

U.D.4 Activitats i Intencions

- 4.1. Crear noves activitats.
 - 4.1.1. Cicle de vida d'una activitat
- 4.2. Comunicació entre activitats.
- 4.3. Afegir un menú.
- 4.4. Icones.
- 4.5. Preferències d'usuari: Organitzar preferències, emmagatzemar preferències d'usuari, accedir als valors de les preferències.
- 4.6. Introduir llista de puntuacions al nostre joc.
- 4.7. La vista ListView.
- 4.8. Les intencions

4.1 Crear noves activitats.

<https://developer.android.com/guide/components/activities/intro-activities?hl=es>

El concepte d'activitat en Android representa una unitat d'interacció amb l'usuari. Correspon al que col·loquialment anomenem una pantalla de l'aplicació. Una aplicació sol estar formada per un seguit d'activitats, de manera que l'usuari pot anar navegant entre activitats.

Tota activitat ha de tenir una vista associada, que serà utilitzada com a interfície d'usuari. Aquesta vista sol ser de tipus layout, dins un arxiu .xml, encara que també pot ser una vista simple.

Una aplicació estarà formada per un conjunt d'activitats independents; és a dir, es tracta de classes independents que no comparteixen variables, encara que totes treballen per un objectiu comú. Un altre aspecte important és que tota activitat ha de ser una subclasse de *Activity* o millor *AppCompatActivity*.

A mesura que la nostra aplicació creixi serà imprescindible crear noves activitats. Aquest procés es pot resumir en quatre passos:

- Crear un nou Layout per a l'activitat. Es pot fer amb el botó dret damunt la carpeta layout, New / Layout Resource File.
- Crear una nova classe descendent de *AppCompatActivity*. En aquesta classe hauràs d'indicar que el layout a visualitzar és el desenvolupat en el punt anterior. Botó dret damunt la carpeta java, New / Java class.

```
public class SecondActivity extends AppCompatActivity {  
  
    @Override public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_second);  
    }  
}
```

- Perquè la nostra aplicació sigui visible serà necessari activar-la des d'una altra activitat. Per exemple al fer click sobre un botó implementant el mètode *onClick* dins el fitxer .xml que defineix el layout:

```
android:onClick="runSecondActivity"
```

- Dins el mètode *runSecondActivity*, que hem d'implementar dins la classe principal (*MainActivity*), s'ha de crear un objecte ***Intent*** que necessita com a primer paràmetre un *Context*, que pot ser *this* perquè estem dins una *Activity*, que ja conté un *Context*; i com a segon paràmetre el nom de la *Activity* que volem arrancar.

```
public void runSecondActivity(View view){
    Intent i = new Intent(this, SecondActivity.class);
    startActivity(i);
}
```

- De forma obligatòria **haurem de registrar tota nova activitat a *AndroidManifest.xml***

```
<activity
    android:name=".SecondActivity"
    android:exported="true" />
```

Cal indicar que totes aquestes passes per crear una nova activitat es fan de manera més automàtica si escollim l'opció *File / New / Activity*.

Alternativa: Un escoltador d'esdeveniment (listener) per codi

Com acabem de veure en un layout podem definir l'atribut XML *onClick* que ens permet indicar un mètode que serà executat al fer clic en una vista. A aquest mètode se'l coneix com escoltador d'esdeveniment. Resulta molt senzill i a més està disponible en qualsevol descendent de la classe *View*. No obstant això, aquesta tècnica presenta dos inconvenients: no es pot usar amb *Fragments* i només està disponible per a l'esdeveniment *onClick ()*. La classe *View* té altres esdeveniments (*onLongClick ()*, *onFocusChange ()*, *onKey ()*, ...) pels quals no s'han definit un atribut xml. Llavors, què fem si volem definir un esdeveniment diferent de *onClick ()*? La resposta la trobaràs aquest exercici:

Pràctica: *OnClickListener*

1. Obre la classe *MainActivity*, i copia aquest contingut:

```
public class MainActivity extends Activity {
    private Button bSecondAct;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        bSecondAct = findViewById(R.id.btnSecond);
        bSecondAct.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                runSecondActivity(null);
            }
        });
    }
}
```

2. Elimina l'atribut afegit al botó:

```
android:onClick="runSecondActivity"
```

3. Executa l'aplicació. El resultat ha de ser idèntic a l'anterior.

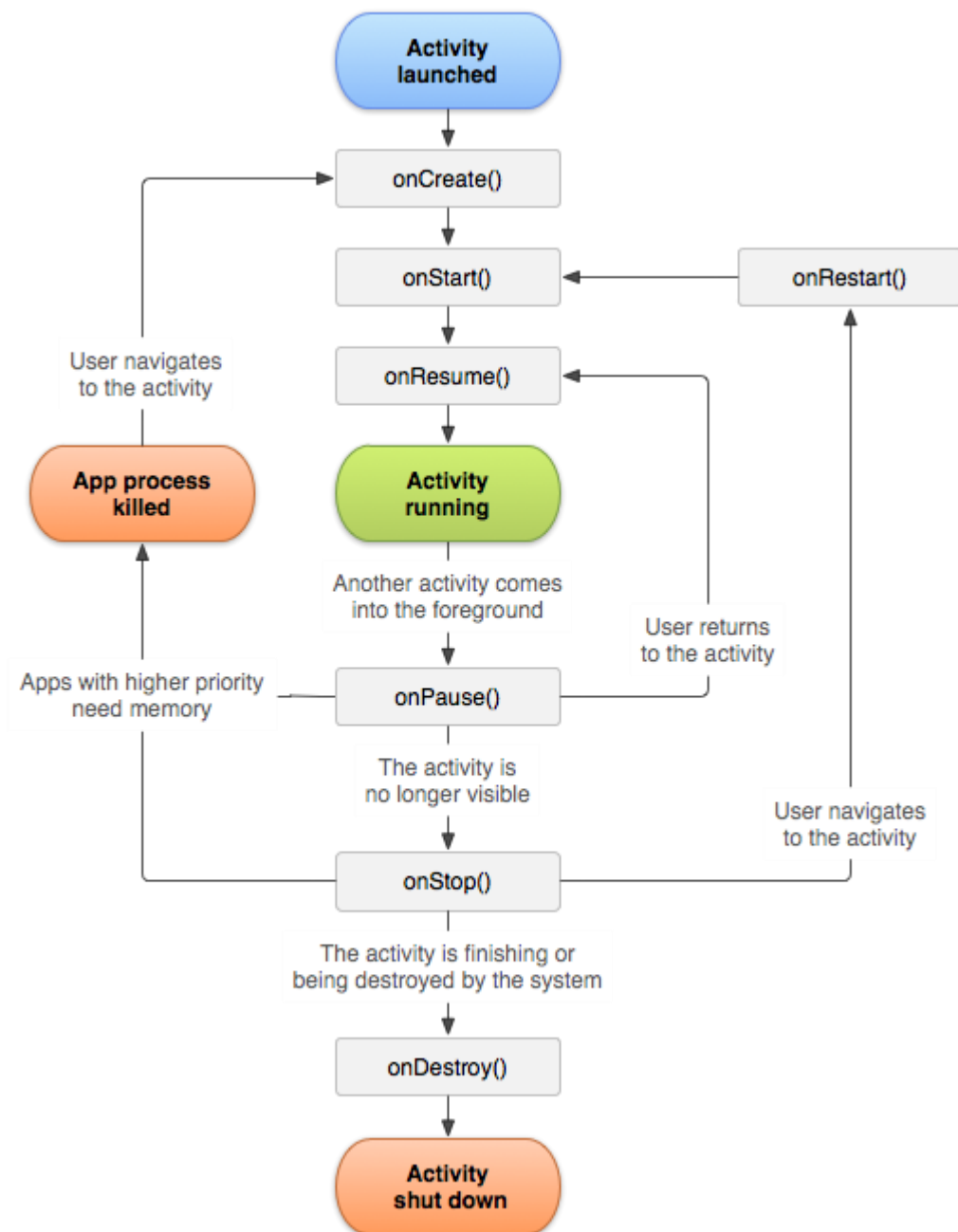
Exercici 4.1: Implementar el botó Sobre i el botó Sortir de l'aplicació Asteroides.

- El botó Sobre ha de mostrar un quadre de text amb la informació referent al nom del creador de l'aplicació, l'any de creació a altra informació que vulgueu mostrar. Se li pot donar un format adequat assignant-li el tema *Theme.AppCompat.Light.Dialog*
- El botó sortir ha d'executar un mètode que cridi a la instrucció `finish()`

4.1.1 Cicle de vida d'una activitat

<https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es>

Quan un usuari navega per la teva app, surt d'ella i torna a entrar, les instàncies de Activity de la teva app passen per diferents estats del seu cicle de vida. La classe Activity proporciona una sèrie de callbacks que permeten a l'activitat saber que ha canviat d'estat, és a dir, que el sistema està creant (`onCreate`), detenint (`onPause`) o reprenent (`onRestart`) una activitat, o finalitzant (`onDestroy`) el procés en què es troba.



Per veure en quin moment s'activa cadascuna de les etapes del cicle de vida es poden implementar dins una activitat tots els mètodes que es criden quan una activitat passa d'una a una altra fase.

Mitjançant un objecte tipus Log podem fer que surti un missatge a la pantalla de Logcat per seguir els canvis d'estat de l'activitat.

Pràctica: Cicle de vida: Copia aquest codi i comprova com apareixen els missatges de Log a la consola a mesura que l'activitat canvia d'estat.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i("Cicle", "Dins onCreate");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.i("Cicle", "Dins onStart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.i("Cicle", "Dins onResume");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.i("Cicle", "Dins onPause");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.i("Cicle", "Dins onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i("Cicle", "Dins onDestroy");
    }

}
```

4.2 Comunicació entre activitats.

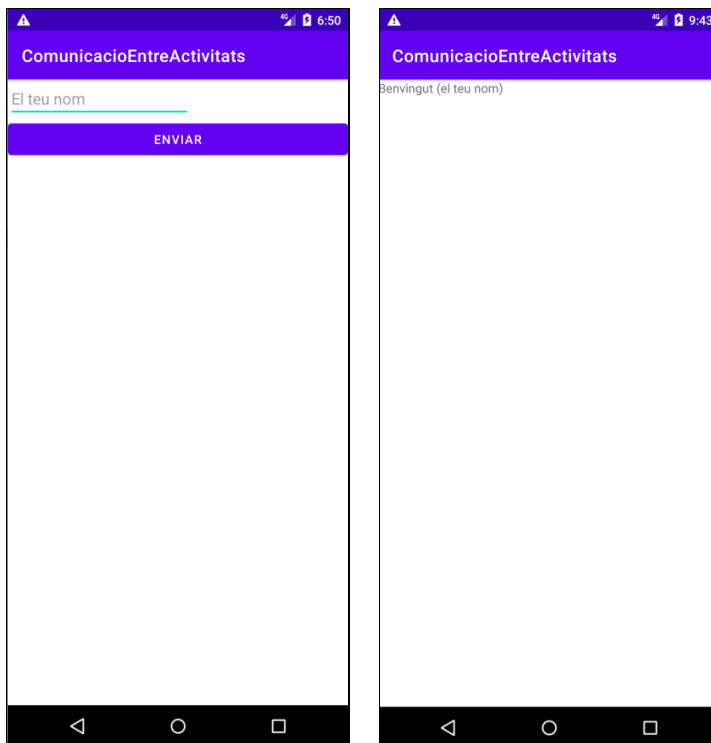
Quan una activitat ha de llançar una altra activitat, en molts casos necessita enviar-li certa informació. Android ens permet aquest intercanvi de dades utilitzant el mètode de la classe Intent: `putExtra()`:

```
Intent intent = new Intent(this, MyClass.class);
intent.putExtra("user", "Antoni Perez");
intent.putExtra("age", 27);
startActivity(intent);
```

En l'activitat llançada (B) podem recollir les dades de la següent manera:

```
Bundle extras = getIntent().getExtras();
String s = extras.getString("user");
int i = extras.getInt("age");
```

Exemple: Es poden implementar les dues activitats següents i passar com a extra el text que l'usuari introdueix dins el camp de text.



MainActivity

```
package com.example.comunicacioentreactivitats;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView txtNom;
    String nom;

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

    public void enviarInformacio(View view) {
        txtNom = (TextView) findViewById(R.id.edtNom);
        nom = txtNom.getText().toString();
        Intent i = new Intent(this, SegonaActivity.class);
        i.putExtra("nom", nom);
        startActivity(i);
    }
}
```

SegonaActivity

```
package com.example.comunicacioentreactivitats;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class SegonaActivity extends AppCompatActivity {

    TextView receptor;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_segona);

        receptor = findViewById(R.id.txtReceptor);
        String nom;
        Intent i = getIntent();
        nom = i.getStringExtra("nom");
        receptor.append(nom);
    }
}
```

```
}  
}
```

Comunicació entre activitats amb retorn de dades.

Quan l'activitat llançada (B) acaba, si ho vol, podrà enviar dades de retorn. Per això, es crearà un nou Intent sense paràmetres, s'hi afegiran les dades a retornar mitjançant *putExtra* i es cridarà al mètode *setResult* amb un codi de RESULT_OK i l'intent amb les dades a retornar.

```
Intent intent = new Intent();  
intent.putExtra("resultat", "valor");  
setResult(RESULT_OK, intent);  
finish();
```

Per poder rebre aquestes dades, des de l'activitat (A), s'haurà de llançar l'activitat (B) amb *startActivityForResult (Intent, int)* enlloc de *startActivity(Intent)*, on el segon paràmetre és un enter amb un codi que identifica l'activitat que llancem (per exemple, 1234).

```
startActivityForResult(intent, 1234);
```

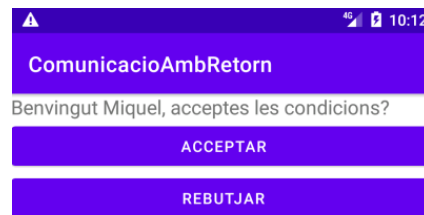
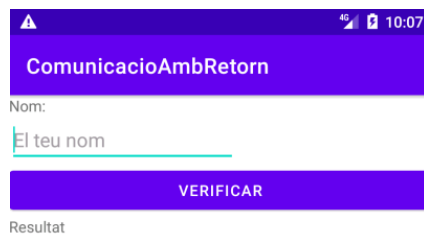
A més, per tractar les dades retornades dins l'activitat (A), haurem de sobreescriure el mètode *onActivityResult (int, int, Intent)*, on en el primer paràmetre es retorna el mateix codi que hem indicat al fer la cridada (1234); el segon, si el resultat ha estat ok (RESULT_OK) o cancel·lat (RESULT_CANCEL) i el tercer, un Intent on s'inclouen les variables retornades mitjançant els extrems (data):

```
@Override protected void onActivityResult(int requestCode, int resultCode, Intent data){  
    super.onActivityResult(requestCode, resultCode, data);  
    if (requestCode==1234 && resultCode==RESULT_OK) {  
        String res = data.getStringExtra("resultat");  
    }  
}
```

Des de l'activitat A es podrien cridar a diverses activitats, no obstant això, només podem tenir un mètode *onActivityResult ()*. Per aquesta raó, resulta necessari identificar cada activitat llançada amb un codi. Així, podrem diferenciar dins el mètode *onActivityResult* entre les diferents dades retornades.

Exercici 4.2: Comunicació entre activitats amb retorn de dades.

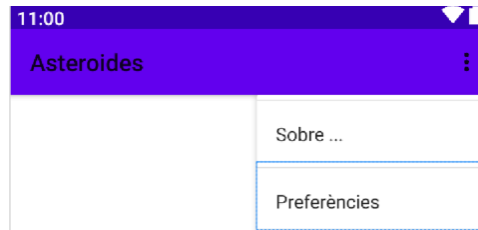
1. Crea un nou projecte amb nom *ComunicacioActivitats* i tipus *Empty Views Activity*.
2. El layout de l'activitat inicial ha de ser semblant al què es mostra a sota.
3. Introdueix el codi perquè quan es premi el botó "Verificar" s'arranqui una segona activitat. A aquesta activitat se li passarà com a paràmetre el nom introduït en el EditText.
4. A l'arrencar l'activitat el text del primer TextView ha de modificar-se perquè posi "Hola" + nom rebut + "Acceptes les condicions?"
5. En aquesta activitat es podran prémer dos botons, de manera que es retorni a l'activitat principal l'String "Acceptat" o "Rebutjat", segons el botó premut. Al prémer qualsevol botó es tornarà a l'activitat anterior.
6. En l'activitat principal es modificarà el text de l'últim TextView perquè posi "Resultat: Acceptat" o "Resultat: Rebutjat", segons el rebut.



4.3 Afegir un menú.

Podem assignar un menú a la nostra activitat de forma molt senzilla.

1. Crear un nou menú, File> New> Android resource file. En el camp *File name*: selecciona *menu_main* i en el camp *Resource type*: selecciona *Menú*.



2. Les opcions de menú es defineixen dins el fitxer *menu_main.xml* de la següent manera:

```
<item android:id="@+id/preferences"
    android:title="Preferències"
    android:icon="@android:drawable/ic_menu_preferences"
    android:orderInCategory="100"
    app:showAsAction="never"/>
<item android:title="Sobre..."
    android:id="@+id/about"
    android:icon="@android:drawable/ic_menu_info_details"/>
```

3. Perquè el menú es mostri dins l'activitat s'ha d'implementar el mètode *onOptionsItemSelected* i dins hem de fer servir el mètode *inflate* de la classe *MenuInflater* de la següent manera:

```
public boolean onOptionsItemSelected(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

4. Per definir quines acció realitzarà cada opció de menú, implementam el mètode *onOptionsItemSelected*:

```
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();
    if (id==R.id.preferences){
        //arrancar activitat preferències
    }

    if (id == R.id.about){
        //arrancar activitat sobre...
    }

    return super.onOptionsItemSelected(item);
}
```

Exercici 4.3: Incloure un menú a la vostra aplicació Asteroides

4.4 Icones.

En l'apartat anterior hem creat una barra d'accions on es mostraven algunes icones. S'han utilitzat icones disponibles en el sistema Android, és a dir, recursos ja emmagatzemats en el sistema. Una altra alternativa és crear les teves pròpies icones i emmagatzemar-les en la carpeta `res / mipmap`.

En Android s'utilitzen diferents tipus d'icones segons la seva utilitat. La següent taula mostra els més importants:

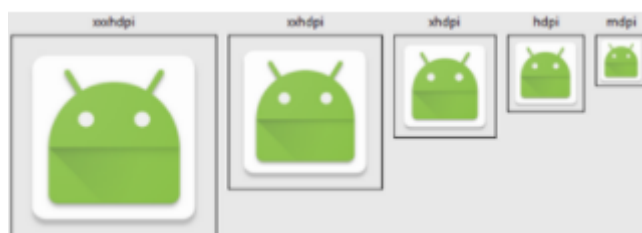
Tipus d'icona	Finalitat	Exemples
Llançadors	Representa l'aplicació a la pantalla principal del dispositiu	
Barra d'accions	Opcions disponibles en la barra d'accions	
Notificacions	Petites icones que apareixen en la barra d'estat	
Altres	També és molt freqüent l'ús d'icones en quadres de diàleg i en RecyclerView, etc.	

La icona més important d'una aplicació és sens dubte la icona llançador o *launcher*. S'utilitzarà per identificar la teva aplicació, tant en el sistema Android com a la botiga d'aplicacions. La icona a utilitzar ha d'indicar-se en un atribut de *AndroidManifest*.

Icones n'hi ha de 3 tipus: Legacy icons (heretats), circulars (round) i adaptatius (adaptive)

Legacy icons

```
<application
    android:icon="@mipmap/ic_launcher"/>
```



Round icons

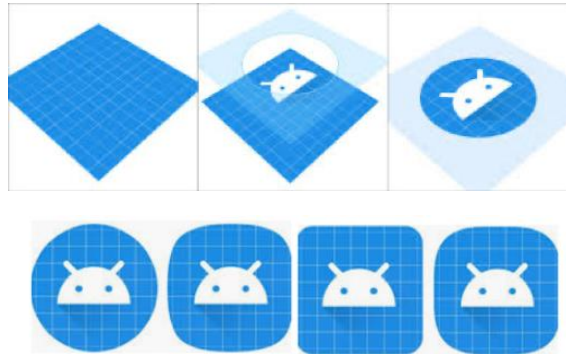
Apareixen en la versió 7.1 (API 25)

```
<application
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"... />
```

Adaptive icons

Apareixen en la versió 8.0 (API 26). Es defineixen en un fitxer xml de la següent manera:

```
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android"
id">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

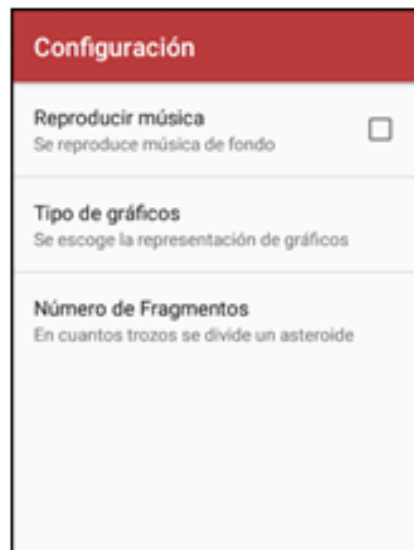


Enllaços d'interès :

- Guia d'estil per a icones: La següent pàgina descriu les guies d'estil per les icones en Material Design: <https://material.io/design/iconography>
- Recursos d'icones: A les següents pàgines pots trobar gran varietat d'icones: <https://material.io/tools/icons>

4.5 Preferències d'usuari: Organitzar preferències, emmagatzemar preferències d'usuari, accedir als valors de les preferències.

Android ens facilita la configuració dels nostres programes al permetre afegir una llista de preferències que l'usuari pot modificar. Per exemple, l'usuari podrà indicar amb quina freqüència l'aplicació ha de sincronitzar-se amb el servidor o si volem que es llancin notificacions. Les preferències també poden utilitzar-se perquè la teva aplicació emmagatzemi informació de forma permanent.



1. Obre el projecte Asteroides.
2. Prem amb el botó dret sobre la carpeta res i selecciona l'opció New> Android resource file.
3. Completa els camps File name: preferences i Resource type: XML. Es crearà el fitxer res / xml / preferences.xml.
4. Edita el fitxer i introdueix el següent codi:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:key="preferencias_principal" >
    <CheckBoxPreference
        android:key="musica"
        android:title="Reproducir música"
        android:summary="Se reproduce música de fons"/>
    <ListPreference
        android:key="grafics"
        android:title="Tipus de gràfics"
        android:summary="Escollir la representació de gràfics"
        android:entries="@array/tipusGrafics"
        android:entryValues="@array/tipusGraficsValors"
        android:defaultValue="1"/>
    <EditTextPreference
```

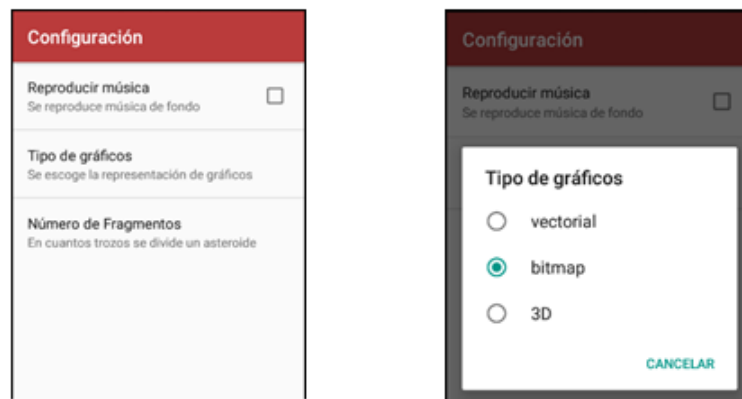
```

        android:key="fragments"
        android:title="Número de fragments"
        android:summary="En quants trossos es divideix un
asteroide"
        android:inputType="number"
        android:defaultValue="3"/>
</PreferenceScreen>

```

El significat de cada etiqueta i atribut es descobreix fàcilment si observes el resultat obtingut que es mostra a continuació. L'atribut *inputType* permet configurar el tipus de teclat que es mostrarà per introduir el valor. Coincideixen amb l'atribut de *EditText*. Per veure els possibles valors consultar

developer.android.com/reference/android/widget/TextView.html#attr_android:inputType .



- Per emmagatzemar els valors del desplegable, has de crear el fitxer */res/values/arrays.xml* amb el següent contingut. Per a això prem amb el botó dret sobre la carpeta *res* i selecciona l'opció *New> Android resource file*. Completa els camps *File name: arrays* i *Resource type: values*.

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="tipusGraphics">
        <item>vectorial</item>
        <item>bitmap</item>
        <item>3D</item>
    </string-array>
    <string-array name="tipusGraphicsValors">
        <item>0</item>
        <item>1</item>
        <item>2</item>
    </string-array>
</resources>

```

- Crea ara una nova classe *PreferencesActivity.java* amb el següent codi:

```

public class PreferencesActivity extends PreferenceActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);
    }
}

```

NOTA: Des del nivell d'API 11, el mètode `addPreferencesFromResource ()` s'ha marcat com a obsolet. En lloc d'utilitzar la classe `PreferenceActivity`, es recomana utilitzar `PreferenceFragment`. Quan un mètode o classe està marcat com obsolet, se'ns indica que hi ha una altra forma més moderna de fer-ho, fins i tot que en un futur és possible que deixi d'estar inclòs en l'API. Però això és una cosa que mai ha passat. En Android és freqüent seguir utilitzant mètodes marcats com obsolets. En el següent apartat s'explica l'ús de `PreferenceFragment`.

7. Cal no oblidar registrar tota nova activitat a `AndroidManifest.xml`.
8. Afegeix a `MainActivity.java` el mètode `launchPreferences()`. Aquest mètode ha de tenir el mateix codi que `launchAbout()` però llançant l'activitat `Preferences`. Al `Layout activity_main.xml` s'afegeix al botó amb text "Configurar" en l'atribut `onClick` el valor `launchPreferences`.
9. Per activar la configuració des de l'opció de menú afegeix el següent codi al fitxer `MainActivity.java` en el mètode `onOptionsItemSelected ()`

```
if (id == R.id.preferences) {  
    launchPreferences(null);  
    return true;  
}
```

10. Arrenca l'aplicació i verifica que pots llançar les preferències mitjançant les dues alternatives.

Utilitzant PreferenceFragment (opcional)

Quan la teva aplicació utilitzi un nivell d'API 11 o superior (versió 3.0), pots utilitzar `PreferenceFragment` en lloc de `PreferenceActivity` per visualitzar la llista de preferències. L'avantatge de `PreferenceFragment` és que mostra les preferències en un fragment en lloc d'una activitat, de manera que podràs visualitzar simultàniament en pantalla un fragment amb les preferències al costat d'un altre fragment. Això tindria sentit en una tauleta, però no seria interessant en un mòbil. En una primera presa de contacte amb Android no recomanem utilitzar fragments. Si és el teu cas, et recomanem que passis a l'apartat següent. Si prefereixes utilitzar `PreferenceFragment` pots fer el següent exercici.

Exercici (opcional): Afegint preferències a amb `PreferenceFragment`.

1. Crearem un fragment que contingui les preferències. Per a això s'afegeix la següent classe en el paquet de l'aplicació:

```
public class PreferencesFragment extends PreferenceFragment {  
  
    @Override  
  
    public void onCreate(Bundle savedInstanceState) {  
  
        super.onCreate(savedInstanceState);  
  
        addPreferencesFromResource(R.xml.preferences);  
  
    }  
  
}
```

Com pots veure el codi és idèntic a l'utilitzat a *PreferenciesActivity*.

3. Ara des de l'Activitat Preferències mostrarem aquest fragment.

```
public class PreferencesActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        getFragmentManager().beginTransaction()  
            .replace(android.R.id.content, new  
PreferencesFragment())  
            .commit();  
    }  
}
```

Des d'una activitat podem visualitzar un fragment en temps d'execució. Per a això utilitzem el gestor de fragments de la activitat (*getFragmentManager()*) i comencem una transacció (*beginTransaction()*). Una transacció és una operació d'inserció, esborrat o reemplaçament de fragments. En l'exemple anem a reemplaçar el contingut de l'activitat per un nou fragment de la classe *PreferenciesFragment*. Finalment es diu a *commit()* perquè s'executi la transacció.

4. Executa l'aplicació i verifica que el resultat és idèntic a l'anterior.

Organitzant preferències

Quan el nombre de preferències és gran, resulta interessant organitzar-les de manera adequada. Una possibilitat consisteix a dividir-les en diverses pantalles, de manera que quan es seleccioni una opció en la primera pantalla, s'obri una nova pantalla de preferències. Per organitzar les preferències d'aquesta manera, fa servir el següent esquema:

```
<PreferenceScreen>  
    <CheckBoxPreference .../>  
    <EditTextPreference .../>  
    ...  
    <PreferenceScreen android:title="Mode multijugador">  
        <CheckBoxPreference .../>  
        <EditTextPreference .../>  
        ...  
    </PreferenceScreen>  
</PreferenceScreen>
```

Pràctica: Organitzant preferències (I)

1. Crea una nova llista de preferències *<PreferenceScreen>* dins de la llista de preferències de el fitxer *res / xml / preferences.xml*.
2. Assigna-li a el paràmetre *android: title* el valor "Mode multijugador".
3. Crea tres elements dins d'aquesta llista: Activa multijugador, Màxim de jugadors i Tipus de connexió. Per a aquest últim han de poder escollir els valors: Bluetooth, Wi-Fi i Internet.

Una altra alternativa per a organitzar les preferències consisteix a agrupar-les per categories. Amb aquesta opció es visualitzaran a la mateixa pantalla, però separades per grups. Has de seguir el següent esquema:

```
<PreferenceScreen>  
    <CheckBoxPreference .../>  
    <EditTextPreference .../>  
    ...
```



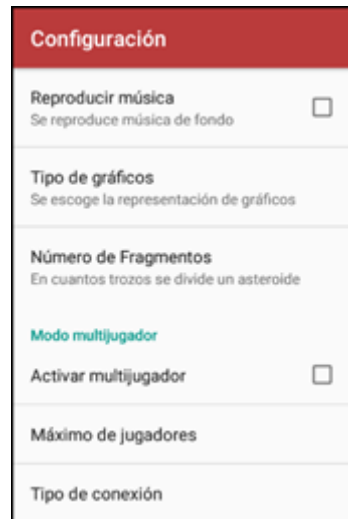
```

<PreferenceCategory android:title="Mode multijugador">
    <CheckBoxPreference .../>
    <EditTextPreference .../>

    ...
</PreferenceCategory>
</PreferenceScreen>

```

A continuació es representa la forma en què Android mostra les categories:



Pràctica: Organitzant preferències (II)

Modifica la pràctica anterior perquè en lloc de mostrar les propietats en dues pantalles, les mostri en una sola, tal com es mostra a la imatge anterior.

Exercici 4.4: Incorporar les preferències dins la vostra aplicació d'Asteroides

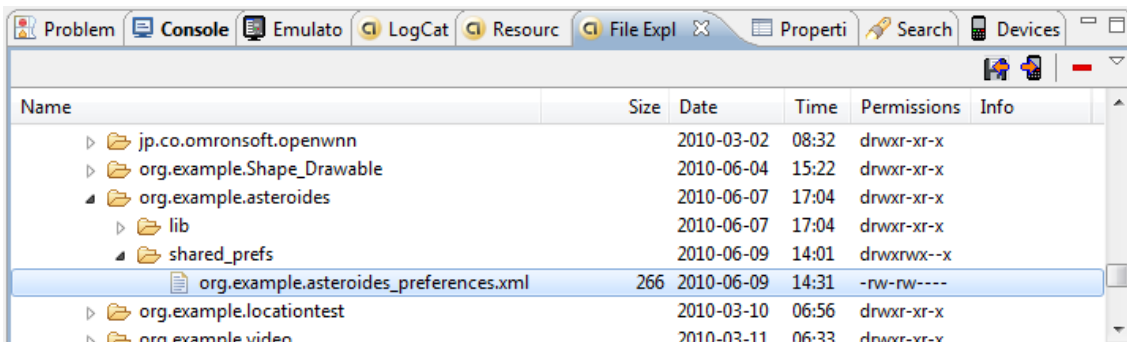
Com s'emmagatzemen les preferències d'usuari

Si un usuari modifica el valor d'una preferència, aquest quedarà emmagatzemat de forma permanent en el dispositiu. Per aconseguir aquesta persistència, Android emmagatzema les preferències seleccionades en un fitxer XML dins de la carpeta *data / data / nom.del.paquet / files / shared_prefs*, on *nom.del.paquet* ha de ser reemplaçat per el paquet de l'aplicació. El nom de fitxer per emmagatzemar les preferències d'usuari ha de ser sempre *nom.del.paquet_preferences.xml*. Això vol dir que només pot haver-hi unes preferències d'usuari per aplicació. Pot haver-hi altres fitxers de preferències; però, a diferència de les preferències d'usuari, no poden ser editades directament per l'usuari, sinó que cal accedir-hi per codi.

Pràctica: On s'emmagatzemen les preferències d'usuari

Vegem on s'han emmagatzemat les preferències que acabem de crear:

1. Per navegar pel sistema de fitxers d'un dispositiu amb Android Studio, obre l'opció *Tools> Android> Android Device Monitor* i selecciona la pestanya *File Explorer*.
2. Busca el següent fitxer: */data/data/org.examples.asteroides/shared_prefs/org.examples.asteroide_preferences.xml*



NOTA: En un dispositiu real no tindràs permís per accedir a aquests fitxers (Llevat que el dispositiu hagi estat rootejat).

3. Prem el botó del disquet per descarregar el fitxer al teu PC.
4. Visualitza seu contingut. Ha de ser similar a:

```
<map>
  <boolean name="musica" value="true" />
  <string name="grafics">1</string>
  <string name="fragments">3</string>
</map>
```

Accedint als valors de les preferències

Per descomptat, caldrà accedir als valors de les preferències per alterar el funcionament de la nostra aplicació. El següent exemple ens mostra com realitzar-ho:

```
public void mostrarPreferencias(){
    SharedPreferences pref =
        PreferenceManager.getDefaultSharedPreferences(this);
    String s = "música: " + pref.getBoolean("musica", true)
        + ", gràfics: " + pref.getString("grafics", "?");
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

La funció comença creant l'objecte *pref* de la classe *SharedPreferences* i li assigna les preferències definides per l'aplicació. A continuació crea el *String s* i li assigna els valors de dues de les preferències. S'utilitzen els mètodes *pref.getBoolean()* i *pref.getString()*, que disposen de dos paràmetres: el valor de key que volem buscar ("musica" i "gràfics") i el valor assignat per defecte en cas de no trobar aquesta key.

Finalment es visualitza el resultat utilitzant la classe *Toast*. Els tres paràmetres indicats són el context (la nostra activitat), l'*String* a mostrar i el temps que s'estarà mostrant aquesta informació.

Pràctica: Accedint als valors de les preferències

1. Còpia la funció anterior a la classe *Asteroides*. Afegeix el paràmetre que es mostra a continuació *mostrarPreferencias (View view)*.
2. Asígnala a l'atribut *onClick* del botó *Jugar* el mètode anterior.
3. Visualitza també la resta de les preferències que hagi introduït.

Verificar valors correctes

En moltes ocasions voldràs limitar els valors que un usuari pot introduir en les preferències. Per exemple, podria ser interessant que el valor introduït per l'usuari en la preferència nombre de fragments només pogués prendre valors entre 0 i 9. Per aconseguir això podem utilitzar el escoltador d'esdeveniment *onPreferenceChangeListener* que podrem assignar a una preferència. Vegem com actuar en el següent exercici:

Exercici 4.5: Verificar valors correctes d'una preferència

1. Copia el següent codi a la fi del mètode onCreate () de l'activitat PreferencesActivity:

```
final EditTextPreference fragments =
(EditTextPreference)findPreference("fragments");
fragments.setOnPreferenceChangeListener(
    new Preference.OnPreferenceChangeListener() {
        @Override
        public boolean onPreferenceChange(Preference
preference, Object newValue) {
            int valor;
            try {
                valor = Integer.parseInt((String)newValue);
            } catch (Exception e) {
                Toast.makeText(getApplicationContext(), "Ha de
ser un número",
                    Toast.LENGTH_SHORT).show();
                return false;
            }
            if (valor >= 0 && valor <= 9) {
                fragments.setSummary(
                    "En quants trossos es divideix un
asteroide (" + valor + ")");
                return true;
            } else {
                Toast.makeText(getApplicationContext(), "Màxim
de fragments 9",
                    Toast.LENGTH_SHORT).show();
                return false;
            }
        }
    });
```

El codi comença obtenint una referència de la preferència fragments, per assignar-li un escoltador que serà cridat quan canviï el seu valor. El listener comença convertint el valor introduït a sencer. En cas de produir-se un error és perquè l'usuari no ha introduït un valor adequat. En aquest cas, mostrarem un missatge i tornem *false* perquè el valor de la preferència no sigui modificat. Si no hi ha error, després de verificar el rang de valors acceptables, modifiquem l'explicació de la preferència perquè aparegui el nou valor entre parèntesis i tornem *true* per acceptar aquest valor. Si no està en el rang, mostrarem un missatge indicant el problema i tornem *false*.

Pràctica: Mostra el valor d'una preferència

En l'exercici anterior quan es modifica el nombre de fragments es mostra entre parèntesi el nou valor introduït. El funcionament no és del tot correcte, quan entrem per primera vegada o quan es canvia el telèfon de vertical a horitzontal aquest valor no es mostra. Afegim el codi necessari perquè aquest valor aparegui sempre.

2. Executa el projecte i verifica que funciona correctament.

4.6 Introduir llista de puntuacions al nostre joc.

Molts videojocs permeten recordar les puntuacions de partides anteriors, d'aquesta manera, un jugador pot tractar de superar el seu propi rècord o millorar el d'altres jugadors.

En aquest capítol ens centrarem en representar aquesta llista de puntuacions de forma atractiva utilitzant la vista **ListView**.

Intentarem que el mecanisme d'accés a aquesta llista de puntuacions sigui el més independent possible del mètode final escollit. Amb aquest propòsit, definirem la interfície *ScoreStorage*.

Exercici pas a pas: La interfície **ScoreStorage**

1. Obre l'aplicació Asteroides.
2. Prem amb el botó dret sobre la carpeta de codi (*org.example.asteroides*) i selecciona *New> Java Class*.
3. Al camp *Name*: introdueix *ScoreStorage*, al camp *Kind* introdueix *Interface* i prem Ok.
4. Introdueix el codi que es mostra a continuació:

```
public interface ScoreStorage {  
    public void storeScore(int score, String name, long date);  
    public List<String> getScoreList(int maxNo);  
}
```

Nota sobre Java: La interfície és una classe abstracta pura, és a dir una classe on s'indiquen els mètodes però no s'implementa cap (en aquest cas es diu que els mètodes són abstractes). Permet al programador de la classe establir l'estructura d'aquesta (noms de mètodes, els seus paràmetres i tipus que retorna, però no el codi de cada mètode). Una interfície també pot contenir constants, és a dir camps de tipus *static* i *final*.

Les diferents classes que definim per emmagatzemar puntuacions han d'implementar aquesta interfície. Com veus té dos mètodes. El primer per guardar la puntuació d'una partida, amb els paràmetres puntuació obtinguda, nom del jugador i data de la partida. La segona és per obtenir una llista de puntuacions prèviament emmagatzemades. El paràmetre *maxNo* indica el nombre màxim de puntuacions que ha de retornar.

1. Vegem a continuació una classe que utilitza aquesta interfície. Per a això crea en el projecte la classe *ScoreStorageList*.
2. Introdueix el següent codi:

```
public class ScoreStorageList implements ScoreStorage{  
    private List<String> scores;  
    public ScoreStorageList() {  
        scores= new ArrayList();  
        scores.add("123000 Juan Sánchez");  
        scores.add("111000 Pedro Martínez");  
        scores.add("011000 Paco Pérez");  
    }  
    @Override  
    public void storeScore(int score,  
                           String name, long date) {  
        scores.add(0, score + " "+ name);  
    }  
}
```

```

    }
    @Override
    public List<String> getScoreList(int maxNo) {
        return scores;
    }
}

```

Aquesta classe emmagatzema la llista de puntuacions (*scores*) en una llista de *String*. Té l'inconvenient que al tractar-se d'una variable local, cada vegada que es tanqui l'aplicació es perdran les puntuacions. El constructor inicialitza la llista i introdueix tres valors. La idea és que tot i que encara no estigui programat el joc i no puguem jugar, tinguem ja algunes puntuacions per poder representar una llista. El mètode *storeScore()* es limita a inserir en la primera posició de l'array un *String* amb els punts i el nom. La data no és emmagatzemada. El mètode *getScoreList()* retorna la llista de *String* sencer, sense tenir en compte el paràmetre *maxNo* que hauria de limitar el nombre de *Strings* retornats.

1. En l'activitat *MainActivity* hauràs de declarar una variable per emmagatzemar les puntuacions:

```
public static ScoreStorage scoreStorage= new ScoreStorageList();
```

Nota sobre Java: El modificador *static* permet compartir el valor d'una variable entre tots els objectes de la classe. És a dir, tot i que es creuen diversos objectes, només hi haurà una única variable magatzem compartida per tots els objectes. El modificador *public* permet accedir a la variable des de fora de la classe. Per tant, no serà necessari crear mètodes *getters* i *setters*. Per accedir a aquesta variable no tindrem més que escriure el nom de la classe seguida d'un punt i el nom de la variable. És a dir *MainActivity.scoreStorage*.

1. Perquè els jugadors puguin veure les últimes puntuacions obtingudes, modifica el quart botó del layout *activity_main.xml* perquè en lloc de el text "Sortir" es visualitzi "Puntuacions". Per a això modifica els fitxers *res / values / strings*. També seria interessant que canviessis el fitxer *res / values-en / strings*.
2. Modifica l'escoltador associat al quart botó perquè cridi al mètode:

```

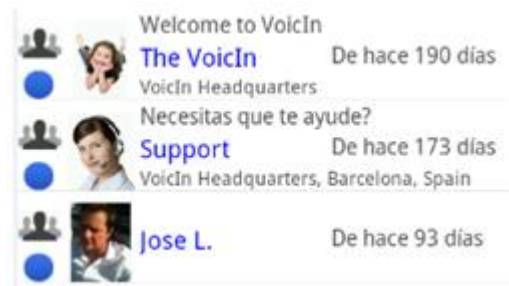
public void showScores(View view) {
    Intent i = new Intent(this, Scores.class);
    startActivity(i);
}

```

3. De moment no et permetrà executar l'aplicació. Fins que en el següent apartat no creem l'activitat *Scores* no serà possible.

4.7 La vista *ListView*.

Una vista *ListView* mostra una llista verticalment de diversos elements, on cada element pot definir com un *Layout*. La seva utilització és una mica complexa, però molt potent. Un exemple el podem veure a la següent figura:



Definir una *ListView* comporta les següents quatre passes:

1. Dissenyar un *Layout* que el conté el *ListView*
2. Dissenyar un *Layout* individual que es repetirà a la llista
3. Implementar una activitat que visualitzi el *Layout* amb el *ListView*
4. Personalitzar cadascuna dels *Layouts* individuals segons les nostres dades

Vegem aquests passos amb més detall:

Per utilitzar un *ListView* dins d'un *Layout* pots d'usar l'estructura següent:







```
<FrameLayout>
    <ListView
        android:id="@android:id/list"... />
    <TextView
        android:id="@android:id/empty"
        ... />
</FrameLayout>
```

On tenim un *FrameLayout* que permet visualitzar dos possibles elements, un o altre, però no els dos simultàniament. El primer és el *ListView* que es visualitzarà quan hi hagi algun element a la llista. El segon pot ser qualsevol tipus de vista i es visualitzarà quan no hi hagi elements a la llista. El sistema controla la visibilitat de forma automàtica, només has d'anar amb compte d'identificar cadascun dels elements amb el valor exacte que es mostra.

NOTA: Recorda que per crear nous identificadors has d'utilitzar l'expressió "@ + id / nom_identificador". El caràcter @ vol dir que es tracta d'un identificador de recurs que es definirà en la classe R.java. El caràcter + vol dir que el recurs ha de ser creat en aquest moment. En aquest cas hem utilitzat identificadors definits en el sistema (és a dir @android: vol dir que és un recurs definit en la classe android.R.java).

Un cop creat el *layout* que conté el *ListView* haurem de visualitzar-lo en una activitat. Per a aquest propòsit utilitzarem un tipus d'activitat especial, *ListActivity*.

També haurem d'indicar al sistema cada un dels *Layouts* individuals que contindrà el *ListView*. Això ho farem cridant al mètode *setListAdapter()*. Existeixen diverses alternatives amb diferents graus de dificultat. Per a una millor comprensió anirem mostrant tres exemples d'ús de *setListAdapter ()*, de més senzill a més complex.

<p>Puntuacions</p> <p>123000 Juan Sánchez</p> <p>111000 Pedro Martínez</p> <p>123000 Paco Pérez</p>	<p>Puntuacions</p> <p> 123000 Juan Sánchez Un altre text</p> <p> 111000 Pedro Martínez Un altre text</p> <p> 123000 Paco Pérez Un altre text</p>	<p>Puntuacions</p> <p> 123000 Juan Sánchez Un altre text</p> <p> 111000 Pedro Martínez Un altre text</p> <p> 123000 Paco Pérez Un altre text</p>
---	---	---

Les captures anteriors mostren els tres *ListView* que construirem. El de l'esquerra es limita a mostrar una llista de *Strings*. El de centre visualitza una llista d'un *Layout* dissenyat per nosaltres. Encara que aquest *Layout* té diversos components (una imatge i dos textos), només canviem un dels textos. En l'últim exemple canviarem també la imatge de cada element.

Un *ListView* que visualiza una lista de *Strings*

Exercici pas a pas: Un *ListView* que visualiza una lista de *Strings*

1. El *Layout* que utilitzarem a Asteroides per mostrar les puntuacions es dirà *scores.xml*. que inclou una vista *ListView*. Crea el *Layout* amb el següent codi:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Puntuacions"
    android:gravity="center"
    android:layout_margin="10px"
    android:textSize="30sp"/>
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="0dip"
    android:layout_weight="1">
    <ListView
        android:id="@android:id/list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:drawSelectorOnTop="false" />
    <TextView
        android:id="@android:id/empty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="No hi ha puntuacions" />
</FrameLayout>
</LinearLayout>
```

2. Necessitem ara crear l'activitat *Scores* per visualitzar el *Layout* anterior. Crea una nova classe en el teu projecte i introdueix el següent codi:

```
public class Scores extends ListActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.scores);
        setListAdapter(new ArrayAdapter<this,
```



```

        android.R.layout.simple_list_item_1,
        MainActivity.scoreStorage.getScoreList(10)));
    }
}

```

Tota activitat que hagi de mostrar un *ListView* ha d'heretar de *ListActivity*. A més, ha de cridar el mètode *setListAdapter()* per indicar l'adaptador amb la llista d'elements a visualitzar. En l'exemple s'ha utilitzat una de les possibilitats més senzilles, per crear un adaptador, fer servir la classe *ArrayAdapter* <classe>. Un *ArrayAdapter* crea les vistes del *ListView* a partir de les dades emmagatzemades en un *array*. Pots utilitzar un *array* que contingui dades de qualsevol classe, no tens més d'indicar a <Classe> la classe desitjada. En aquest cas s'utilitza d'una matriu de *String*. El constructor d'*ArrayAdapter* <classe> té tres paràmetres: El primer paràmetre és un *Context* amb informació sobre l'entorn de l'aplicació. Utilitzarem com a context la mateixa activitat que fa la cridada. El segon paràmetre és un *layout*, utilitzat per representar cada element de la llista. En aquest exemple, en lloc de definir un de nou, farem servir una ja definit en el sistema. L'últim paràmetre és un *array* amb els *strings* a mostrar. Per a això, fem una crida a mètode *getScoreList ()* que ens retorna aquesta llista de l'objecte estàtic magatzem de la classe *Asteroides*.

3. Recorda que tota nova activitat ha de ser registrada a *AndroidManifest.xml*.
4. Prova si funcionen les modificacions introduïdes.

Un *ListView* que visualiza *Layouts* personalitzats

Anem a personalitzar el *ListView* anterior perquè cada element de la llista sigui un *Layout* definit per nosaltres. Per a això segueix els següents passos:

Exercici pas a pas: Un *ListView* que visualitza *layouts* personalitzats

1. Substitueix la classe anterior per:

```

public class Scores extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.scores);
        setListAdapter(
            new ArrayAdapter(this,
                R.layout.list_element,
                R.id.title,
                MainActivity.scoreStorage.getScoreList(10)));
    }
}

```

Com hem explicat, la classe *ArrayAdapter* <*String*> permet inserir les dades des d'un *array* d'*String* en el nostre *ListView*. En aquest exemple s'utilitza un constructor amb quatre paràmetres:

- **this**: és el context, amb informació sobre l'entorn de l'aplicació.

- **R.layout.list_element**: és una referència de recurs a la vista que serà utilitzada repetides vegades per formar la llista. Es defineix a continuació.
 - **R.id.title**: identifica un id de la vista anterior que ha de ser un *TextView*. El seu text serà reemplaçat pel que s'indica en el següent paràmetre.
 - **MainActivity.scoreStorage.getScoreList (10)**: vector de String amb els textos que seran visualitzats en cada un dels *TextView*. Aquesta llista s'obté accedint a la classe *MainActivity* i a la seva variable estàtica *scoreStorage* cridant al seu mètode **getScoreList ()**.
2. Ara hem de definir el *layout* que representarà cadascun dels elements de la llista. Crea el fitxer de *res / layout / list_element.xml* amb el següent codi:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"

    android:layout_height="?android:attr/listPreferredItemHeight">
    <ImageView android:id="@+id/icon"
        android:layout_width="?android:attr/listPreferredItemHeight"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true"
        android:src="@drawable/asteroide2"/>

    <TextView android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/icon"
        android:layout_alignParentTop="true"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:singleLine="true" />

    <TextView android:id="@+id/subtitle"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Un altre text"
        android:layout_toRightOf="@id/icon"
        android:layout_below="@id/title"
        android:layout_alignParentBottom="true"
        android:gravity="center"/>
</RelativeLayout>
```

Aquest *layout* representa una imatge a l'esquerra amb dos textos a la dreta, un més gran a la part superior. Per combinar aquests elements s'ha escollit un *RelativeLayout*, on l'alt s'estableix a partir d'un paràmetre de configuració de sistema? *Android: attr / listPreferredItemHeight*. El primer element que conté és un *IMAGEVIEW* alineat a l'esquerra. El seu alt és la mateixa que el contenidor (*match_parent*) mentre que l'ample s'estableix amb el mateix paràmetre que l'alt del contenidor. Per tant la imatge serà quadrada.

3. Pots descarregar-te les imatges utilitzades en l'aplicació Asteroides del classroom. Copia el fitxer *asteriode1.png*, *asteriode2.png* i *asteriode3.png* a la carpeta *res / drawable*.
4. Executa l'aplicació i verifica el resultat.

Un ListView amb el nostre propi adaptador.

En l'exercici anterior hem vist com podíem associar un *layout* definit per nosaltres al *ListView* i personalitzar un dels seus camps. Si volem alguna cosa més adaptable, per exemple canviar diversos camps, haurem d'escriure el nostre propi adaptador extenent la classe *BaseAdapter*. En aquesta classe caldrà sobreescrivre els següents quatre mètodes:

- *View getView (int position, View convertView, ViewGroup parent)*

Aquest mètode ha de construir un nou objecte *View* que serà visualitzat a la posició *position*. Opcionalment podem partir d'una vista base *convertView* per generar més ràpid aquest objecte. L'últim paràmetre correspon al contenidor de vistes on l'objecte serà afegit.

- *int getCount ()*

Retorna el nombre d'elements de la llista.

- *Object getItem (int position)*

Retorna l'element en una determinada posició de la llista.

- *long getItemId (int position)*

Retorna l'identificador de fila d'una determinada posició de la llista.

Exercici pas a pas: Un *ListView* amb el nostre propi adaptador

1. Crea la classe *MyAdapter.java* en el projecte amb el següent codi:

```
public class MyAdapter extends BaseAdapter {
    private final Activity activity;
    private final List<String> list;

    public MyAdapter(Activity activity, List<String> list) {
        super();
        this.activity = activity;
        this.list = list;
    }

    public View getView(int position, View convertView,
                        ViewGroup parent) {
        LayoutInflater inflater =
activity.getLayoutInflater();
        View view = inflater.inflate(R.layout.list_element,
null, true);
        TextView textView
=(TextView)view.findViewById(R.id.title);
        textView.setText(list.get(position));
        ImageView
imageView=(ImageView)view.findViewById(R.id.icon);
        switch (Math.round((float)Math.random()*3)){
            case 0:

imageView.setImageResource(R.drawable.asteroide1);
```

```

        break;
    case 1:
imageView.setImageResource(R.drawable.asteroide2);
        break;
    default:
imageView.setImageResource(R.drawable.asteroide3);
        break;
    }
    return view;
}

public int getCount() {
    return list.size();
}

public Object getItem(int arg0) {
    return list.get(arg0);
}

public long getItemId(int position) {
    return position;
}
}

```

2. En el constructor de la classe s'indica l'activitat on s'executarà i la llista de dades a visualitzar. El mètode més important d'aquesta classe és *getView ()* el qual ha de construir els diferents Layouts que seran afegits a la llista. Comencem construint un objecte View a partir del codi xml definit en *list_element.xml*. Aquest treball es realitza per mitjà de la classe *LayoutInflater*. Després, es modifica el text d'un dels *TextView* segons l'array que es passa al constructor. Finalment, s'obté un nombre a l'atzar (*Math.round ()*) i s'assigna un dels tres gràfics de forma aleatòria.
3. Reemplaça en la classe Scores la crida a constructor de *ArrayAdapter <String>* per:

```

setListAdapter(new MyAdapter(this,

MainActivity.scoreStorage.getScoreList(10)));

```

4. Executa la aplicació i verifica el resultat.

Exercici 4.6: Incorporar la llista de puntuacions dins la vostra aplicació d'Asteroides

Detectar una pulsació sobre un element de la llista

Un *ListView* pot tenir diferents components que ens permetin interaccionar amb l'usuari. Per exemple, cada element definit en *getView ()* pot tenir botons per a diferents accions.

Hi ha un tipus d'interacció molt senzilla de definir. La classe *ListActivity* té un mètode que és invocat cada vegada que es prem sobre un element de la llista. El següent exercici il·lustra com utilitzar-lo.

Exercici pas a pas: Detectar una pulsació sobre un element de la llista a Asteroides

1. Afegeix el següent mètode a la classe Scores.java:

```
@Override protected void onItemClick(ListView listView,
                                     View view, int position, long id) {
    super.onItemClick(listView, view, position, id);
    Object o = getListAdapter().getItem(position);
    Toast.makeText(this, "Selecció: " + Integer.toString(position)
                  + " - " + o.toString(), Toast.LENGTH_LONG).show();
}
```

2. Executa l'aplicació, prem a "Puntuacions" i després en una puntuació per verificar el resultat.

4.8 Les intencions

Com ja hem vist, una intenció representa la voluntat de realitzar alguna acció o tasca, com fer una trucada de telèfon o visualitzar una pàgina web. Una intenció ens permet llançar una activitat o servei de la nostra aplicació o d'una aplicació diferent. Tenen un gran potencial en Android, pel que resulta important conèixer-les i dominar-les.

Hi ha dos tipus d'intencions:

- **Intencions explícites:** s'indica exactament el component (la activitat) a llançar. La seva utilització típica és la d'anar executant els diferents components interns d'una aplicació. Per exemple, des de l'activitat Asteroides llancem *SobreActivity* per mitjà d'una intenció explícita.
- **Intencions implícites:** poden sol·licitar tasques abstractes, com "vull fer una foto" o "vull enviar un missatge". A més les intencions es resolen en temps d'execució, de manera que el sistema mirarà quants components han registrat la possibilitat d'executar aquest tipus d'intenció. Si en troba més d'una, el sistema pot preguntar a l'usuari el component que prefereix utilitzar.

Per llançar una activitat de forma implícita podem fer servir el constructor Intent (String *action*, Uri *uri*). On *action* és l'acció que volem llançar i *uri* són les dades que necessita l'acció i que depèn del tipus d'acció. Per exemple, per fer una cridada de telèfon:

```
Intent intent = new Intent(Intent.ACTION_DIAL,
                          Uri.parse("tel:971000000"));
startActivity(intent);
```

Les accions estàndard les podeu trobar a la documentació d'Android (<https://developer.android.com/reference/android/content/Intent>) i són:

- | | |
|--------------------------------------|--------------------------------------|
| • ACTION_MAIN | • ACTION_CHOOSER |
| • ACTION_VIEW | • ACTION_GET_CONTENT |
| • ACTION_ATTACH_DATA | • ACTION_DIAL |
| • ACTION_EDIT | • ACTION_CALL |
| • ACTION_PICK | • ACTION_SEND |

- [ACTION_SENDTO](#)
- [ACTION_ANSWER](#)
- [ACTION_INSERT](#)
- [ACTION_DELETE](#)
- [ACTION_RUN](#)
- [ACTION_SYNC](#)
- [ACTION_PICK_ACTIVITY](#)
- [ACTION_SEARCH](#)
- [ACTION_WEB_SEARCH](#)
- [ACTION_FACTORY_TEST](#)

Si volem indicar al sistema operatiu que una de les nostres aplicacions és capaç de resoldre una d'aquestes *actions* ho hem d'indicar dins AndroidManifest utilitzant una etiqueta <intent-filter>

```
<activity
    android:name=".EnviarCorreuPersonalitzat"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

Exercici 4.7: Ús d'intencions implícites:

1. Crea un nou projecte amb nom Intencions i tipus Empty Views Activity
2. El Layout de l'activitat inicial ha d'estar format per cinc botons, tal com es mostra a continuació



3. Obre l'activitat principal i incorpora els mètodes corresponents per llançar cada una de les accions indicades en els botons.

El nom de les accions i les *uri* que necessita cada acció són:

- Pàgina web:
 - o `Intent.ACTION_VIEW`
 - o `Uri.parse("http://www.iesjoanramis.org/")`
- Cridada de telèfon:
 - o `Intent.ACTION_DIAL`
 - o `Uri.parse("tel:971000000")`
- Google Maps:
 - o `Intent.ACTION_VIEW`
 - o `Uri.parse("geo:39.887642,4.254319")`
- Fer foto:
 - o `MediaStore.ACTION_IMAGE_CAPTURE`
- En el cas d'enviar correu:
 - o `Intent.ACTION_SEND`
 - o Un cop creat l'objecte intent se li poden afegir extres com en el cas dels intents explícits:

- `intent.setType("text/plain");`
- `intent.putExtra(Intent.EXTRA_SUBJECT, "assumpte");`
- `intent.putExtra(Intent.EXTRA_TEXT, "text del correu");`
- `intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"info@iesjoanramis.org"});`

4. Associa l'atribut `onClick` de cadascun dels botons al mètode corresponent
5. Executa l'aplicació en un terminal real. Observa com el botó enviar Correu et permet seleccionar entre diferents aplicacions amb aquesta funcionalitat.

Recursos addicionals: Taula amb intencions que podem utilitzar d'aplicacions Google

Aplicació	URI	Acció	Resultat
Browser	<code>http://direcció_web</code> <code>https://direcció_web</code>	VIEW	Obre una finestra del navegador amb una URL.
	<code>""</code> (cadena buida) <code>http://direcció_web</code> <code>https://direcció_web</code>	WEB_SEARCH	Realitza una cerca web. S'indica la cadena de cerca en l'extra <code>SearchManager.QUERY</code>
Dialer	<code>tel:número_telemó</code>	CALL	Realitza una trucada de telèfon. Els números vàlids es defineixen en IETF RFC 3966 . Entre aquests s'inclouen: tel:9999999999 y tel:(999)999999 . Necessitam el permís <code>android.permission.CALL_PHONE</code>
	<code>tel:número_telemó</code> <code>voicemail:</code>	DIAL	Introdueix un número sense arribar a realitzar la trucada.
Google Maps	<code>geo:latitud,longitud</code> <code>geo:lat,long?z=zoom</code> <code>geo:0,0?q=direcció</code> <code>geo:0,0?q=búsqueda</code>	VIEW	Obre l'aplicació Google Maps per a una localització determinada. El camp <code>z</code> especifica el nivell de zoom.
Google Streetview	<code>google.streetview:cbll=latitud,longitud</code> & <code>cbp=1,yaw,pitch,zoom</code> & <code>mz=mapZoom</code>	VIEW	Obre l'aplicació Street View per a la ubicació donada. L'esquema de URI se basa en la sintaxis que utilitza Google Maps. Només el camp <code>cbll</code> és obligatori.

Pràctica: Ús d'intencions implícites

1. Crea nous botons en l'aplicació de l'exercici anterior i experimenta amb un altre tipus d'accions i URI. Pots consultar la taula anterior.
2. Compara les accions VIEW i WEB_SEARCH. Trobes alguna diferència?
3. Compara les accions CALL i DIAL. Trobes alguna diferència?
4. Experimenta amb Google Streetview