CFGS: DESENVOLUPAMENT D'APLICACIONS MULTIPLATAFORMA

Mòdul: Programació multimèdia i dispositius mòbils

TEMA 3: INTERFÍCIES D'USUARI. LAYOUTS I CONTROLS BÀSICS

Germán Gascón
german.gascon.d@ies1xabia.org

Introducció

- Els **layouts** són elements **no visuals** destinats a controlar la distribució, posició i dimensions dels controls que s'insereixen en el seu interior. És a dir, actuen com a **contenidors** d'una o més **Views**.
- Aquests components **estenen** a la classe base **ViewGroup**, com molts components contenidors, és a dir, capaços de contindre a altres controls.
- A més un Layout pot contindre a un altre Layout ja que és un descendent de la classe **View**.

FrameLayout

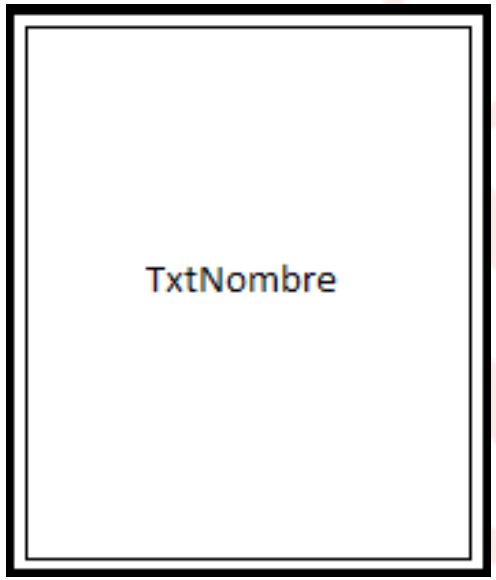
- És el més simple de tots els layouts d'Android.
- Col·loca **tots** els seus controls fills alineats amb la seua **cantonada superior esquerra**, de manera que cada control quedarà ocult pel control següent (llevat que aquest últim tinga transparència).
- Sol utilitzar-se per a mostrar un únic control en el seu interior, a manera de contenidor (placeholder) senzill per a un només element substituïble, per exemple una imatge.
- Propietats **android:layout_width** i **android:layout_height**, que podran prendre els valors "**match_parent**" (perquè el control fill prenga la dimensió de la seua layout contenidor) o "**wrap_content**" (perquè el control fill prenga la dimensió del seu contingut)

Exemple FrameLayout

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

    <EditText android:id="@+id/TxtNombre"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text" />

</FrameLayout>
```



LinearLayout

- Aquest layout **apila un darrere l'altre** tots els seus **elements** fills en sentit **horitzontal** o **vertical** segons s'establisca la seua propietat **android:orientation**.
- Igual que en un FrameLayout, els elements continguts en un LinearLayout poden establir les seues propietats **android:layout_width** i **android:layout_height** per a determinar les seues dimensions dins del layout.
- La propietat **android:layout_weight** permet donar als elements continguts en el layout unes dimensions proporcionals entre ells.
- Per exemple, si incloem en un LinearLayout **vertical** dos quadres de text i a un d'ells li establim un **layout_weight**=”1″ i a l'altre un **layout_weight**=”2″ aconseguirem com a efecte que tota la superfície del layout quede ocupada pels dos quadres de text i que a més el segon siga **el doble** d'alt.

Exemple LinearLayout

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText android:id="@+id/TxtDato1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text"
        android:layout_weight="1" />

    <EditText android:id="@+id/TxtDato2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text"
        android:layout_weight="2" />

</LinearLayout>
```

TableLayout (I)

- Permet distribuir els seus elements fills de forma tabular, definint les files i columnes necessàries, i la posició de cada component dins de la taula.
- L'estructura de la taula es defineix de forma similar a com es fa en HTML, és a dir, indicant les files que compondran la taula (objectes **TableRow**), i dins de cada fila les columnes necessàries, amb l'excepció que no existeix cap objecte especial per a definir una columna sinó que directament inserirem els controls necessaris dins del **TableRow** i cada component inserit (que pot ser un control senzill o fins i tot un altre ViewGroup) correspondrà a una columna de la taula.
- El **nombre** final de **files** de la taula es correspondrà amb el número d'elements **TableRow** inserits, i el **nombre** total de **columnes** quedarà determinat pel **nombre de components de la fila que més components continga**.

TableLayout (II)

- L'ample de cada **columna** es correspondrà amb **l'ample del major component** d'aquesta columna, però existeixen una sèrie de propietats que ens ajudaran a modificar aquest comportament:
  - **android:stretchColumns**. Indicarà quines columnes poden **expandir-se** per a absorvir l'espai lliure deixat per les altres columnes a la dreta de la pantalla.
  - **android:shrinkColumns**. Indicarà quines columnes es poden **contraure** per a deixar espai a la resta de columnes que es puguen eixir per la dreta de la pantalla.
  - **android:collapseColumns**. Indicarà quines columnes de la taula es volen **ocultar** completament.
- Aquestes propietats poden rebre una llista d'índexs de columnes separats per comes (exemple: **android:stretchColumns**="0,1,2") o un asterisc per a indicar que ha d'aplicar a totes les columnes.
- Una cel·la pot ocupar l'espai **de diverses columnes** de la taula (anàleg a l'atribut colspan d'HTML). Això s'indicarà mitjançant la propietat **android:layout_span** del component concret que haurà de prendre aquest espai.

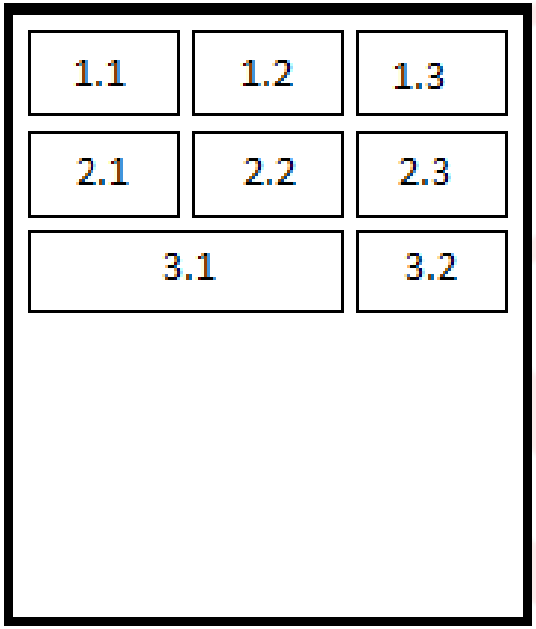
Exemple TableLayout

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

    <TableRow>
        <TextView android:text="Celda 1.1" />
        <TextView android:text="Celda 1.2" />
        <TextView android:text="Celda 1.3" />
    </TableRow>

    <TableRow>
        <TextView android:text="Celda 2.1" />
        <TextView android:text="Celda 2.2" />
        <TextView android:text="Celda 2.3" />
    </TableRow>

    <TableRow>
        <TextView android:text="Celda 3.1"
                  android:layout_span="2" />
        <TextView android:text="Celda 3.2" />
    </TableRow>
</TableLayout>
```



RelativeLayout

- **RelativeLayout** permet especificar la **posició** de cada **element** de **forma relativa al seu element pare o a qualsevol altre element inclòs en el propi layout**.
- En incloure un nou element X podrem indicar per exemple que ha de col·locar-se davall de l'element I i alineat a la dreta del layout pare.
- Per exemple, les propietats **android:layout_below** i **android:layout_alignParentRight** permeten especificar davall de quin element se situarà l'element actual i amb quina alineació respecte a l'element pare.

RelativeLayout propietats relatives a un altre control

- **android:layout_above** a dalt de l'element indicat.
- **android:layout_below** davall de l'element indicat.
- **android:layout_toLeftOf** a l'esquerra de l'element indicat.
- **android:layout_toRightOf** a la dreta de l'element indicat.
- **android:layout_alignLeft** la vora esquerra quedarà en la mateixa posició que la vora esquerra de l'element indicat.
- **android:layout_alignRight** la vora dreta quedarà en la mateixa posició que la vora dreta de l'element indicat.
- **android:layout_alignTop** la vora superior quedarà en la mateixa posició que la vora superior de l'element indicat.
- **android:layout_alignBottom** la vora inferior quedarà en la mateixa posició que la vora inferior de l'element indicat.
- **android:layout_alignBaseline** les línies base dels elements coincidiran.

RelativeLayout propietats relatives al layout pare

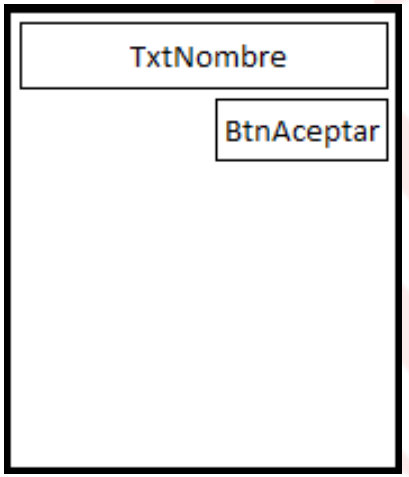
- Les següents propietats són booleanes (true o false).
- **android:layout_alignParentLeft** si el valor és "**true**", la vora esquerra quedarà en la mateixa posició que la vora esquerra de l'element pare.
- **android:layout_alignParentRight** si el valor és "**true**", la vora dreta quedarà en la mateixa posició que la vora dreta de l'element pare.
- **android:layout_alignParentTop** si el valor és "**true**", la vora superior quedarà en la mateixa posició que la vora superior de l'element pare.
- **android:layout_alignParentBottom** si el valor és "**true**", la vora inferior quedarà en la mateixa posició que la vora inferior de l'element pare.
- **android:layout_centerHorizontal** si el valor és "**true**", l'element quedarà centrat horitzontalment respecte a l'element pare.
- **android:layout_centerVertical** si el valor és "**true**", l'element quedarà centrat verticalment respecte a l'element pare.
- **android:layout_centerInParent** si el valor és "**true**", l'element quedarà centrat horitzontal i verticalment respecte a l'element pare.

Exemple RelativeLayout

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

        <EditText android:id="@+id/TxtNombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="text" />

        <Button android:id="@+id/BtnAceptar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/TxtNombre"
            android:layout_alignParentRight="true" />
</RelativeLayout>
```



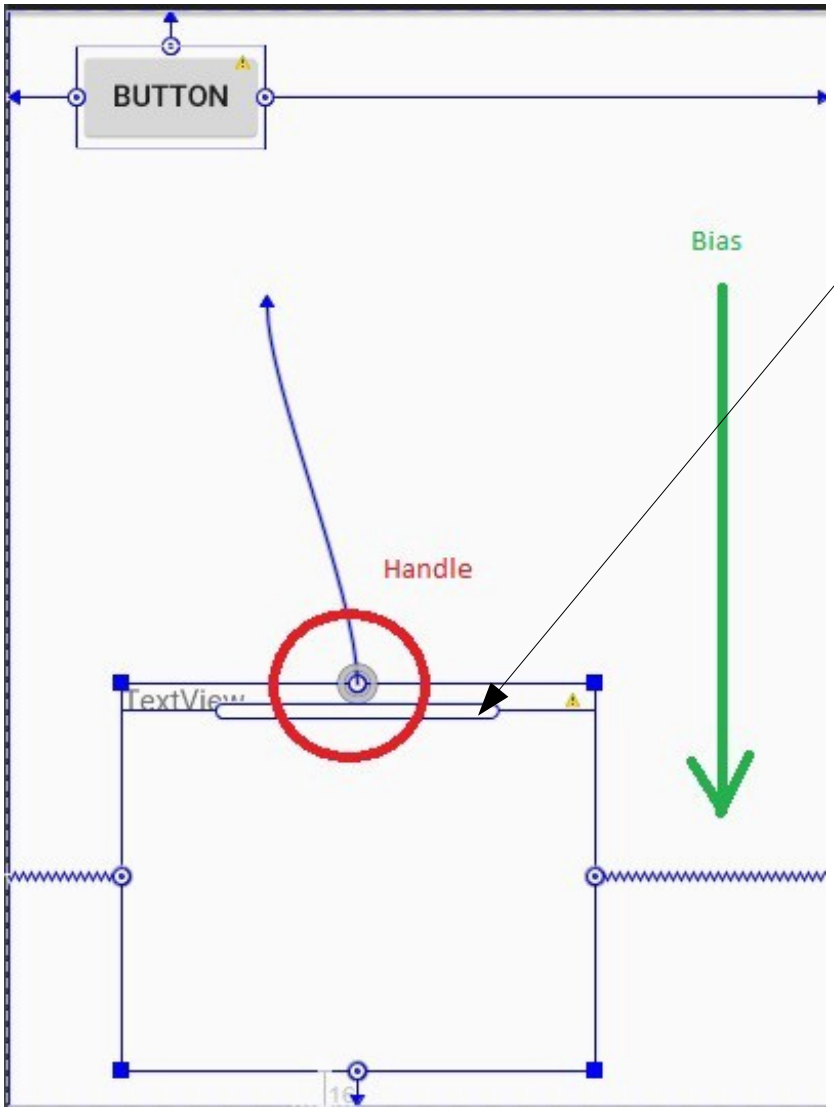
ConstraintLayout

- **ConstraintLayout**, al igual que el RelativeLayout, permet especificar la **posició** de cada **element** de **forma relativa al seu element pare o a qualsevol altre element inclòs en el propi layout**.
- És més **flexible** que el RelativeLayout i permet fer dissenys complexes sense tindre que aniuar varios layouts.
- El principal avantatge respecte al RelativeLayout es que podem dissenyar lliurement l'interfície d'usuari al nostre gust de forma visual i deixar a l'Android Studio que faça la tasca dura.

ConstraintLayout

- Ofereix uns paràmetres anomenats _**bias** que s'utilitzen per posicionar una vista mitjançant un desplaçament horitzontal i vertical del 0% i del 100% respecte dels identificadors (marcats en el cercle roig).
- Example:
  - app:layout_constraintHorizontal_bias=”0.33"
  - app:layout_constraintVertical_bias=”0.53"

ConstraintLayout



- **Guia base** (rectangle amb les voreres arrodonides, davall del cercle roig) s'utilitza per alinear el contingut de la vista amb alguna altra referència.
- **Guia quadrada** (en cada cantonada de la vista) utilitzades per ajustar la mida en dp.

Unitats de mesura en Layouts

- **dp** (Density-independent Píxels): unitat abstracta de mesura basada en la densitat de de la pantalla. Aquestes unitats són relatives a 160 dpi, per tant, 1dp és igual a 1 píxel en pantalles amb una densitat de 160 dpi. Per a pantalles de major o menor densitat, augmenta o decrementa proporcionalment, d'aquesta forma s'aconsegueix el contingut es veurà igual independentment de la densitat del dispositiu mòbil. **Unitat recomanada**.
- **sp** (Scale-independent Píxels): similar a dp però també escala les fonts en cas que l'usuari haja indicat una grandària específica.
- Existeixen altres unitats vàlides com px (píxels), pt (punts), in (polzades), mm (mil·límetres) però no estan recomanades per al desenvolupament d'interfícies en Android.

Propietats comunes a tots els Layouts

- El valor de les següents propietats **s'expressen** en **unitats de mesura**, habitualment dp.
- Opcions de **marge exterior**:
  - **android:layout_margin** indica el marge que deixarà l'element en totes les seues vores (esquerre, dret, superior i inferior).
  - **android:layout_marginBottom** indica el marge que deixarà l'element en la vora inferior.
  - **android:layout_marginTop** indica el marge que deixarà l'element en la vora superior.
  - **android:layout_marginLeft** indica el marge que deixarà l'element en la vora esquerra
  - **android:layout_marginRight** indica el marge que deixarà l'element en la vora dreta.

Propietats comunes a tots els Layouts

- Opcions de **marge interior**:
  - **android:padding** indica el farciment que deixarà l'element des de totes les seues vores (esquerre, dret, superior i inferior) cap a dintre.
  - **android:paddingBottom** indica el farciment que deixarà l'element des de la vora inferior cap a dintre.
  - **android:paddingTop** indica el farciment que deixarà l'element des de la vora superior cap a dintre.
  - **android:paddingLeft** indica el farciment que deixarà l'element des de la vora esquerra cap a dintre.
  - **android:paddingRight** indica el farciment que deixarà l'element des de la vora dreta cap a dintre.

Padding vs margin

Botons

- El SDK d'Android ens proporciona tres tipus bàsics de botons:
  - Els clàssics de **text** (Button)
  - Els de tipus **on/off** (ToggleButton i Switch).
  - Els que poden contindre una **imatge** (ImageButton).
- En els següents apartats veurem algunes de les propietats de cadascun d'aquests botons. Per a una referència completa de totes les propietats consultar developer.android.com.

Button

- És el botó més bàsic que podem utilitzar i normalment conté un simple text, que com ja hem comentat, és recomanable que siga una referència a una entrada de l'arxiu strings.xml.
- Propietats més utilitzades:
  - **android:text** permet especificar el text que apareixerà en el botó.
  - **android:background** permet especificar el color de fons del botó.
  - **android:typeface** permet especificar l'estil de la font (none, normal, sans, serif, monospace).
  - **android:textcolor** permet especificar el color de la font.
  - **android:textSize** permet especificar la grandària del text.

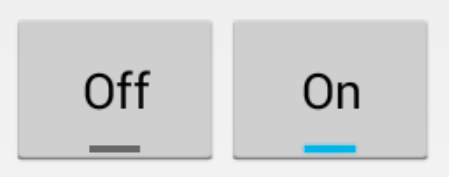
Exemple Button

```
<Button android:id="@+id/BtnBotonSimple"
        android:text="@string/click"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

ToggleButton

- És un tipus de botó que pot romandre en **dos possibles estats**, premut o no_premut.
- Hem de definir dos textos depenent del seu estat, assignant les propietats **android:textOn** i **android:textoOff**.

```
<ToggleButton android:id="@+id/BtnToggle"
        android:textOn="@string/on"
        android:textOff="@string/off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```



Switch

- És un botó molt similar al ToggleButton anterior, on tan sols **canvia el seu aspecte visual**, que en comptes de mostrar un estat o un altre sobre el mateix espai, es mostra en forma de deslizador o interruptor.

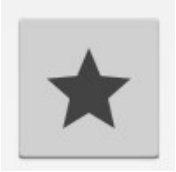
```
<Switch android:id="@+id/BtnSwitch"
        android:textOn="@string/on"
        android:textOff="@string/off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

ImageButton

- En aquest tipus de botons podem definir una imatge a mostrar en comptes d'un text, per al que haurem d'assignar la propietat **android:src**.
- Assignarem aquesta propietat amb el descriptor d'algun recurs que hàgem inclòs en les carpetes **/res/drawable**.
- Si per exemple, incloem una imatge anomenada “ic_estrela.png”, charemos referència a aquesta imatge mitjançant el recurs “@drawable/ic_estrela”.
- Addicionalment, en tractar-se d'un control de tipus imatge també hauríem d'acostumar-nos a assignar la propietat **android:contentDescription** amb una **descripció textual** de la imatge, de manera que la nostra **aplicació** siga el **més accessible** possible.

Exemple ImageButton

```
<ImageButton android:id="@+id/BtnImagen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:contentDescription="@string/icono_ok"
        android:src="@drawable/ic_estrella" />
```



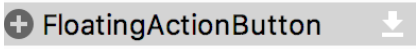
Button amb imatges

- També podem afegir una imatge a un botó normal (**Button**) a manera d'element suplementari al text.
- Per a això podem utilitzar les següents propietats:
  - **android:drawableLeft**
  - **android:drawableRight**
  - **android:drawableTop**
  - **android:drawableBottom**
- Indicant com a valor l'aneu de la imatge que volem mostrar.
- A més també podem indicar l'espai entre la imatge i el text mitjançant la propietat **android:drawablePadding**.

Exemple Button amb imatges

```
<Button android:id="@+id/BtnBotonMasImagen"
        android:text="@string/click"
        android:drawableLeft="@drawable/ic_estrella"
        android:drawablePadding="5dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

FloatingActionButton (I)

- Amb la publicació per part de Google de la llibreria **Design Support Library,** podem incloure aquest tipus de botons flotants al més pur estil Material Design i a més assegurar-nos la **compatibilitat amb versions anteriors** a Android 5.0.
- El primer que hem de fer és afegir aquesta llibreria al nostre projecte, podem fer-ho de dos formes:
  - Desde l'editor d'interfícies gràfiques 

  - Editant l'arxiu **build.gradle** del nostre **mòdul app** i afegim la següent línia:

```
dependencies {
    ...
    compile 'com.android.support:design:27.1.1'
}
```

- Els números de la versió poden canviar.
- Una vegada afegida la referència a la llibreria, guardem l'arxiu i ens assegurem de prémer l'opció "**Sync Now**" que ens apareixerà en la part superior dreta de l'editor de codi.
- Després d'això, **Gradle descarregarà automàticament** els fitxers necessaris perquè puguem fer ús de la llibreria.

FloatingActionButton (II)

- Quan es fa ús de components de les llibreries de suport amb compatibilitat per a versions anteriors, la forma d'instanciar el component canvia una mica.
- Cal indicar la ruta completa al component, per exemple per a fer referència al component FloatingActionButton escriuríem:

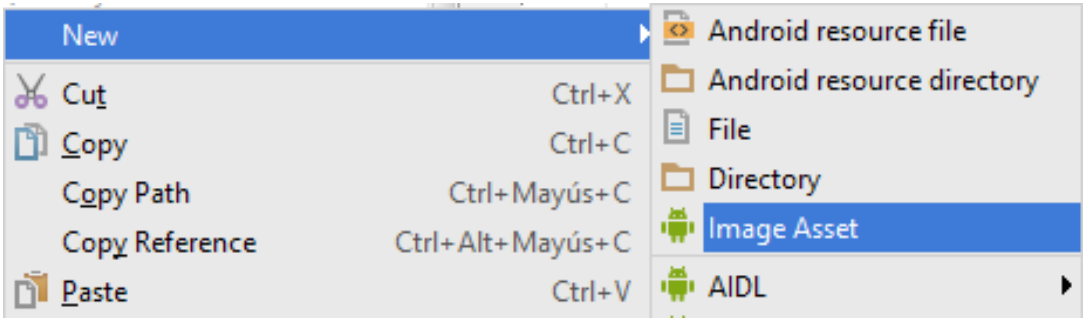
```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_add"
    app:fabSize="normal"
    app:borderWidth="0dp" />
```

FloatingActionButton (III)

- La propietat més rellevant, igual que en el cas d'un ImageButton, és **android:src**, amb la qual podem assignar al control la imatge a mostrar.
- Una altra propietat és **app:fabSize** que pot rebre els valors “normal” i “mini”, que determinen la grandària del control dins de les dues grandàries estandar definits en les especificacions de Material Design.
- Llevat que específiquem el **color del botó** explícitament, el botó prendrà per defecte l'accent **color**.

Afegir imatges a Android Studio (I)

- Android Studio incorpora una utilitat anomenada **Asset Studio** amb la qual podem afegir ràpidament a un projecte algunes imatges o icones estandar de una llista bastant àmplia de mostres disponibles, o utilitzar les nostres pròpies imatges personalitzades.
- Per a accedir a aquesta utilitat fem clic dret sobre la carpeta **/main/res** del projecte i seleccionem **New/Image Asset**.

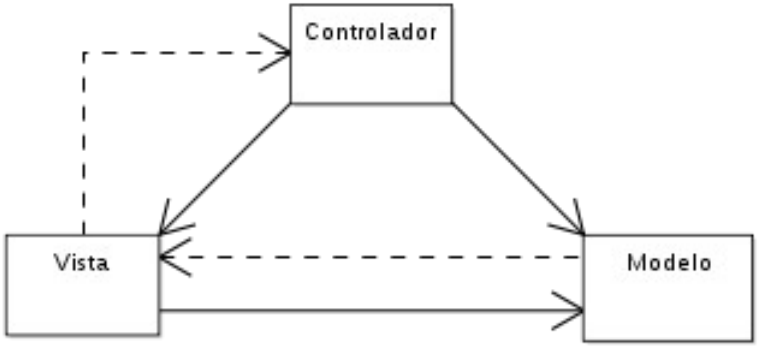


Afegir imatges a Android Studio (II)

- Des de l'Asset **Studio** podem indicar:
  - El **tipus d'imatge** a afegir
    - Icona de llançador (Launcher Icons)
    - Icona de la barra d'accions (ActionBar) i pestanyes (Tab)
    - Icona per a notificacions (Notification Icons)
  - L'origen **de la imatge**
    - Arxiu d'imatge extern (Image)
    - Col·lecció d'icones estandar (Clipart)
    - Text personalitzat (Text)
- Una vegada seleccionades les opcions corresponents, **Asset Studio crearà automàticament** la **icona** per a les **diferents densitats** de píxels i col·locar-lo en la seua carpeta de recursos corresponent.

Model – Vista – Controlador (MVC)

- El model–vista–controlador (**MVC**) és un **patró** d'arquitectura de programari que **separa** les **dades i la lògica de negoci** (**model**) d'una aplicació de la **interfície d'usuari** (**vista**) i el **mòdul encarregat de gestionar els esdeveniments i les comunicacions** (**controlador**).
- En **Android** la vista i el **controlador estan** clarament **separats**. Les **vistes** són dissenyades en el llenguatge **xml**, mentre que els **controladors** estan escrits en **Java**.
- Android deixa que el desenvolupador decidisca si vol **separar els models** (**opció recomanada**) de les **vistes** i els **controladors**, ja que aporten una major independència el que permet reutilitzar més codi i a més ho fa més fàcil de mantindre.



Unint vistes i controladors

- En Android la unió de la vista (xml de l'Activity) amb el controlador (classe Java que descendeix d'Activity) es realitza mitjançant codi Java.
- La classe **Activity** disposa del mètode **setContentView()** que rep com a **paràmetre** el **layout** de l'activity.
- Per exemple, suposant que en /res/layout tenim el layout d'una **Activity** anomenat **activity_main.xml**, des de l'Activity Java invocaríem al mètode setContentView de la següent forma per a indicar-li a l'Activity quin a de ser el seu contingut visual:

  setContentView(R.layout.activity_main);

- La invocació a aquest mètode ha de realitzar-se en el mètode onCreate, immediatament després de l'anomenada al constructor pare super.onCreate(savedInstanceState);

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

Esdeveniments d'un botó (I)

- Els botons són capaços de llançar molts esdeveniments, encara que el més comú de tots ells és l'esdeveniment **onClick**, que es llança cada vegada que el botó és premut.
- Per a definir la lògica de l'esdeveniment onClick hem d'implementar-la definint un nou objecte **View.OnClickListener()** i associar-ho al botó mitjançant el mètode **setOnClickListener()**.

```
btnBotonSimple = (Button)findViewById(R.id.BtnBotonSimple);

btnBotonSimple.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0)
    {
        lblMensaje.setText("Botón Simple pulsado!");
    }
});
```

Esdeveniments d'un botó (II)

- En el cas d'un botó de tipus **ToggleButton** o **Switch** sol ser d'utilitat conèixer en quin estat ha quedat el botó després de ser premut, per al que podem utilitzar el seu mètode **isChecked()**.
- Veiem un exemple en el qual es comprova l'estat del botó després de ser premut i es realitzen accions diferents segons el resultat.

```
btnToggle = (ToggleButton)findViewById(R.id.BtnToggle);

btnToggle.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0)
    {
        if(btnToggle.isChecked())
            lblMensaje.setText("Botón Toggle: ON");
        else
            lblMensaje.setText("Botón Toggle: OFF");
    }
});
```

TextView (I)

- Els **TextView** o etiquetes de text són utilitzades per a mostrar un determinat text a l'usuari.
- Igual que els botons, el text a mostrar s'estableix mitjançant la propietat **android:text**.
- Altres propietats interessants són **android:background** (color de fons), **android:textColor** (color del text), **android:textSize** (grandària de la font) i **android:typeface** (estil del text: negreta, cursiva, etc.)
- Veámos un exemple:

```
<TextView android:id="@+id/LblEtiqueta"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/escribe_algo"
    android:background="#ff1ca5ff"
    android:typeface="monospace"/>
```

TextView (II)

- Podem **manipular** les **propietats des de** codi **Java**.
- En el següent exemple recuperem el text de l'etiqueta amb un **getText()**, i posteriorment li concatenem uns números, actualitzem el seu contingut mitjançant un **setText()** i li canviem el seu color de fons amb **setBackgroundColor()**:

```
final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);
String texto = lblEtiqueta.getText().toString();
texto += "123";
lblEtiqueta.setText(texto);
lblEtiqueta.setBackgroundColor(Color.BLUE);
```

EditText

- L'**EditText** és el component d'edició de text que proporciona la plataforma Android. Permet la **introducció i edició de text** per part de l'usuari.
- Com a propietats més importants tenim **android:text** (**text** per defecte), **android:inputType** (**tipus de contingut** ej: number, text, password, etc.) que tindrà efecte en el tipus de teclat que mostrarà Android per a editar aquest camp.
- Igual que amb els botons, podem indicar les propietats **android:drawableLeft** i **android:drawableRigh**t que ens permeten afegir una **imatge** a l'esquerra o dreta.
- Una altra opció addicional serà indicar un **text d'ajuda o descripció** (hint), que apareixerà en el quadre de text mentre l'usuari no haja escrit res (com a s'escriu alguna cosa aquest text desapareix). Per a això utilitzarem les propietats **android:hint** per a indicar el text i **android:textColorHint** per a indicar el seu color.

Exemple EditText

```
<EditText android:id="@+id/TxtImagenHint"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:drawableLeft="@drawable/ic_usuario"
    android:hint="@string/usuario"
    android:textColorHint="#CFCFCF"
    android:inputType="text" />
```


- Per a recuperar i establir text des de codi Java podem utilitzar els mètodes **getText()** i **setText()**.

```
EditText txtTexto = (EditText)findViewById(R.id.TxtBasico);
String texto = txtTexto.getText().toString();
txtTexto.setText("Hola mundo!");
```

TextInputLayout (I)

- Les **etiquetes flotants** o **TextInputLayout** també formen part de la **Design Support Library**, que com ja hem comentat, ens permet seguir les especificacions de **Material Design** en dispositius amb versions inferiors a Android 5.0.
- Una **etiqueta flotant** no és més que un hint que en lloc de desaparèixer, **es desplaça automàticament** a la part superior del quadre de text quan l'usuari prem sobre ell.
- Com ja hem vist per a **importar la llibreria de suport per a diseny** al nostre projecte, hem d'afegir una referència al fitxer **build.gradle**. Si ja la tenim importada no és necessari no és necessari tornar a fer-ho.

```
dependencies {
    ...
    compile 'com.android.support:design:27.1.1'
}
```

TextInputLayout (II)

- Per a crear una etiqueta flotant hem d'afegir un **EditText** amb **hint** dins d'un contenidor **TextInputLayout**.

```
<android.support.design.widget.TextInputLayout
    android:id="@+id/TiLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <EditText android:id="@+id/TxtInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Escribe un número par" />

</android.support.design.widget.TextInputLayout>
```

TextInputLayout (III)

- Una altra característica destacada d'aquest component és la possibilitat de **mostrar errors**, molt útils en **la validació de formularis**, davall del quadre de text.
- Per a això podem utilitzar els mètodes **setErrorEnabled(true)** que reservarà espai davall del quadre de text per a mostrar els errors, i **setError()** per a indicar el text de l'error o per a eliminar-lo (passant null com a paràmetre).
- Veiem en la següent diapositiva un exemple on es comprova si el número introduït per l'usuari és un nombre parell, mostrant un error en cas de no ser-ho.

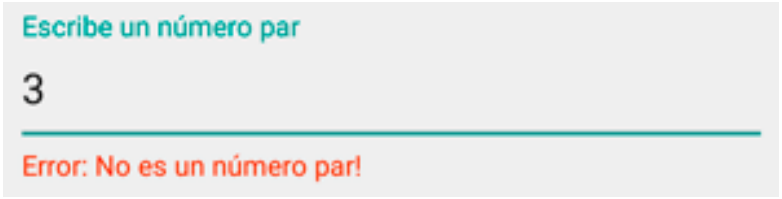
Exemple TextInputLayout

```
txtInputLayout = (TextInputLayout)findViewById(R.id.TiLayout);
txtInputLayout.setErrorEnabled(true);

txtInput = (EditText)findViewById(R.id.TxtInput);

btnComprobar = (Button)findViewById(R.id.BtnInputLayout);
btnComprobar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String num = txtInput.getText().toString();

        if(num.isEmpty() || Integer.parseInt(num)%2 != 0)
            txtInputLayout.setError("Error: No es un número par!");
        else
            txtInputLayout.setError(null);
    }
});
```



ImageView

- El control **ImageView** permet **mostrar imatges** en l'aplicació. La propietat més interessant és **android:src**, que permet indicar la **imatge a mostra**r, el valor de la qual serà l'identificador d'un recurs de la nostra carpeta **/res/drawable**, per exemple android:src="@drawable/unaimagen".
- Altres propietats útils són **android:maxWidth** i **android:maxHeight** que permeten indicar la **grandària màxima** que pot ocupar la imatge.
- A més, com ja hem comentat en els controls ImageButton, en tractar-se d'un control de tipus imatge hauríem d'establir sempre la propietat **android:contentDescription** per a oferir una **breu descripció textual** de la imatge, així la nostra aplicació serà més accessible.

Exemple ImageView

```
<ImageView android:id="@+id/ImgFoto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_launcher"
    android:contentDescription="@string/imagen_ejemplo" />
```

- Per a **establir la imatge** a mostrar **des de codi** podem utilitzar el mètode **setImageResource()**, passant-li l'ID del **recurs** a utilitzar com a imatge.

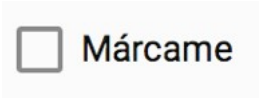
```
ImageView img= (ImageView)findViewById(R.id.ImgFoto);
img.setImageResource(R.drawable.ic_launcher);
```

CheckBox

- El control **CheckBox** sol utilitzar-se per a **marcar o desmarcar opcions**.
- La forma de definir-ho i els mètodes disponibles són anàlegs als ja comentats per al control **ToggleButton**.
- La propietat booleana **android:checked** permet inicialitzar l'estat del control, "true" **marcat** i "false" **desmarcat**. Si no especifiquem aquesta propietat, per defecte apareixerà desmarcat.

Exemple CheckBox (I)

```
<CheckBox android:id="@+id/ChkMarcame"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/marcame"
    android:checked="false" />
```



- Com un **CheckBox** estén indirectament al control **TextView**, totes les opcions de format del TextView també són vàlides per a aquest control.
- Des de codi Java podem **consultar l'estat** mitjançant el mètode **isChecked()** que retornarà un valor booleà i **canviar-lo** mitjançant **setChecked()** que acceptarà com a paràmetre un valor booleà que representarà el nou estat.
- Entre els esdeveniments que pot llançar aquest control el més destacat és **onClick**, ja que ens indicarà quan ha sigut premuda la casella de selecció.
- Vegem en la següent diapositiva un exemple complet en Java on s'utilitza l'esdeveniment onClick i el mètode per a consultar l'estat.

Exemple CheckBox (II)

```
cbMarcame = (CheckBox)findViewById(R.id.ChkMarcame);

cbMarcame.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        boolean isChecked = ((CheckBox)view).isChecked();

        if (isChecked) {
            cbMarcame.setText("Checkbox marcado!");
        }
        else {
            cbMarcame.setText("Checkbox desmarcado!");
        }
    }
});
```

Exemple CheckBox (III)

- Un altre esdeveniment que pot resultar útil és **onCheckedChanged**, que ens permet **saber quan ha canviat d'estat** el control. Per a implementar-ho hem de crear un listener del tipus CheckBox.**OnCheckedChangeListener()**.

```
cbMarcame = (CheckBox)findViewById(R.id.ChkMarcame);

cbMarcame.setOnCheckedChangeListener(new CheckBox.OnCheckedChangeListener()
    public void onCheckedChanged(CompoundButton buttonView, boolean isCheck
        if (isChecked) {
            cbMarcame.setText("Checkbox marcado!");
        }
        else {
            cbMarcame.setText("Checkbox desmarcado!");
        }
    }
});
```

RadioButton (I)

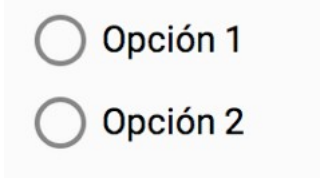
- Igual que el control **CheckBox**, un **RadioButton** pot estar marcat o desmarcat, però solen utilitzar-se dins d'un grup d'opcions mútuament excloents entre si, és a dir, només una pot ser marcada. Si es marca una de les opcions es desmarcarà automàticament la que estiguera activa.
- Per a això tots els radiobutton que formen part del mateix grup d'opcions hauran de definir-se dins d'un **RadioGroup** que com a propietat més interessant té **android:orientation** que indicarà si els radiobutton seran mostrats en “horizontal” o en “vertical”.

Exemple RadioButton (I)

```
<RadioGroup android:id="@+id/GrbGrupo1"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <RadioButton android:id="@+id/RbOpcion1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/opcion_1" />

    <RadioButton android:id="@+id/RbOpcion2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/opcion_2" />
</RadioGroup>
```



RadioButton (II)

- Una vegada definida la interfície podrem manipular el control des del nostre codi java fent ús dels diferents **mètodes** del control **RadioGroup**, els més importants: **check(id)** per a **marcar una opció** determinada mitjançant el seu **id**, **clearCheck()** per a **desmarcar totes** les opcions, i **getCheckedRadioButtonId()** que com el seu nom indica **retornarà l'id** de l'opció marcada (o el valor -1 si no hi ha cap marcada).

```
RadioGroup rg = (RadioGroup)findViewById(R.id.GrbGrupo1);
rg.clearCheck();
rg.check(R.id.RbOpcion1);
int idSeleccionado = rg.getCheckedRadioButtonId();
```

RadioButton (III)

```
lblMensaje = (TextView)findViewById(R.id.LblSeleccion);
rbOpcion1 = (RadioButton)findViewById(R.id.RbOpcion1);
rbOpcion2 = (RadioButton)findViewById(R.id.RbOpcion2);

View.OnClickListener list = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String opcion = "";
        switch(view.getId()) {
            case R.id.RbOpcion1:
                opcion = "opción 1";
                break;
            case R.id.RbOpcion2:
                opcion = "opción 2";
                break;
        }

        lblMensaje.setText("ID opción seleccionada: " + opcion);
    }
};

rbOpcion1.setOnClickListener(list);
rbOpcion2.setOnClickListener(list);
```

- Quant als esdeveniments llançats, recorrerem novament a l'esdeveniment **onClick** per a saber quan es prem cadascun dels botons del grup.
- Normalment utilitzarem un **mateix listener** per a tots els radiobutton del grup, per la qual cosa el definirem de forma independent i després l'assignarem a tots els botons.

RadioButton (IV)

- Igual que en el cas del CheckBox, també podrem utilitzar l'esdeveniment **onCheckedChange**, que ens informarà dels canvis en l'element seleccionat dins d'un grup. La diferència és que aquest **esdeveniment** està **associat al RadioGroup**, i no als diferents RadioButton del grup.

```
rgOpciones = (RadioGroup)findViewById(R.id.GrbGrupo1);
rgOpciones.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener()
    public void onCheckedChanged(RadioGroup group, int checkedId) {

        String opcion = "";
        switch(checkedId) {
            case R.id.RbOpcion1:
                opcion = "opción 1";
                break;
            case R.id.RbOpcion2:
                opcion = "opción 2";
                break;
        }

        lblMensaje.setText("ID opción seleccionada: " + opcion);
    }
});
```