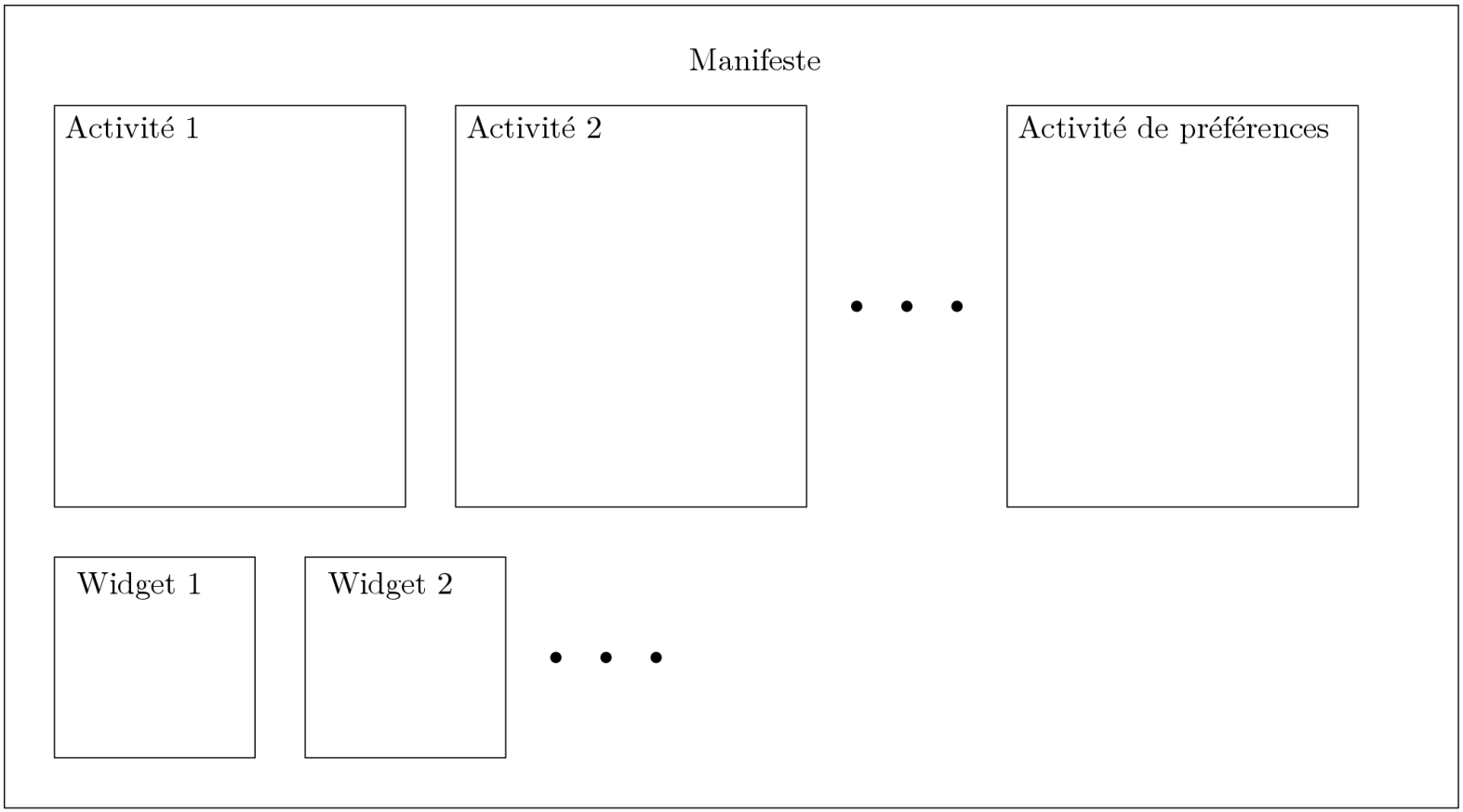
Activités

Chantal Keller

Architecture d'une application



Architecture d'une application

Description de l'application : le manifeste (XML)

- les différents activités et widgets
- quelle activité lancer au démarrage de l'application
- permissions : accès au réseau, au carnet d'adresse, ...
- possibilité de diffuser des messages aux autres applications

Composants :

- activités (au moins une)
- activité de préférences
- widgets

↪ aujourd'hui : fonctionnement d'une activité

Activité

“Page” de l’application :

- interface (vue) en XML
- à laquelle on donne vie *via* l’API Java (contrôleur et modèle)

MVC : trois fonctions **séparées le plus possible**

- les données **ne dépendent pas** de l’affichage
- elles doivent être stockées par le modèle

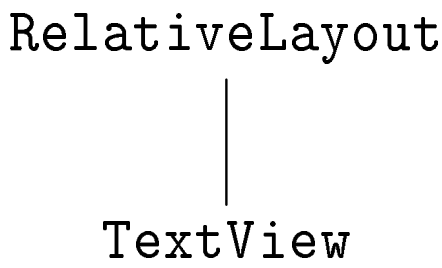
Activités : vue

Description de l'interface d'une activité en XML :

- description de chaque objet : couleur, taille, fonte, ...
- placement des objets : texte, image, boutons, ...
- imbrication les uns dans les autres : "boîtes"

Exemple

```
<RelativeLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="16dp"
  android:paddingRight="16dp"
  android:paddingTop="16dp"
  android:paddingBottom="16dp">

  <TextView
    android:text="Hello world!"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</RelativeLayout>
```



Exemples d'objets les plus répandus

Les objets de base :

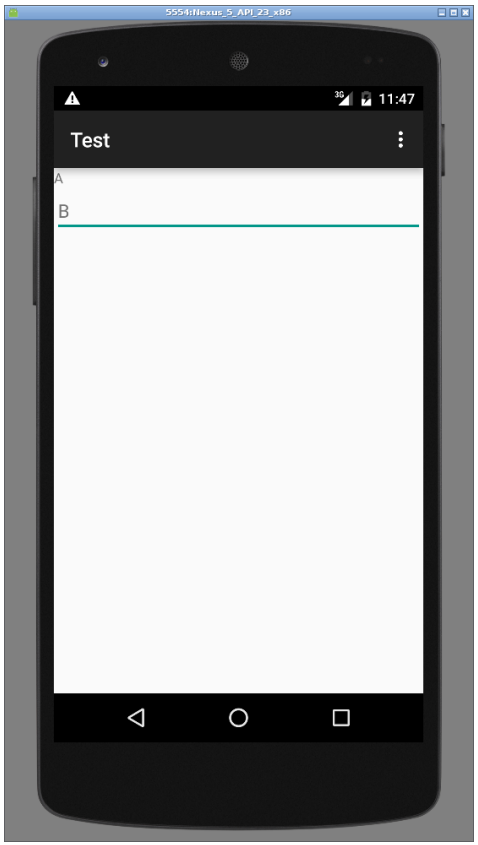
- zones de texte (saisie ou non), boutons, images
- attributs : contenu, taille, couleur, ...
- + un identifiant

Les boîtes :

- autres noms : *ViewGroup*, *layout*
- contiennent d'autres boîtes
- le choix du *layout* détermine la manière de décrire la position de ses fils
- Autres : zones de texte, boutons, images, ...

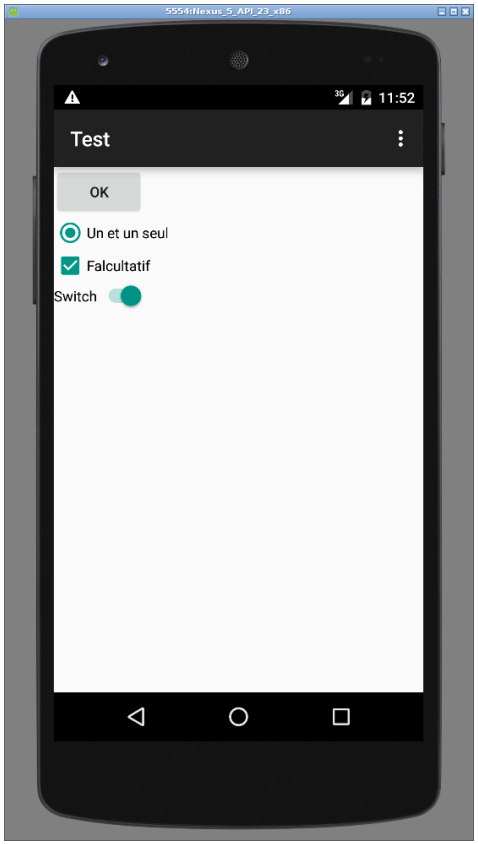
Zone de texte

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="A"
  android:id="@+id/textViewA" />

<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:hint="B"
  android:id="@+id/editTextB" />
```



Boutons

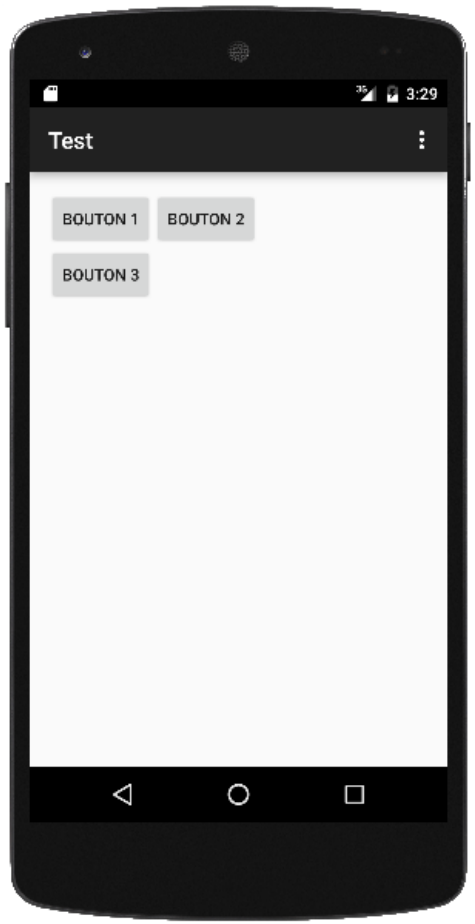
```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="OK"
  android:id="@+id/buttonOK" />

<RadioButton
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Un et un seul"
  android:id="@+id/radioButtonUn" />

<CheckBox
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Falcultatif"
  android:id="@+id/checkBoxFac" />

<Switch
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Switch"
  android:id="@+id/switchS" />
```



RelativeLayout

Place les fils les uns par rapport aux autres

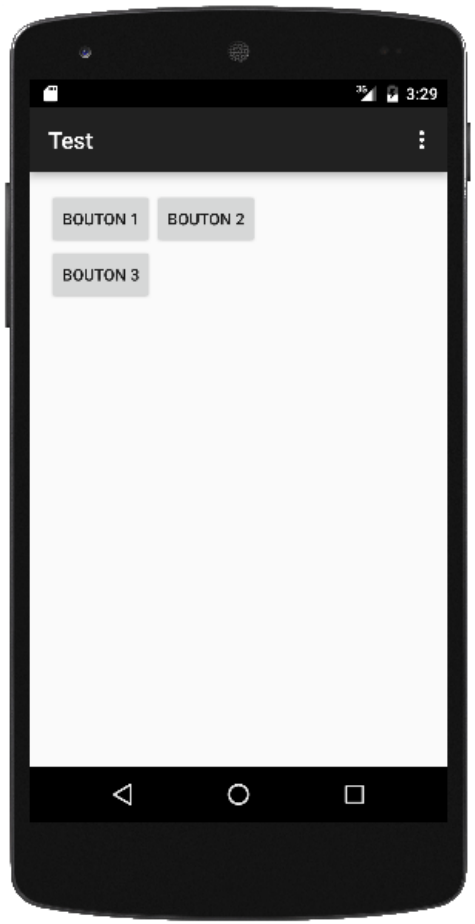
```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 1"
        android:id="@+id/button1"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 2"
        android:id="@+id/button2"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@id/button1"
        android:layout_toEndOf="@id/button1" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 3"
        android:id="@+id/button3"
        android:layout_below="@id/button1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```



GridLayout

Place les objets sur une grille

```
<GridLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Bouton 1"
      android:id="@+id/button1"
      android:layout_row="0"
      android:layout_column="0" />

  <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Bouton 2"
      android:id="@+id/button2"
      android:layout_row="0"
      android:layout_column="1" />

  <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Bouton 3"
      android:id="@+id/button3"
      android:layout_row="1"
      android:layout_column="0" />
</GridLayout>
```



Boîte adaptative : `LinearLayout`

Décrit la position des fils par rapport à la boîte :

- être le plus petit possible (`wrap_content` : taille du contenu)
- occuper toute la largeur, toute la hauteur (`match_parent` : taille du contenant)
- occuper un pourcentage de la largeur ou de la hauteur (**poids**)

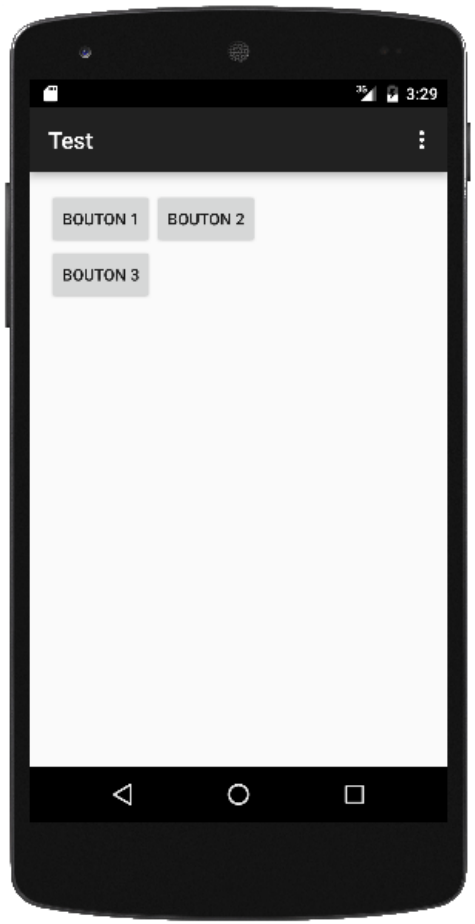
Préférer `LinearLayout` pour la programmation mobile :

- **s'adapte** à la forme et la taille de l'écran

LinearLayout : utilisation de wrap_content

Aligne les fils dans une même direction

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bouton 1"
            android:id="@+id/button1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bouton 2"
            android:id="@+id/button2" />
    </LinearLayout>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 3"
        android:id="@+id/button3" />

</LinearLayout>
```



LinearLayout : utilisation de match_parent

Aligne les fils dans une même direction

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bouton 1"
            android:id="@+id/button1" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bouton 2"
            android:id="@+id/button2" />
    </LinearLayout>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bouton 3"
        android:id="@+id/button3" />

</LinearLayout>
```

LinearLayout : utilisation des poids

Aligne les fils dans une même direction

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Bouton 1"
            android:id="@+id/button1"
            android:layout_weight="1"/>
        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Bouton 2"
            android:id="@+id/button2"
            android:layout_weight="2"/>
    </LinearLayout>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bouton 3"
        android:id="@+id/button3" />
```

Conclusion sur la vue

Interface en XML :

- feuilles : objets de base, attributs pour les caractéristiques, identifiant
- nœuds : *layouts*, choisis selon la manière dont on veut agencer le contenu ; **préférer** `LinearLayout`

Activité : contrôleur et modèle

Animer l'interface d'une activité :

- réaction au clic sur un bouton
- affichage de l'interface au lancement de l'activité
- autres actions au lancement de l'activité
- réaction à un changement d'orientation de la tablette
- ...

Contrôleur d'une activité grâce à la POO

Rappel :

- définir une classe (Java) **héritant** de la classe `Activity`
- **redéfinir** les méthodes appelées lors des évènements prédéfinis (ex : création de l'activité $\Rightarrow$ appel à la méthode `onCreate`)
- **associer** des objets (écouteurs) aux nouveaux évènements (ex : clic sur un bouton)

Héritage de `Activity`

```
public class MainActivity extends AppCompatActivity {

    // ... méthodes à définir ou redéfinir

}
```

Remarques :

- le contrôleur d'une activité est une classe héritant de `Activity`
- pour plus de fonctionnalités, hériter de `AppCompatActivity` (qui elle-même hérite de `Activity`) ou d'une autre classe fille

Redéfinition de méthodes : exemple de création de l'activité

```
public class MainActivity extends AppCompatActivity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_xml);
        System.out.println(``Activity created'');
    }

    // ... autres méthodes à définir ou redéfinir

}
```

Lorsqu'une activité est créée :

- `onCreate` est appelée par le système
- dans l'exemple ci-dessus :
  - appel à la méthode de la super-classe (toujours)
  - affichage de l'interface graphique (toujours)
  - autres (notamment, récupération des objets de l'interface)

Objets de l'interface : exemple d'un bouton

```
public class MainActivity extends AppCompatActivity {

    private Button button;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_xml);
        button = (Button) findViewById(R.id.button);
    }

    // ... autres méthodes à définir ou redéfinir

}
```

Récupération des objets de l'interface :

- dans `onCreate`, après avoir lancé la création de l'interface, dans des variables globales du programme
- utilisation/modification quand on veut
- `findViewById` : utilise l'identifiant choisi dans le XML
- `(Button)` : pour avoir un objet de type `Button`

Nouvel évènement : exemple du clic sur un bouton

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    private Button button;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_xml);
        button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    public void onClick(View v) {
        if (v.getId() == R.id.button) {
          // action à définir
        }
    }

    // ... autres méthodes à définir ou redéfinir

}
```

Rappel : écouteur sur le bouton

Conclusion sur le contrôleur et le modèle

Contrôleur d'activité :

- classe Java héritant d'`Activity` (*via* `AppCompatActivity`)
- redéfinition de méthodes associées aux évènements prédéfinis : leurs noms commencent souvent par `on` (ex. `onCreate`, appelée lors de la création de l'activité, qui charge l'interface et récupère les objets de l'interface)
- gestion des nouveaux évènements grâces à des écouteurs (*listeners*)