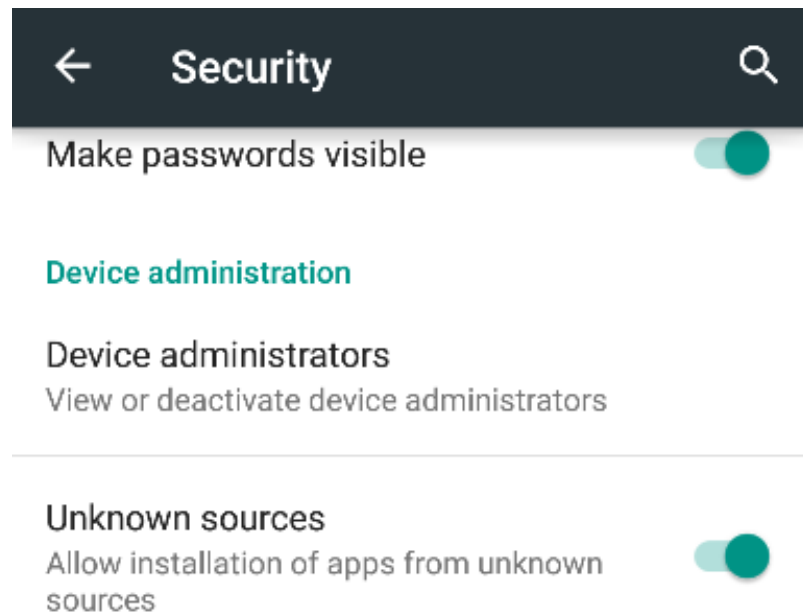
---

Per a una informació més detallada de com publicar a Google Play, podeu consultar la darrera informació publicada per Google a <http://developer.android.com/distribute/tools/launch-checklist.html>.

---

### **Publicar l'aplicació des de la vostra pròpia web**

Podeu distribuir la vostra aplicació fent-la descarregable des d'una pàgina web emmagatzemada en un servidor de la vostra elecció. Tot el que heu de fer és crear el fitxer .apk definitiu signat i crear un enllaç directe al fitxer des de la vostra pàgina web. Quan els usuaris visitin la pàgina des d'un dispositiu Android i descarreguin l'aplicació, el sistema Android detectarà el tipus de fitxer i instal·larà l'aplicació al dispositiu. Això únicament es podrà fer si l'usuari ha configurat el seu dispositiu per permetre la instal·lació d'aplicacions desconegudes, com es pot veure a la figura 1.8.

**FIGURA 1.8.** Opció d'habilitar aplicacions d'origen desconegut a Android Lollipop

Publicar a través de la vostra pròpia web és una forma senzilla de publicar l'aplicació, però aquesta no tindrà la visibilitat que tenen les aplicacions a Google Play, i si voleu cobrar per ella n'haureu de gestionar vosaltres mateixos el pagament.

### **Publicant per correu electrònic**

La forma més fàcil i ràpida de publicar una aplicació en un dispositiu és enviar el fitxer .apk signat per correu electrònic i obrir-lo des d'un dispositiu Android. El dispositiu reconeixerà el tipus de fitxer i instal·larà l'aplicació. Aquesta és la forma més còmoda de provar les vostres aplicacions en un telèfon o d'enviar la vostra aplicació a un grup reduït de destinataris (per exemple, la resta de membres del vostre grup de treball).

## 2. Programació de comunicacions

Cada vegada és més habitual que les aplicacions facin ús de la comunicació per Internet, integrant-se en les xarxes socials o utilitzant serveis web. En aquest apartat veureu com fer ús de les comunicacions del dispositiu per tal de comunicar-vos amb l'exterior.

### 2.1 Comunicacions en Android

El primer pas per treballar amb l'API de comunicacions en Android és demanar els permisos per a la vostra aplicació al document de *manifest*. Això ho podeu fer afegint les següents línies en l'esmentat document:

```
1 <uses-permission android:name="android.permission.INTERNET"/>
2 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Això permet a la vostra aplicació tenir accés a Internet (obrir *sockets* de xarxa) i a informació de l'estat de la xarxa (per exemple, per saber si el Wi-Fi està activat o desactivat).

Abans de fer un intent de connexió a la xarxa, hauríeu de comprovar si el dispositiu té una connexió de xarxa disponible i funcionant correctament. Per això heu de fer servir objectes de la classe `ConnectivityManager` i `NetworkInfo` de la següent manera:

```
1 //Obtenim un gestor de les connexions de xarxa
2 ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.
   CONNECTIVITY_SERVICE);
3
4 //Obtenim l'estat de la xarxa
5 NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
6
7 //Si està connectat
8 if (networkInfo != null && networkInfo.isConnected()) {
9     //Xarxa OK
10    Toast.makeText(this, "Xarxa ok", Toast.LENGTH_LONG).show();
11 } else {
12     //Xarxa no disponible
13    Toast.makeText(this, "Xarxa no disponible", Toast.LENGTH_LONG).show();
14 }
```

Amb `getSystemService()` obteniu un objecte `ConnectivityManager` que serveix per gestionar la connexió de xarxa. Amb aquest objecte podeu obtenir un objecte `NetworkInfo`, amb `getActiveNetworkInfo()` que retorna una instància de `NetworkInfo`, que representa la primera interfície de xarxa que pot trobar o un *null* si cap connexió està connectada.

A `NetworkInfo` teniu el mètode `isConnected()`, que indica si existeix connectivitat a la xarxa i si és possible establir connexions.

En el cas que vulgueu obtenir informació individual dels diferents tipus de xarxes, ho podeu fer de forma individual amb `getNetworkingInfo()`, amb un argument que és una constant que indica el tipus de xarxa que voleu comprovar, com es pot veure en el següent codi:

```
1 //Obtenim l'estat de la xarxa mòbil
2 NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.
   TYPE_MOBILE);
3 boolean connectat3G = networkInfo.isConnected();
4
5 //Obtenim l'estat de la xarxa Wi-Fi
6 networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
7 boolean connectatWifi = networkInfo.isConnected();
```

És molt important que sempre proveu la connexió a la xarxa i no doneu per fet que la xarxa està disponible. Penseu que les operacions de xarxa a un dispositiu mòbil solen ser mòbils, 3G o Wi-Fi, i existeixen possibilitats reals de caiguda de la connexió. Per tant, heu de fer aquestes comprovacions quan engegueu l'aplicació o quan hi torneu després d'haver-ne sortit. El lloc ideal per fer-ho seria al mètode `onStart()` de la vostra aplicació.

Imagineu que engegueu l'aplicació i es fa la comprovació de connexió al començament, i trobeu que existeix connexió. Si l'usuari executa l'aplicació de configuració del dispositiu i activa el mode d'avió, la vostra aplicació seguirà funcionant correctament, ja que quan l'usuari torni a la vostra aplicació `onStart()` tornarà a ser cridat i detectareu que la xarxa està desconnectada.

Però què passa quan la connexió a Internet canvia d'estat mentre esteu dins de la vostra aplicació? Això pot passar perquè el dispositiu perdi la cobertura o la connexió a la xarxa. Com que no es torna a cridar `onStart()`, no podríeu comprovar la connectivitat. Per això, el que heu de fer és registrar un *broadcastReceiver* (receptor *broadcast* o multidifusió) que escoltarà els canvis en l'estat de la connexió i comprovarà quin és l'estat de la xarxa.

En primer lloc, creeu una classe que hereti de `BroadcastReceiver` i implementeu el mètode abstracte `onReceive()`.

```
1 public class ReceptorXarxa extends BroadcastReceiver {
2
3     @Override
4     public void onReceive(Context arg0, Intent arg1) {
5         //Actualitzar l'estat de la xarxa
6         ActualitzaEstatXarxa();
7     }
8 }
```

Si aquesta classe es farà servir dins de la vostra activitat no cal que creeu la classe com a un fitxer `.java` independent al vostre projecte, pot estar definida dins de la classe de la vostra aplicació. Dins del mètode `onReceive()` actualitzareu l'estat de la xarxa a la vostra aplicació.



El següent pas és registrar el receptor de *broadcast* perquè escolti els missatges relacionats amb els canvis de la connectivitat de la xarxa. Per fer-ho, creeu una instància de la classe receptora i registreu-la amb un filtre d'*intent* per escoltar únicament els missatges de *broadcast* sobre la connectivitat del dispositiu.

```
1 private ReceptorXarxa receptor;
2
3 @Override
4 public void onCreate(Bundle savedInstanceState) {
5     (...) //Resta de codi onCreate()
6
7     IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
8     ;
9     receptor = new ReceptorXarxa();
10    this.registerReceiver(receptor, filter);
```

El mètode `registerReceiver()` registra un receptor de *broadcast* per executar-se al fil principal de l'aplicació. El receptor de *broadcast* escoltarà qualsevol *intent* de *broadcast* que coincideixi amb el filtre definit en el fil principal de l'aplicació.

En aquest moment, els missatges de *broadcast* que anunciïn els canvis en la connectivitat de la xarxa seran rebuts pel vostre receptor, el qual actualitzarà l'estat de la xarxa de l'aplicació.

Tenir un receptor de *broadcast* que és cridat constantment o innecessàriament pot derivar en un consum excessiu dels recursos del sistema (entre d'altres, de la bateria). Si declareu el receptor de *broadcast* en el document de *manifest*, aquest pot engegar l'aplicació automàticament, encara que aquesta no estigui en execució.

Per aquest motiu, el més adequat seria registrar el receptor de *broadcast* quan es crea l'aplicació `onCreate()` i donar-lo de baixa al mètode `onDestroy()`.

```
1 public void onDestroy() {
2     super.onDestroy();
3
4     //Donem de baixa el receptor de broadcast quan es destrueix l'aplicació
5     if (receptor != null) {
6         this.unregisterReceiver(receptor);
7     }
8 }
```

## 2.2 Mostrar pàgines web amb un 'widget'

Possiblement, el servei de comunicació més utilitzat avui en dia són les pàgines web, i per aquest motiu començarem mostrant com podem visualitzar-les dins les nostres aplicacions.

El procediment és molt senzill perquè hi ha un component anomenat **WebView** que ja fa gairebé tota la feina, només cal dir-li quina adreça volem visitar.

Creeu un nou projecte Android. Com que la vostra aplicació accedirà, lògicament, a Internet, heu d'afegir-hi el permís corresponent al fitxer **AndroidManifest.xml** (tot seguit del `uses-sdk`):

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

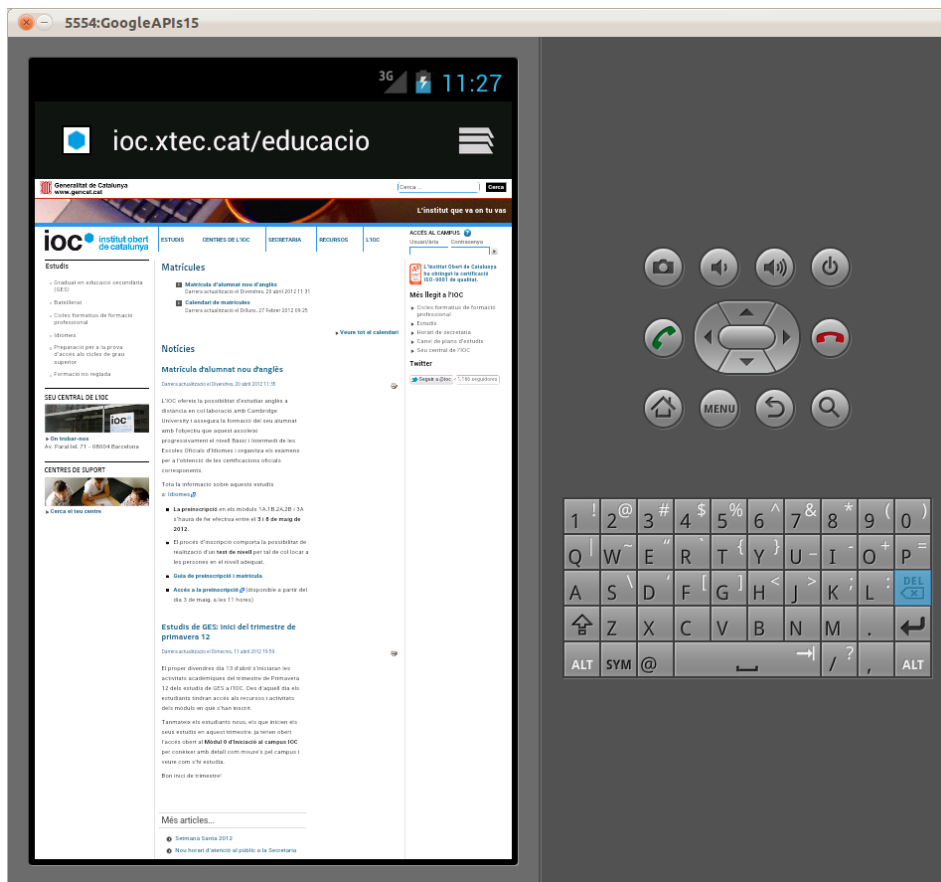
Guardeu i aneu al **layout.xml** per afegir el component **WebView**, a més d'un **EditText** per introduir l'adreça web i un botó per anar-hi:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="fill_parent"
4     android:layout_height="fill_parent"
5     android:orientation="vertical">
6
7     <LinearLayout
8         android:id="@+id/linearLayout1"
9         android:layout_width="match_parent"
10        android:layout_height="wrap_content"
11        android:orientation="horizontal">
12
13        <EditText
14            android:id="@+id/editText1"
15            android:layout_width="264dp"
16            android:layout_height="wrap_content"
17            android:hint="Introdueix l'adreça web">
18
19            <requestFocus/>
20        </EditText>
21
22        <Button
23            android:id="@+id/button1"
24            android:layout_width="wrap_content"
25            android:layout_height="wrap_content"
26            android:onClick="onClickAnar"
27            android:text="Anar"/>
28
29    </LinearLayout>
30
31    <WebView
32        android:id="@+id/webView1"
33        android:layout_width="match_parent"
34        android:layout_height="match_parent"/>
35
36</LinearLayout>
```

Només cal anar al codi de l'activitat per afegir la funcionalitat necessària. Bàsicament, es tracta d'escriure el mètode `onClickAnar()` que s'activarà quan cliquem el botó, i que ha de dir al **WebView** que obri una pàgina:

```
1 public void onClickAnar(View v) {
2     WebView webView = (WebView) findViewById(R.id.webView1);
3     EditText editText = (EditText) findViewById(R.id.editText1);
4
5     String adreca = editText.getText().toString();
6     webView.loadUrl(adreca);
7 }
```

Amb això podem executar l'aplicació, indicar un web i anar-hi, tal com es pot veure a la figura 2.1.

**FIGURA 2.1.** Obrint la pàgina web de l'IOC

Observareu que, si escriviu una adreça sense el prefix “http://”, l’aplicació dóna un error. Això és perquè realment l’adreça completa ha d’incloure aquest prefix (que és el protocol d’aplicació). Per tant, podeu afegir el següent codi perquè ho faci, editant el mètode `onClickAnar()`:

```

1 String adreca = editText.getText().toString();
2
3 if(!adreca.startsWith("http://") && !adreca.startsWith("https://")) {
4     adreca = "http://" + adreca;
5     editText.setText(adreca);
6 }
7
8 webView.loadUrl(adreca);

```

Podeu comprovar com fent doble clic a la pàgina la vista s’apropa (fa zoom) i us permet arrossegar-la per tal d’anar-vos desplaçant per tots els seus continguts segons us calgui, així que podeu clicar enllaços i fer qualsevol altra activitat amb les pàgines que aneu obrint.

## 2.3 Model dels fils

Quan s’engega una aplicació, el sistema Android posa en marxa un nou procés per a l’aplicació amb un únic fil d’execució.

### Processos del sistema

Un procés és una instància d’un programa que s’està executant. Conté el codi i els recursos que està fent servir en aquests moments (per exemple, la memòria reservada). Depenent del sistema operatiu, pot contenir diversos fils d’execució que s’executen concurrentment.

### Fils d'execució

Un fil d'execució és la unitat de processament més petita que es pot programar en un sistema operatiu. Poden existir múltiples fils dins d'un procés i compartir recursos (com memòria, codi i context), cosa que no succeeix entre diferents processos.

Per defecte, tots els components de l'aplicació s'executen en aquest fil d'execució. Però en alguns casos pot ser convenient executar diferents tasques de l'aplicació en fils d'execució independents.

Quan una aplicació accedeix a Internet o a altres recursos externs, es produeix un retard de durada variable però que pot arribar a ser de molts segons, o fins i tot d'alguns minuts, des que l'aplicació demana les dades fins que aquestes hi arriben.

Si l'aplicació funciona amb un únic fil d'execució, aquest es pot quedar aturat en la crida a un mètode (per exemple, alguna que descarregui dades d'un servidor extern, el qual pot tardar a enviar la informació). En aquest cas, l'execució de l'aplicació es quedarà bloquejada fins que aquest mètode retorni les dades i així es continuï executant la resta del codi.

Si això ocorregués, els usuaris tindrien la impressió que l'aplicació s'ha quedat penjada i segurament la tancarien i no tornarien a obrir-la. És necessari que l'aplicació continuï responent a les accions de l'usuari i, si el temps d'espera és molt llarg, convé que mostri algun tipus d'animació o barra de progrés per tal que l'usuari vegi que l'aplicació està funcionant.

El mecanisme que permet que les aplicacions puguin fer dues o més operacions alhora són els fils (*threads*, en anglès).

Crear i sincronitzar dos fils és una tasca relativament complexa, i per aquest motiu la llibreria d'Android ens facilita la classe `AsyncTask`, que es fa càrrec de la major part de la feina per nosaltres.

El que fa l'`AsyncTask` és crear un altre fil que podeu fer servir per a qualsevol operació que preveieu que pugui tenir una durada notable i que, per tant, pugui bloquejar la interfície d'usuari. Concretament, qualsevol operació d'accés a Internet hauria de fer-se en un fil separat obligatòriament, i la manera més senzilla és fer servir una `AsyncTask`.

`AsyncTask` s'hauria de fer servir, idealment, per a operacions curtes (d'alguns segons, com a molt). Si necessiteu que els fils estiguin executant-se durant períodes de temps llargs, es recomana que feu servir l'API del paquet `java.util.concurrent`.

Una tasca asíncrona es defineix com una tasca que s'executa en el fil de *background* (fil BG, per *background*, és a dir, que s'executa pel darrere) i els resultats de la qual es mostren en el fil de la interfície d'usuari (l'anomenarem fil UI, de *User Interface*).

Per crear una `AsyncTask` heu de crear una nova classe que derivi d'`AsyncTask`, i implementar els mètodes descrits a continuació. El més important és saber que alguns mètodes s'executen al fil UI i d'altres al fil que fa l'operació en el *background*. Això és fonamental perquè només es poden modificar les vistes de l'aplicació (escriure valors, mostrar missatges, fer avançar barres de progrés o mostrar altre tipus de continguts) des del fil UI.

Un `AsyncTask` es defineix fent servir tres tipus genèrics:

---

Quan es fa servir un *widget* com `WebView` no cal un `AsyncTask`, perquè el `WebView` ja genera un fil propi internament.

---

- **Params:** el tipus dels paràmetres enviats a la tasca durant l'execució.
- **Progress:** el tipus d'unitats de progrés publicades durant la computació en el *background*. Això serveix per si voleu mostrar un progrés de la tasca feta mentre aquesta s'executa en el *background*. Per exemple, podríeu animar una barra per mostrar el progrés de la càrrega d'un fitxer.
- **Result:** el tipus del resultat de la computació en el *background*.

No sempre es fan servir tots els tipus en una tasca asíncrona. Per marcar un tipus que no es farà servir, simplement utilitzeu el tipus *void*. Per exemple, la següent classe asíncrona defineix que tindrà un argument de tipus *String* i retornarà un *Bitmap*. No farà servir cap tipus de progrés de la tasca:

```
1 private class CarregaImatge extends AsyncTask<String, Void, Bitmap> {  
2     ...  
3 }
```

Quan s'executa una tasca de forma asíncrona, aquesta realitza quatre passos:

- **onPreExecute()** (UI): executat a l'UI immediatament després que la tasca és executada. Permet inicialitzar els elements de la interfície que siguin necessaris per a la tasca, com ara crear una barra de progrés.
- **doInBackground(Params...)** (BG): és cridada després que **onPreExecute()** acaba d'executar-se. Posa en marxa l'operació que voleu executar en un altre fil en el *background*. Aquest mètode rep els paràmetres necessaris per fer l'operació asíncrona. El resultat de l'execució s'ha de retornar i es passarà de tornada en l'últim pas. Dins de **doInBackground()** es pot cridar **publishProgress(Progress...)** per publicar una o més unitats de resultat. Aquests valors es publiquen en el fil UI, a **onProgressUpdate(Progress...)**.
- **onProgressUpdate(Progress...)** (UI): és cridat en el fil UI després d'haver cridat **publishProgress(Progress...)**. Us permet actualitzar algun element de la interfície per mostrar el progrés de l'operació mentre l'operació en el *background* encara està en execució, com ara actualitzar una animació o fer avançar una barra de progrés.
- **onPostExecute(Result)** (UI): aquest mètode és cridat quan l'execució en el *background* finalitza. Rep el resultat de l'operació que ha executat l'*AsyncTask* i com que s'executa al fil UI permet visualitzar els resultats obtinguts (com ara una imatge, una pàgina web o una llista de *tweets*).

Per fer servir *AsyncTask* us heu de crear la vostra classe, que hereti d'*AsyncTask* i implementi el mètode de *callback* **doInBackground()**, que s'executa en un fil en el *background*. Per actualitzar la vostra interfície d'usuari (o UI, *user interface*) heu d'implementar el mètode **onPostExecute()** que incorpora els resultats d'haver executat **doInBackground()** i s'executa en el fil UI, i per tant pot actualitzar l'estat de la vostra interfície sense problemes. Quan tingueu la classe creada, per llançar el fil de l'*AsyncTask* heu d'elaborar un nou objecte de

la subclasse que heu creat i cridar el mètode `execute()`, passant-li la informació necessària perquè pugui fer l'operació.

Per exemple, la següent classe descarrega una imatge de la xarxa, mitjançant el mètode propi `Bitmap DescarregaImatgeDeLaXarxa(String url)`, i fa servir el `Bitmap` per mostrar-lo a un *widget*:

```

1 private class CarregaImatge extends AsyncTask<String, Void, Bitmap> {
2
3     @Override
4     protected Bitmap doInBackground(String... url) {
5         //Descarreguem la imatge d'Internet
6         return DescarregaImatgeDeLaXarxa(url);
7     }
8
9     protected void onPostExecute(Bitmap result) {
10        //Rebem el resultat de doInBackground (el Bitmap)
11        //i el fem servir per carregar la imatge a l'UI
12        mImageView.setImageBitmap(result);
13    }
14 }
```

Existeixen algunes regles per tal que `AsyncTask` funcioni correctament:

- La classe `AsyncTask` s'ha de crear al fil UI.
- La instància de la tasca s'ha de crear al fil UI.
- `execute(Params...)` s'ha d'executar des del fil UI.
- Mai heu de cridar `onPreExecute()`, `onPostExecute(Result)`, `doInBackground(Params...)` o `onProgressUpdate(Progress...)` directament.
- La tasca es pot executar únicament una vegada al mateix temps (si n'intenteu executar una altra, en llançarà una excepció).

## 2.4 Connexions HTTP

El protocol HTTP (*HyperText Transfer Protocol*, protocol de transferència d'hipertext) és un protocol estàndard d'Internet que serveix per a la transferència d'hipertext. És el protocol base sobre el qual està fonamentada la WWW (*World Wide Web*, 'xarxa d'extensió mundial'). Fent servir HTTP a la vostra aplicació podreu realitzar diverses tasques, com descarregar-vos informació de text i binària (com ara pàgines web o imatges, respectivament).

Un *hipertext* és un text mostrat a un ordinador o un altre dispositiu electrònic amb enllaços (*hyperlinks*) a altres textos, que són d'accés immediat pel lector a través d'un clic del ratolí o de la pantalla tàctil. A banda de text, pot contenir taules, imatges i altres formats de representació de continguts multimèdia.

També es pot fer servir el protocol HTTPS (*HTTP Secure*, HTTP segur) per a comunicacions segures a través d'una xarxa d'ordinadors. HTTPS proporciona

autenticació del lloc web i encriptació bidireccional en les comunicacions entre el client i el servidor.

Android inclou dos clients HTTP que podeu fer servir a la vostra aplicació: `URLConnection` i el client d'Apache `HttpClient`. Ambdós suporten HTTPS, fluxos de pujada i descàrrega i IPv6. Segons articles de la pròpia Google, el client HTTP d'Apache és més estable a les versions d'Android Eclair (versió 2.1) i Froyo (versió 2.2). Però per a les versions Gingerbread (versió 2.3) i posteriors recomana fer servir `URLConnection`, en el que enfocarà els seus esforços futurs.

Una connexió `URLConnection` permet enviar i rebre dades per la web. Les dades poden ser de qualsevol tipus i mida, fins i tot podeu enviar i rebre dades de les quals no coneixeu la mida prèviament. Per usar aquesta classe heu de seguir els següents passos:

- Obtingueu un objecte nou `URLConnection` cridant el mètode `URL.openConnection()` i forçant el seu resultat a `URLConnection`.
- Prepareu la petició. La principal propietat de la petició és la seva URI. Les capçaleres de la petició poden tenir altres metadades com credencials, tipus de contingut, galetes de sessió, etc.
- Si aneu a pujar dades a un servidor, la instància ha d'estar configurada amb `setDoOutput(true)`. Heu de transmetre dades escrivint al flux de dades retornat per `getOutputStream()`.
- Llegiu la resposta. Les capçaleres de la resposta generalment inclouen metadades com el tipus de dades de la resposta i la seva mida, la data de modificació o galetes de sessió, entre d'altres. El cos de la resposta es pot llegir del flux retornat per `getInputStream()`. Si la resposta no té cos (*body* en anglès), el mètode retorna un flux buit.
- Desconnecteu. Quan hàgiu llegit la resposta, heu de tancar l'`URLConnection` amb `disconnect()`.

Per exemple, amb el següent codi estaríeu llegint el contingut del web de l'IOC:

```
1 URL url =new URL("http://ioc.xtec.cat/");
2 HttpURLConnection urlConnection =(URLConnection) url.openConnection();
3 try {
4     InputStream in = new Buffered InputStream(urlConnection.getInputStream());
5     treballarAmbElFluxDeDades(in);
6 } finally {
7     urlConnection.disconnect();
8 }
9 }
```

### 2.4.1 Comunicacions segures amb HTTPS

Si crideu `openConnection()` amb una URL que comenci amb la cadena “https”, se us retornarà un objecte `HttpsURLConnection` (fixeu-vos que no és el mateix que l'anterior, té una “s” al final del nom). Aquesta classe us permet especificar uns `HostnameVerifier` i `SSLSocketFactory` necessaris per al protocol HTTPS. Podeu consultar el funcionament d'aquesta classe a la *Guia del desenvolupador d'Android* <http://developer.android.com/reference/javax/net/ssl/HttpsURLConnection.html>.

### 2.4.2 Pujant dades a un servidor

Per pujar dades a un servidor web heu de configurar la connexió per sortida amb `setDoOutput(true)`. Per millorar el rendiment de la vostra aplicació cal que crideu `setFixedLengthStreamingMode(contentLength)` quan conegueu per avançat la mida del cos de les dades que voleu enviar, o `setChunkedStreamingMode(chunkLength)` quan no la sabeu (té com a argument la mida de *chunk*, porció de dades, o 0 per fer servir la mida estàndard). Si no ho feu, `URLConnection` copiarà les dades completes a enviar en un *buffer* (memòria cau) abans d'enviar-les, desaprofitant així memòria i fent que l'aplicació vagi més lenta. Per exemple, el codi següent estableix una connexió a un servidor i realitza una descàrrega i una pujada de dades. Les funcions `EscriureFluxDeDades()` i `LlegirFluxDeDades()` seran funcions pròpies que treballaran amb l'`OutputStream` i l'`InputStream`.

```
1 HttpURLConnection urlConnection =(HttpURLConnection) url.openConnection();
2     try{
3         urlConnection.setDoOutput(true);
4         urlConnection.setChunkedStreamingMode(0);
5
6         OutputStream out=new BufferedOutputStream(urlConnection.getOutputStream());
7         EscriureFluxDeDades(out);
8
9         InputStream in=new BufferedInputStream(urlConnection.getInputStream());
10        LlegirFluxDeDades(in);
11    finally {
12        urlConnection.disconnect();
13    }
14 }
```

## 2.5 Missatgeria

Els serveis de missatgeria SMS (*Short Message Service*, servei de missatges curts) es troben entre els serveis de telefonia més populars. El seu cost reduït (en comparació amb les trucades) ha fet que hagi esdevingut la forma més comuna



per realitzar comunicacions senzilles durant anys. Darrerament, amb l'aparició dels *smartphones* i la seva connexió contínua a Internet, han aparegut serveis de missatgeria gratuïts a través de la xarxa de dades 3G (com Whatsapp), que han esdevingut una alternativa per a la comunicació de missatges entre dispositius que tenen connexió a Internet. En qualsevol cas, tots els telèfons permeten l'ús d'SMS, també els *smartphones*, tot i que aquest servei ha evolucionat cap a altres funcionalitats, com ara la validació del *login* en dues passes, la confirmació d'operacions bancàries, informacions del govern o d'empreses de subministraments...

Android ja porta inclosa per defecte una aplicació per enviar i rebre missatges SMS. Però potser voleu integrar les capacitats SMS dins de la vostra aplicació. Tot seguit teniu alguns exemples d'aplicacions Android que podrien fer servir el servei SMS:

- Una aplicació que envii un missatge de felicitació automàticament als nostres contactes el dia del seu aniversari.
- Una aplicació que envii un missatge quan algun valor de la borsa pugi o baixi d'un cert límit.
- Una aplicació que envii regularment la posició en la qual es troba el telèfon (per a persones amb alzheimer o nens petits).

Com veieu, les possibilitats són infinites i depèn únicament de la imaginació del desenvolupador.

### 2.5.1 Enviament d'SMS

Enviar missatges SMS des del vostre programa és molt senzill. En primer lloc, heu de declarar que la vostra aplicació tindrà permisos per enviar SMS. Per fer-ho, afegiu la següent declaració al fitxer de *manifest* del projecte:

```
1 <uses-permission android:name="android.permission.SEND_SMS">
2 </uses-permission>
```

El *layout* de l'activitat principal és molt senzill, amb unes caixes de text per introduir el número de telèfon i el text a enviar i un botó per enviar el missatge:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/RelativeLayout1"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     android:orientation="vertical">
7
8     <TextView
9         android:id="@+id/textView1"
10        android:layout_width="wrap_content"
11        android:layout_height="wrap_content"
12        android:layout_alignBaseline="@+id/editTelèfon"
13        android:layout_alignParentLeft="true"
14        android:layout_alignParentTop="true"
```

```
15     android:text="Telefon"
16     android:textAppearance="?android:attr/textAppearanceLarge" />
17
18     <EditText
19         android:id="@+id/editTelèfon"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:layout_alignParentRight="true"
23         android:layout_alignParentTop="true"
24         android:layout_toRightOf="@+id/textView1"
25         android:layout_weight="1"
26         android:ems="10"
27         android:inputType="phone">
28
29         <requestFocus/>
30     </EditText>
31
32     <TextView
33         android:id="@+id/textView2"
34         android:layout_width="wrap_content"
35         android:layout_height="wrap_content"
36         android:layout_alignParentLeft="true"
37         android:layout_below="@+id/editTelèfon"
38         android:text="Missatge"
39         android:textAppearance="?android:attr/textAppearanceLarge" />
40
41     <EditText
42         android:id="@+id/editMissatge"
43         android:layout_width="fill_parent"
44         android:layout_height="wrap_content"
45         android:layout_alignParentLeft="true"
46         android:layout_below="@+id/textView2" />
47
48     <Button
49         android:id="@+id/btnEnviarSMS"
50         android:layout_width="fill_parent"
51         android:layout_height="wrap_content"
52         android:layout_alignParentLeft="true"
53         android:layout_below="@+id/editMissatge"
54         android:text="Enviar SMS" />
55
56 </RelativeLayout>
```

Fixeu-vos que aquest *layout* està definit com un `RelativeLayout`. Podríeu obtenir un resultat similar amb un `LinearLayout`.

Les operacions amb el servei SMS estan gestionades mitjançant la classe `SMSManager`, que us permet enviar missatges de dades, text i missatges per parts. Per obtenir un objecte de la classe `SMSManager` heu de cridar el mètode `SMSManager.getDefault()`, per exemple:

```
1 SMSManager sms = SMSManager.getDefault();
```

Quan hàgiu aconseguit un objecte `SMSManager`, podeu fer servir el mètode `sendTextMessage`, que té els següents arguments:

- `String destinationAddress`: el número de telèfon on voleu enviar el missatge.
- `String scAddress`: el número del centre de servei d'SMS. Si introduïu *null*, es farà servir el que té definit el telèfon a la seva configuració.
- `String text`: text a enviar.

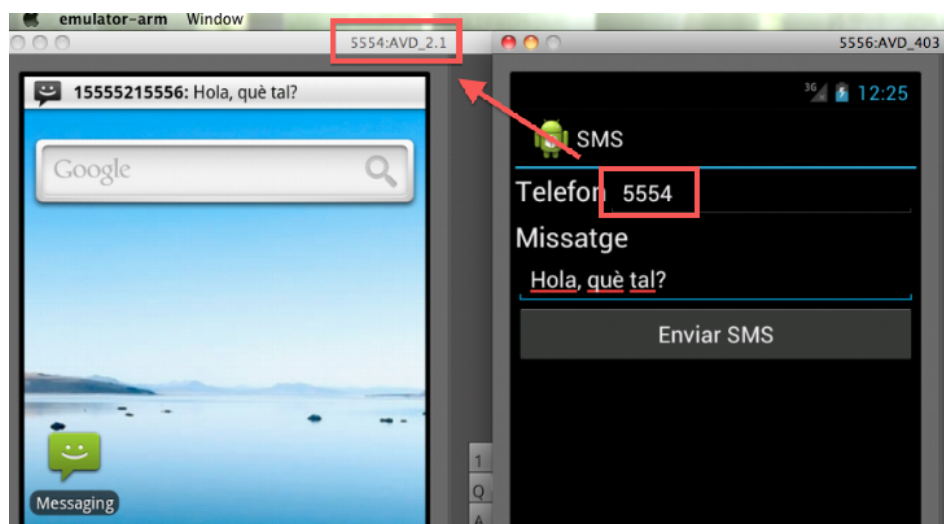
- `PendingIntent sentIntent`: si està a *null*, no fa res. Si li passeu un `PendingIntent`, aquest serà enviat com a missatge de *broadcast* al dispositiu quan el missatge hagi estat enviat, o bé quan falli l'enviament. En el segon cas, aquest `Intent` us permetrà saber el tipus d'error que s'ha produït.
- `PendingIntent deliveryIntent`: si és *null* no fa res. Si passeu un `Intent`, aquest serà enviat com a missatge de *broadcast* al dispositiu quan el missatge ha estat lliurat al destinatari.

La forma més fàcil d'enviar un missatge és simplement passar el número de telèfon i el missatge, com es veu a continuació:

```
1 sms.sendMessage(numeroTelefon, null, missatge, null, null);
```

Podeu fer una prova de funcionament des del simulador. Executeu dos simuladors des de l'AVD Manager i envieu un missatge SMS de l'un a l'altre. Per fer-ho, heu de fer servir el número de port de la instància de simulador Android (està a la capçalera de la finestra) com a número de telèfon, com es pot veure a la figura 2.2.

**FIGURA 2.2.** Enviant un missatge entre dues instàncies de l'emulador



## 2.5.2 Rebre SMS

Per rebre missatges dins de la vostra aplicació heu de fer servir un objecte `BroadcastReceiver`. Quan es rep un SMS al dispositiu, s'envia un missatge de *broadcast* anunciant la recepció. És aquest missatge el que podeu obtenir per fer alguna acció quan es rep un SMS.

Per rebre missatges SMS, en primer lloc heu de modificar el fitxer de *manifest* per indicar que l'aplicació té permisos per rebre SMS amb el codi. També heu de registrar al fitxer de *manifest* el receptor de *broadcast* que fareu servir:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```
3 package="com.android.ioc"
4 android:versionCode="1"
5 android:versionName="1.0">
6
7 <uses-sdk android:minSdkVersion="15"/>
8
9 <uses-permission android:name="android.permission.RECEIVE_SMS"/>
10
11 <application
12     android:icon="@drawable/ic_launcher"
13     android:label="@string/app_name">
14     <activity
15         android:name=".RebreSMSActivity"
16         android:label="@string/app_name">
17         <intent-filter>
18             <action android:name="android.intent.action.MAIN"/>
19             <category android:name="android.intent.category.LAUNCHER"/>
20         </intent-filter>
21     </activity>
22
23     <receiver android:name=".SMSReceiver">
24         <intent-filter>
25             <action android:name="android.provider.Telephony.SMS_RECEIVED"></action>
26         </intent-filter>
27     </receiver>
28 </application>
29
30 </manifest>
```

Cal que indiqueu el nom del receptor de *broadcast* i, dins del filtre d'*intents*, quin tipus de missatge de *broadcast* estareu escoltant (en aquest cas el missatge de rebuda d'SMS, SMS\_RECEIVED).

Dins de la carpeta */src* del projecte, afegiu una nova classe dins del paquet de la vostra aplicació i anomenau-la *SMSReceiver*. Aquesta classe heretarà de *BroadcastReceiver*. La classe *BroadcastReceiver* és la classe base que rebrà els *Intents* enviats mitjançant el mètode *sendBroadcast()*. La implementació del receptor ja ha estat anunciada a través del *manifest* de l'aplicació amb l'etiqueta *<receiver>*. Una altra forma de registrar el receptor de *broadcast* és mitjançant *Context.registerReceiver()*.

La classe que creeu, que hereta de *BroadcastReceiver*, ha d'implementar el mètode *onReceive(Context c, Intent i)*. Aquest és el mètode que és cridat quan el *BroadcastReceiver* rep un *Intent* de *broadcast*. Aquest mètode rep dos arguments: el context on el receptor s'està executant (on s'ha cridat) i l'*Intent* que s'està rebent.

L'objecte *BroadcastReceiver* únicament és vàlid durant la crida al mètode *onReceive()*. Una vegada que el codi retorna d'aquesta funció, el sistema considera l'objecte terminat i deixa d'estar actiu. Això té una repercussió important en el que es pot fer dins de la implementació del mètode *onReceive()*. Per exemple, no es poden fer coses que requereixen una operació asíncrona, ja que per fer-ho heu de sortir de la funció per manegar l'operació asíncrona, però en el moment en què se surt d'*onReceive()*, el *BroadcastReceiver* ja no està actiu i, per tant, el sistema és lliure de finalitzar el procés abans que l'operació asíncrona s'hagi completat.

Aquesta és la definició de la classe que fareu servir per escoltar els *Intents* de *broadcast*:

```
1 package com.android.ioc;
2
3 import android.content.BroadcastReceiver;
4 import android.content.Context;
5 import android.content.Intent;
6
7 public class SMSReceiver extends BroadcastReceiver {
8     public void onReceive(Context context, Intent intent) {
9
10    }
11 }
```

Quan es rep un missatge SMS, el mètode `onReceive()` és cridat. El missatge SMS està contingut dins de l'objecte `Intent` (el segon argument que rep `onReceive()`) a través dels extrems de l'`Intent`, que són un objecte `Bundle`. Reviseu la definició de la classe `Intent` a la *Guia del desenvolupador* d'Android per veure la seva definició i els extrems, a <http://developer.android.com/reference/android/content/Intent.html>.

Aquest és el codi que obté el `Bundle` de l'`Intent`.

```
1 //Obtenim els extrems de l'Intent
2 Bundle bundle = intent.getExtras();
3
4 //Si hi ha extrems
5 if(bundle !=null) {
6
7 }
```

Els missatges estan emmagatzemats en un *array* d'Objectes en el format PDU.

Per obtenir l'*array* d'objectes a partir del `Bundle` cal que feu servir:

```
1 //obtenir llistat d'SMS
2 Object[] pdus = (Object[]) bundle.get("pdus");
```

Per extraure cada missatge, heu de fer servir el mètode estàtic `SmsMessage.createFromPdu()`, que us retorna un `SmsMessage` a partir d'un PDU (dels que trobeu a l'*array* d'Objectes). El següent codi mostra com obtenir el missatge 0 (en forma d'`SmsMessage`) a partir de l'*array* d'objectes obtingut:

```
1 SmsMessage missatge = null;
2
3 missatge = SmsMessage.createFromPdu((byte[])pdus[0]);
```

Quan tingueu l'objecte `SmsMessage` podeu obtenir accés al cos de missatge o el número de telèfon que l'ha enviat amb `getMessageBody()` i `getOriginatingAddress()`, respectivament.

Finalment, el codi següent mostra tot el codi del mètode `onReceive()`. Penseu que, com que es pot haver rebut més d'un missatge, la lectura dels missatges de l'*array* d'objectes es fa dins d'un bucle *for*, per llegir tots els missatges si és el cas.

#### Unitat de Descripció de Protocol

PDU (*Protocol Description Unit*, unitat de descripció de protocol) és un dels mètodes definits per la indústria per enviar i rebre missatges SMS. No cal que feu una dissecció de la PDU, ja que la classe `SmsMessage` hi pot treballar directament. Per a més informació sobre l'estructura d'una PDU podeu consultar: [www.dreamfabric.com/sms](http://www.dreamfabric.com/sms).

```

1 public void onReceive(Context context, Intent intent) {
2
3     //Rebre l'SMS
4
5     //Obtenim els extrems de l'Intent
6     Bundle bundle = intent.getExtras();
7
8     SmsMessage[] missatges = null;
9     String s = "";
10
11     //Si hi ha extrems
12     if(bundle !=null) {
13         //obtenir llistat d'SMS
14         Object[] pdus = (Object[]) bundle.get("pdus");
15
16         SmsMessage missatge = null;
17
18         for(int i=0; i<pdus.length; i++) {
19             missatge = SmsMessage.createFromPdu((byte[])pdus[i]);
20             s += "SMS de " + missatge.getOriginatingAddress();
21             s += " :";
22             s += missatge.getMessageBody().toString();
23             s += "\n";
24         }
25
26         Toast.makeText(context, s, Toast.LENGTH_SHORT).show();
27     }
28
29 }

```

## 2.5.3 Enviant un SMS mitjançant un 'Intent'

Podeu fer servir la classe `SmsManager` per enviar missatges SMS des de la vostra aplicació. Però si no ho voleu programar, podeu invocar l'aplicació de missatgeria inclosa en el mateix sistema Android perquè envii el missatge per vosaltres.

Per activar l'aplicació de missatgeria des de la vostra aplicació podeu llançar un objecte *Intent* amb el tipus MIME “vnd.android-dir/mms-sms”. En aquest cas, l'únic que heu de fer és crear un objecte *Intent*, afegir les dades del missatge que voleu enviar com a extrems: “address” en el cas del número de telèfon (se'n pot introduir més d'un, separats per punts i coma) i “sms\_body” en el cas del cos del missatge. També heu d'introduir el tipus de l'*Intent* com “vnd.android-dir/mms-sms”. Una vegada fet això, simplement invoqueu `startActivity()` amb l'*Intent* creat, com es pot veure al següent codi:

```

1 Intent i = new Intent(android.content.Intent.ACTION_VIEW);
2 i.putExtra("address", editTelefon.getText().toString());
3 i.putExtra("sms_body", editMissatge.getText().toString());
4 i.setType("vnd.android-dir/mms-sms");
5 startActivity(i);

```

## 2.5.4 Seguretat

Quan feu servir els receptors de *broadcast* heu de pensar que aquests són una porta d'entrada a la vostra aplicació per a la resta d'aplicacions del dispositiu. Heu de considerar que altres aplicacions poden abusar de l'ús de missatges i, per tant, heu de tenir en compte alguns conceptes de seguretat per a la vostra aplicació:

- L'espai de noms d'*Intent* és global, per tant assegureu-vos que els noms d'acció dels *intents* estiguin escrits en un espai de noms vostre, o sense adonar-vos-en podríeu entrar en conflicte amb altres aplicacions.
- Quan registreu un receptor de *broadcast* amb `registerReceiver()`, qualsevol aplicació pot enviar *broadcasts* a la vostra. Podeu controlar qui pot enviar *broadcast* a través de permisos.
- Quan publiqueu un receptor al *manifest* de l'aplicació i especifiqueu filtres d'*Intent* per aquest, qualsevol altra aplicació pot enviar *broadcast*. Per evitar que altres aplicacions enviïn missatges de *broadcast* a la vostra, podeu fer-la “no disponible” per la resta amb `android:exported="false"` al fitxer de *manifest*.
- Quan feu servir `sendBroadcast(Intent)` o algun altre mètode similar, normalment qualsevol altra aplicació pot rebre aquests *broadcasts*. Podeu controlar qui rep els *broadcasts* mitjançant els permisos. A partir de la versió *Ice Cream Sandwich* d'Android podeu restringir el *broadcast* a una única aplicació.

Cap d'aquests problemes existeix quan es fa servir `LocalBroadcastManager()`, ja que els *intents broadcast* en aquest cas mai surten del procés de l'aplicació.

Es poden forçar els permisos en l'enviament i recepció *broadcast*. Per forçar els permisos quan s'envien missatges de *broadcast* compteu amb l'argument `String receiverPermission` dels mètodes `sendBroadcast()` i `sendOrderedBroadcast()`. Si no definiu aquest argument com a *null*, únicament els receptors als quals s'ha permès aquest permís (amb l'etiqueta de petició `<uses-permission>` al fitxer de *manifest*) podran rebre el *broadcast*.

Per establir els permisos sobre allò que pot rebre l'aplicació, heu d'introduir a l'argument `String broadcastPermission` un valor no *null* quan registreu el receptor amb el mètode `registerReceiver()`. També ho podeu fer estàticament dins de l'etiqueta `<receiver>` del fitxer de *manifest* de l'aplicació. Únicament els emissors de *broadcast* que tinguin aquest permís concedit (amb la petició amb l'etiqueta `<uses-permission>` al fitxer de *manifest*) podran enviar missatges a l'aplicació.

Podeu trobar més informació sobre seguretat a la *Guia del desenvolupador* d'Android, a <http://developer.android.com/guide/topics/security/permissions.html>.

### 2.5.5 Enviant un MMS mitjançant 'Intent'

Els MMS (*Multimedia Messaging System*, sistema de missatgeria multimèdia) són missatges de text semblant als SMS però amb un contingut extra de tipus multimèdia (foto, àudio, vídeo...). El procés d'enviament d'un missatge MMS és semblant al de l'SMS mitjançant un Intent, però en aquest cas s'han d'afegir una sèrie d'extres per indicar les dades multimèdia que es volen afegir al missatge. Per començar, l'Intent es crea amb el constructor d'Intent que té com a arguments una acció i un URI de les dades que es volen enviar:

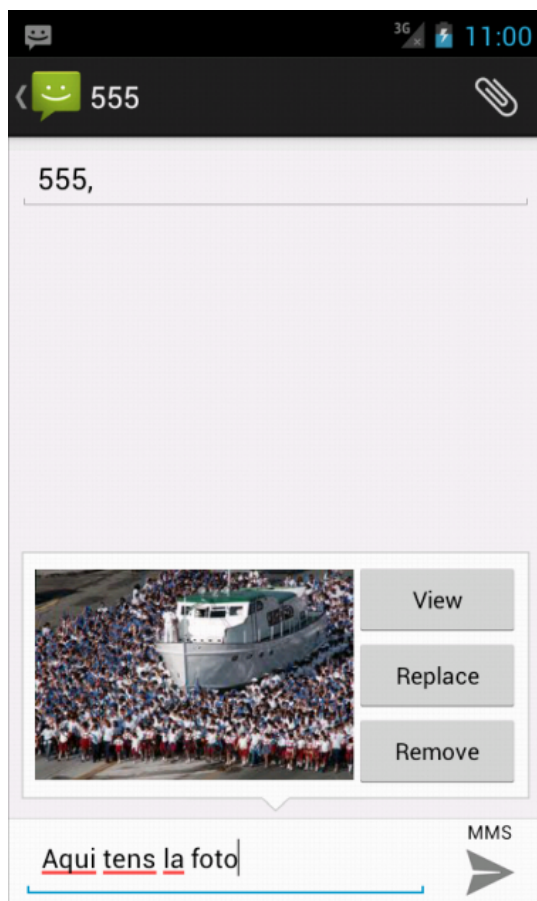
```
1 Intent intentMMS = new Intent(Intent.ACTION_SEND, Uri.parse("mms://"));
```

Després s'afegeixen els extres. A més del número de telèfon i el cos del missatge, afegiu el contingut multimèdia que es vol afegir al missatge. Per fer-ho agregueu un contingut extra del tipus que aneu a afegir. En aquest exemple de codi, s'inclou una imatge que es troba en la targeta SD del dispositiu.

```
1 intentMMS.putExtra("address", editTelefon.getText().toString());  
2  
3 intentMMS.putExtra("sms_body",  
4 editMissatge.getText().toString());  
5 String url = "file://mnt//sdcard//foto.jpeg";  
6  
7  
8 intentMMS.setType("image/jpeg");  
9  
10  
11 intentMMS.putExtra(Intent.EXTRA_STREAM, Uri.parse(url));  
12  
13 startActivity(Intent.createChooser(intentMMS, "MMS:"));
```

Quan s'executa l'aplicació, es fa una crida que es tractada per l'aplicació de missatgeria del sistema. Es fa servir aquesta aplicació per enviar un missatge MMS amb les dades que s'han passat a través de l'Intent, com es pot veure a la figura [2.3](#).



**FIGURA 2.3.** Enviament de missatgeria MMS

## 2.6 Analitzant codi XML

Un format molt útil per intercanviar informació entre dos dispositius és mitjançant XML. XML és un llenguatge de marques que defineix una sèrie de regles per codificar documents en un format que és comprensible, tant per les màquines (ordinadors, telèfons...) com per les persones. És un format que es fa servir freqüentment per compartir dades a Internet. Per exemple, els webs que tenen contingut que s'actualitzi freqüentment, com els blogs o pàgines de notícies, solen proporcionar codi XML per tal que programes externs puguin mostrar les actualitzacions de la pàgina. Per poder fer servir aquestes dades, les aplicacions s'han de descarregar les dades XML (per exemple, amb una connexió HTTP) i, posteriorment, analitzar el codi XML per obtenir les dades i tractar-les dins del programa com vulgui.

Existeixen diferents analitzadors d'XML que podeu fer servir a Android. Google recomana fer servir `XmlPullParser`, la documentació del qual podeu trobar a la *Guia del desenvolupador* d'Android:

<http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html>.

Tot seguit veurem un exemple d'utilització de l'analitzador a través d'un exemple. L'objectiu és descarregar les notícies del web `StackOverflow.com`. El *feed*

d’RSS d’aquest web sobre Android ordenat per les últimes entrades el podeu trobar a <http://stackoverflow.com/feeds/tag?tagnames=android&sort=newest> i té la següent forma:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <feed xmlns="http://www.w3.org/2005/Atom" xmlns:creativeCommons="http://backend
  .userland.com/creativecommonsRssModule" xmlns:re="http://purl.org/atompub/
    rank/1.0">
3 <title type="text">newest questions tagged android – Stack Overflow</title>
4 ...
5 <entry>
6 ...
7 </entry>
8 <entry>
9 <id>http://stackoverflow.com/q/11530594</id>
10 <re:rank scheme="http://stackoverflow.com">0</re:rank>
11 <title type="text">No puc compilar</title>
12 <category scheme="http://stackoverflow.com/feeds/tag?tagnames=android&
    sort=newest/tags" term="android"/><category scheme="http://
    stackoverflow.com/feeds/tag?tagnames=android&sort=newest/tags"
    term="android-emulator"/>
13 <author>
14 <name>aturing</name>
15 <uri>http://stackoverflow.com/users/803649</uri>
16 </author>
17 <link rel="alternate" href="http://stackoverflow.com/questions/11530594/
    launch-of-android-emulator-failing" />
18 <published>2012-07-17T20:44:06Z</published>
19 <updated>2012-07-17T20:47:59Z</updated>
20 <summary type="html">
21 No trobo el botó per compilar l’aplicació. Algú em pot ajudar?
22 </summary>
23 </entry>
24 <entry>
25 ...
26 </entry>
27 ...
28 </feed>

```

Si us hi fixeu, dins de les etiquetes `<entry>` i `</entry>` teniu tota la informació d’una entrada en el web.

Les entrades les emmagatzemem com una llista d’objectes de la classe `Entrada`, que conté tres *strings* per les diferents parts que us interessin de les notícies.

```

1 //Aquesta classe representa una entrada de notícia de l’RSS Feed
2 public static class Entrada {
3     public final String titol; //Títol de la notícia
4     public final String enllac; //Enllaç a la notícia completa
5     public final String resum; //Resum de la notícia
6
7     private Entrada(String title, String summary, String link) {
8         this.titol = title;
9         this.resum = summary;
10        this.enllac = link;
11    }
12 }

```

Tot seguit crearem una altra classe, `StackOverflowXmlParser`, per analitzar el contingut de l’XML. El mètode analitzat serveix per començar l’anàlisi de l’XML i rep un `InputStream` (que heu obtingut a través d’una connexió HTTP) com a entrada. El mètode retorna una llista d’objectes `Entrada`:

```
1 public List<Entrada> analitza(InputStream in) throws XmlPullParserException,
   IOException {
2     try {
3         //Obtenim analitzador
4         XmlPullParser parser = Xml.newPullParser();
5
6         //No fem servir namespaces
7         parser.setFeature(XmlPullParser.FEATURE_PROCESS_NAMESPACES, false);
8
9         //Especifica l'entrada de l'analitzador
10        parser.setInput(in, null);
11
12        //Obtenim la primera etiqueta
13        parser.nextTag();
14
15        //Retornem la llista de notícies
16        return llegirNotícies(parser);
17    } finally {
18        in.close();
19    }
20 }
```

L'objecte `XmlPullParser` l'obteniu amb una crida a `Xml.newPullParser()`. Una vegada especificades les seves característiques i el flux de dades d'entrada de l'analitzador crideu a `nextTag()` per obtenir el primer *tag* (etiqueta) del document XML. El mètode `next()` avança l'analitzador al següent *event* del document XML. Retorna un valor que indica el tipus d'*event* de l'estat de l'analitzador (el podeu obtenir també amb `getEventType()`). El mètode `nextTag()` ignora els espais en blanc i retorna la següent etiqueta de l'XML.

El mètode és l'encarregat de llegir totes les notícies i retornar el `List` d'entrades.

```
1 private List<Entrada> llegirNotícies(XmlPullParser parser) throws
   XmlPullParserException, IOException {
2
3     List <Entrada> llistaEntrades = new ArrayList<Entrada>();
4
5     //Comprova si l'event actual és del tipus esperat (START_TAG) i del nom "feed"
6     parser.require(XmlPullParser.START_TAG, ns, "feed");
7
8     //Mentre que no arribem al final d'etiqueta
9     while (parser.next() != XmlPullParser.END_TAG) {
10        //Ignorem tots els events que no siguin un començament d'etiqueta
11        if (parser.getEventType() != XmlPullParser.START_TAG) {
12            //Saltem al següent event
13            continue;
14        }
15
16        //Obtenim el nom de l'etiqueta
17        String name = parser.getName();
18
19        // Si aquesta etiqueta és una entrada de notícia
20        if (name.equals("entry")) {
21            //Afegim l'entrada a la llista
22            llistaEntrades.add(llegirEntrada(parser));
23        } else {
24            //Si és una altra cosa la saltam
25            saltar(parser);
26        }
27    }
28    return llistaEntrades;
29 }
```

El mètode `parser.require()` comprova si l'*event* actual és del tipus adequat i amb el nom que esperem, en aquest cas, "feed"; ho podeu comprovar en l'estructura del codi XML.

A continuació s'aniran llegint etiquetes dins del bucle *while* fins que es trobi el final d'etiqueta. S'ignoraran tots els *events* llegits que no siguin de començament d'etiqueta amb el codi:

```
1 //Ignora fins que no trobem un començament d'etiqueta
2 if (parser.getEventType() != XmlPullParser.START_TAG) {
3     continue;
4 }
```

El mètode `parser.getEventType()` retorna el tipus d'esdeveniment que ha llegit l'analitzador. Existeixen els següents tipus d'*events* definits a `XmlPullParser`:

- `START_TAG`: es llegeix un començament d'etiqueta XML.
- `TEXT`: es llegeix text. Es pot obtenir el seu contingut amb el mètode `parser.getText()`.
- `END_TAG`: es llegeix el final d'una etiqueta XML.
- `END_DOCUMENT`: no hi ha més *events* disponibles.

A continuació es cercarà entre les etiquetes fins que es trobi l'etiqueta *entry*. El mètode `parser.getName()` retorna el nom de l'etiqueta, es compara aquest nom per veure si és *entry* (fixeu-vos dins l'estructura del fitxer XML com les entrades estan contingudes dins d'una parella d'etiquetes `<entry> </entry>`).

Tota la resta d'etiquetes seràn ignorades. Una vegada trobada una secció *entry* de l'XML, es cridarà el mètode encarregat de llegir una entrada. El mètode `llegirEntrada()` retornarà un objecte `Entrada` que serà afegit al `List` retornat per `llegirNotícies()`.

```
1 //Analitza el contingut d'una entrada. Si troba un títol, resum o enllaç, crida
  els mètodes de lectura
2 //propis per processar-los. Si no, ignora l'etiqueta.
3
4 private Entrada llegirEntrada(XmlPullParser parser) throws
  XmlPullParserException, IOException {
5     String títol = null;
6     String resum = null;
7     String enllaç = null;
8
9     //L'etiqueta actual ha de ser "entry"
10    parser.require(XmlPullParser.START_TAG, ns, "entry");
11
12    //Mentre no acabi l'etiqueta d'"entry"
13    while (parser.next() != XmlPullParser.END_TAG) {
14        //Ignora fins que no trobem un començament d'etiqueta
15        if (parser.getEventType() != XmlPullParser.START_TAG) {
16            continue;
17        }
18
19        //Obtenim el nom de l'etiqueta
20        String etiqueta = parser.getName();
21
22        //Si és un títol de notícia
```

```

23     if (etiqueta.equals("title")) {
24         titol = llegirTitol(parser);
25
26         //Si l'etiqueta és un resum de notícia
27     } elseif (etiqueta.equals("summary")) {
28         resum = llegirResum(parser);
29
30         //Si és un enllaç
31     } elseif (etiqueta.equals("link")) {
32         enllac = llegirEnllac(parser);
33     } else {
34         //les altres etiquetes les saltem
35         saltar(parser);
36     }
37 }
38
39 //Creem una nova entrada amb aquestes dades i la retornem
40 return new Entrada(titol, resum, enllac);
41 }

```

El mètode `llegirEntrada()` comprova que està dins de l'etiqueta *entry* amb una crida a `parser.require()`. Posteriorment, comprova el nom de l'etiqueta que troba i es crida els mètodes `llegirTitol()`, `llegirResum()` i `llegirEnllac()` per llegir els diferents tipus de contingut de l'entrada. Tots ells retornen un `String` que fareu servir finalment en la crida al constructor d'`Entrada`.

```

1 //Llegeix el títol d'una notícia del feed i el retorna com String
2 private String llegirTitol(XmlPullParser parser) throws IOException,
3     XmlPullParserException {
4     //L'etiqueta actual ha de ser "title"
5     parser.require(XmlPullParser.START_TAG, ns, "title");
6
7     //Llegeix
8     String titol = llegeixText(parser);
9
10    //Fi d'etiqueta
11    parser.require(XmlPullParser.END_TAG, ns, "title");
12    return titol;
13 }
14
15 //Llegeix l'enllaç d'una notícia del feed i el retorna com String
16 private String llegirEnllac(XmlPullParser parser) throws IOException,
17     XmlPullParserException {
18     String enllac = "";
19
20     //L'etiqueta actual ha de ser "link"
21     parser.require(XmlPullParser.START_TAG, ns, "link");
22
23     //Obtenim l'etiqueta
24     String tag = parser.getName();
25
26     //Obtenim l'atribut rel (mirar l'XML d'Stackoverflow)
27     String relType = parser.getAttributeValue(null, "rel");
28
29     //Si l'enllaç és link
30
31     if (tag.equals("link")) {
32         //Obtenim l'enllaç del valor de l'atribut "href". Revisar format XML
33         //stackoverflow
34         if (relType.equals("alternate")) {
35             enllac = parser.getAttributeValue(null, "href");
36             parser.nextTag();
37         }
38     }
39
40     //Fi d'etiqueta
41     parser.require(XmlPullParser.END_TAG, ns, "link");

```

```

39
40     return enllac;
41 }
42
43 //Llegeix el resum d'una notícia del feed i el retorna com String
44 private String llegirResum(XmlPullParser parser) throws IOException,
45     XmlPullParserException {
46
47     //L'etiqueta actual ha de ser "summary"
48     parser.require(XmlPullParser.START_TAG, ns, "summary");
49
50     String resum = llegeixText(parser);
51
52     parser.require(XmlPullParser.END_TAG, ns, "summary");
53     return resum;
54 }
55
56 //Extrau el valor de text per les etiquetes títol, resum
57 private String llegeixText(XmlPullParser parser) throws IOException,
58     XmlPullParserException {
59
60     String resultat = "";
61
62     if (parser.next() == XmlPullParser.TEXT) {
63         resultat = parser.getText();
64         parser.nextTag();
65     }
66
67     return resultat;
68 }

```

En aquests mètodes, el que es fa és llegir el contingut de l'XML i retornar un String. El mètode per obtenir l'enllaç és una mica diferent, ja que els enllaços en aquest XML no vénen dins d'una etiqueta, sinó com un atribut de la mateixa, tal com es pot observar en aquesta secció d'XML:

```

1 <link rel="alternate" href="http://stackoverflow.com/questions/tagged/?tagnames
  =xml&sort=newest" type="text/html" />

```

Per obtenir el valor de l'atribut heu de fer servir el mètode `parser.getAttributeValue()`.

Així doncs, l'analitzador anirà recorrent el fitxer XML tot analitzant les etiquetes i per cada *entry* crearà un objecte de la classe *Entrada* que quedarà emmagatzemat a un List *<Entrada>*, que serà retornat. Quan tinguem tota la informació de les entrades, aquestes s'hauran de mostrar en la nostra aplicació. Existeixen diferents formes de fer-ho: amb objectes creats dinàmicament, amb *ListView*... la que durem a terme nosaltres consisteix a crear un codi HTML a partir de la informació, que mostrarem en un *WebView*.

```

1 ...
2 //Llista de d'entrades de notícies
3 List<Entrada> entrades = null;
4
5 //Cadena on construirem el codi HTML que mostrarà el widget webView
6 StringBuilder htmlString = new StringBuilder();
7
8 ...
9
10 entrades = analitzador.analitza(stream);
11
12 ...

```

```
13
14 //analitzador.parse() retorna una llista (entrades) d'entrades de notícies (
    objectes
15 //de la classe Entrada). Cada objecte representa un post de l'XML Feed. Ara es
    processen
16 //les entrades de la llista per crear un codi HTML. Per cada entrada es crea un
    enllaç a la notícia completa
17
18 //Si tenim notícies
19 if(entrades != null) {
20
21
22 //Creem l'HTML a partir dels continguts del List<Entrada>
23
24 //Per indicar quan s'ha actualitzat l'RSS
25
26 Calendar ara = Calendar.getInstance();
27 DateFormat formatData = new SimpleDateFormat("dd MMM h:mmaa");
28
29 //Títol de la pàgina
30 htmlString.append("<h3> Notícies </h3>");
31
32 //Data d'actualització
33 htmlString.append("<em>Actualitzat el " + formatData.format(ara.getTime()) + "
    </em>");
34
35 //Per cada notícia de la llista
36 for (Entrada noticia : entrades) {
37 //Creem un títol de la notícia que serà un enllaç HTML a la notícia completa
38 htmlString.append("<p> <a href='");
39 htmlString.append(noticia.enllaç);
40 htmlString.append("'>" + noticia.títol + "</a>");
41
42 //Si la notícia té un resum, l'afegim (els primers 70 chars)
43 String breu = noticia.resum.substring(0, 70);
44
45 htmlString.append("<br><i>Resum:</i>" + breu + "...");
46 htmlString.append("</p> <hr>");
47 }
48 }
49 ...
50 setContentView(R.layout.main);
51
52 //Mostra la cadena HTML a l'UI a través del WebView
53 WebView myWebView = (WebView) findViewById(R.id.webView1);
54
55 myWebView.loadData(htmlString.toString(), "text/html", null);
```

Es van llegint totes les entrades del List i generant el codi HTML per tal que es mostri a l'aplicació tal com es pot veure a la figura [2.4](#).

Podeu descarregar el codi de la solució en el document XMLParse.zip, que trobareu en la secció "Annexos" del web del mòdul.

**FIGURA 2.4.** Lector d'RSS en funcionament

