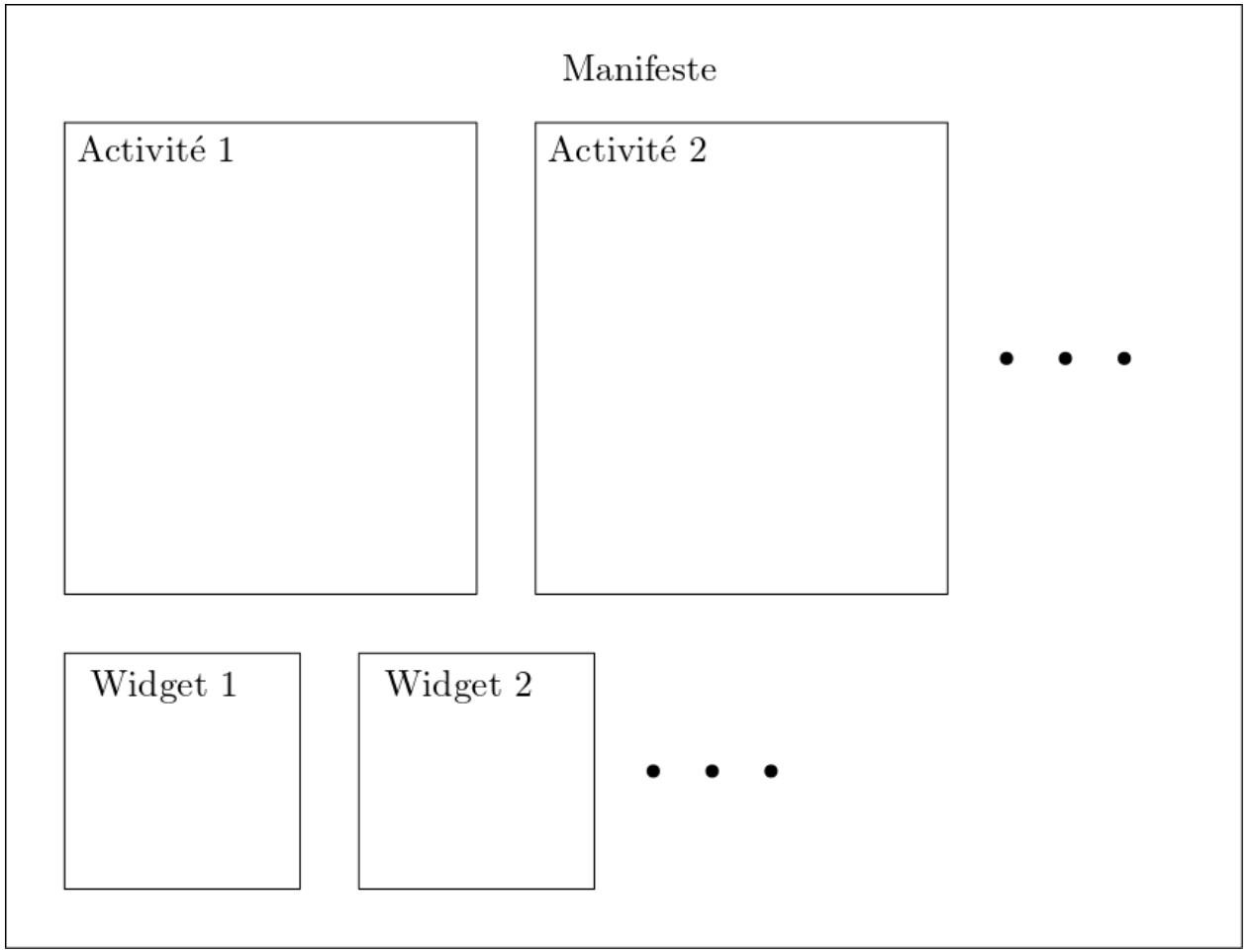
Activités

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)
- widgets (facultatifs)

↪ 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)

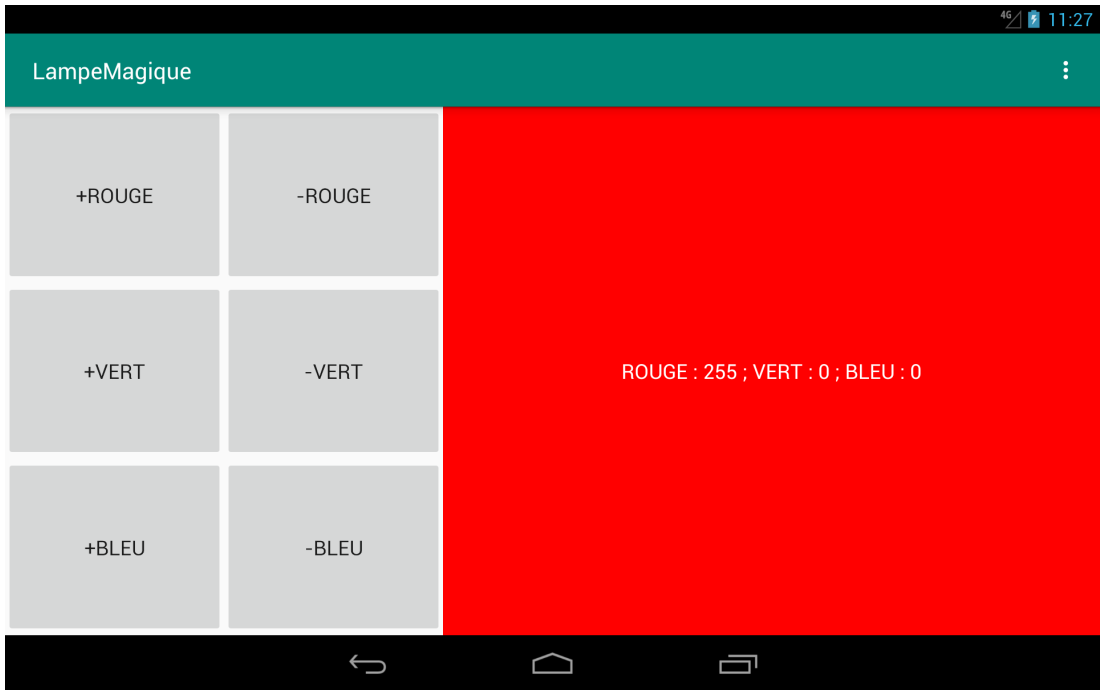
MVC : trois fonctions **séparées le plus possible**

- 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 :

- nature des objets : texte, image, bouton, ...
- description des objets : couleur, taille, texte, fonte, ...
- placement des objets : les uns par rapports aux autres, imbrication, ...



Objets (*View*)

Objets de base (feuilles) :

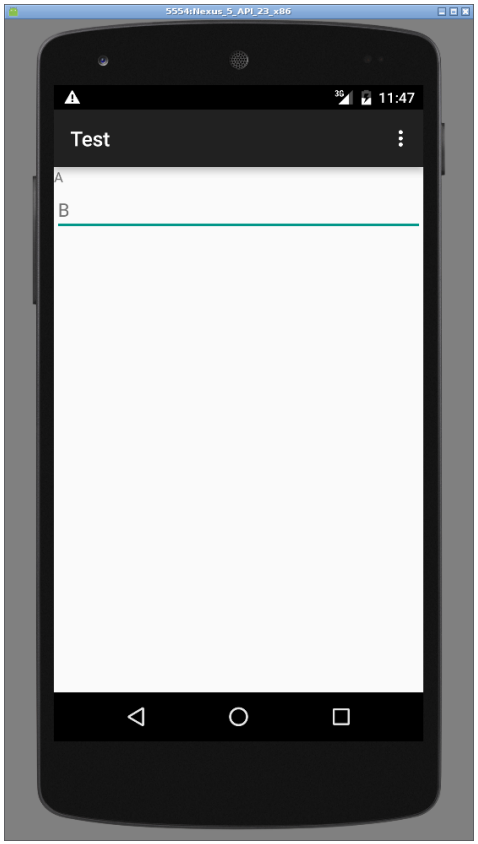
- zones de texte (saisie ou non), boutons, images
- attributs : contenu, taille, couleur, ... + un identifiant

Imbrication dans des boîtes (nœuds) :

- noms en Android : *ViewGroup*, *layout*
- contiennent des objets de bases et d'autres boîtes
- le choix du *layout* détermine la manière de décrire la position de ses éléments

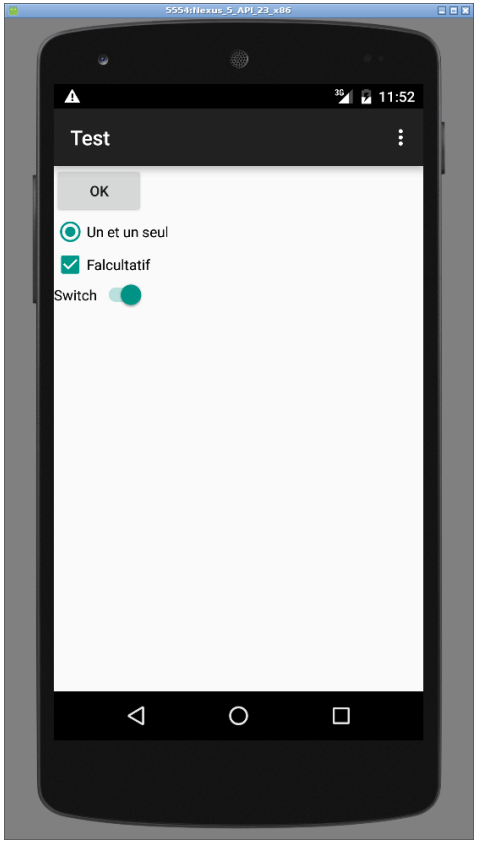
Objet de base : zone de texte

```
<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" />
```

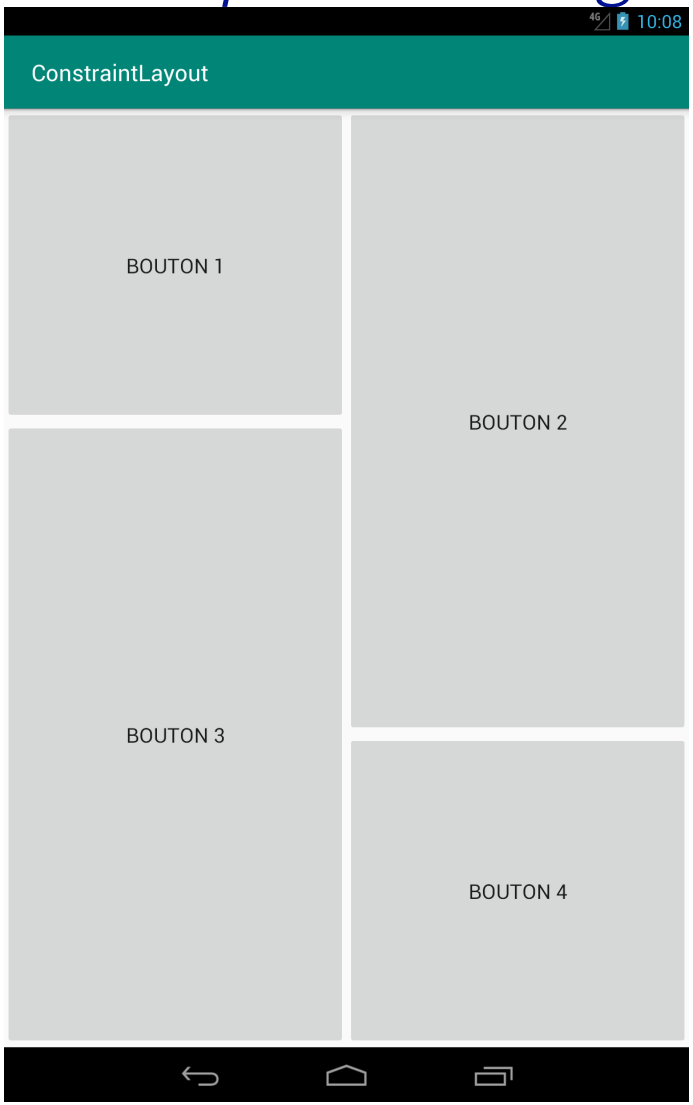


Objet de base : bouton

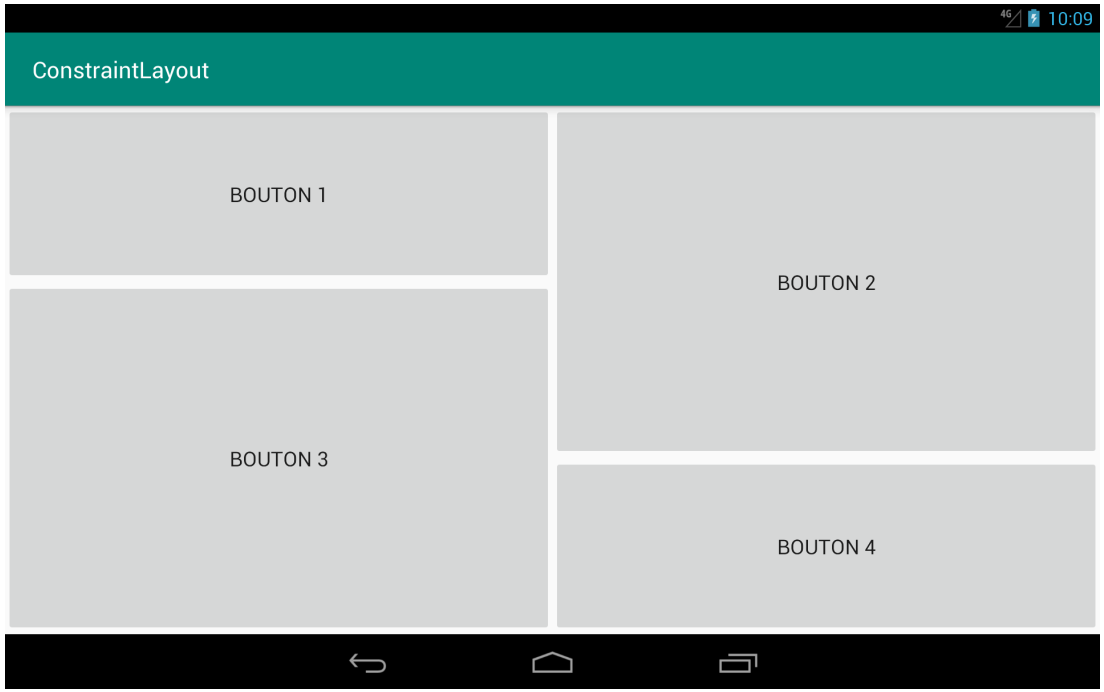
```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="OK"
  android:id="@+id/buttonOK" />

<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" />
```



Placement des objets : *responsive design*



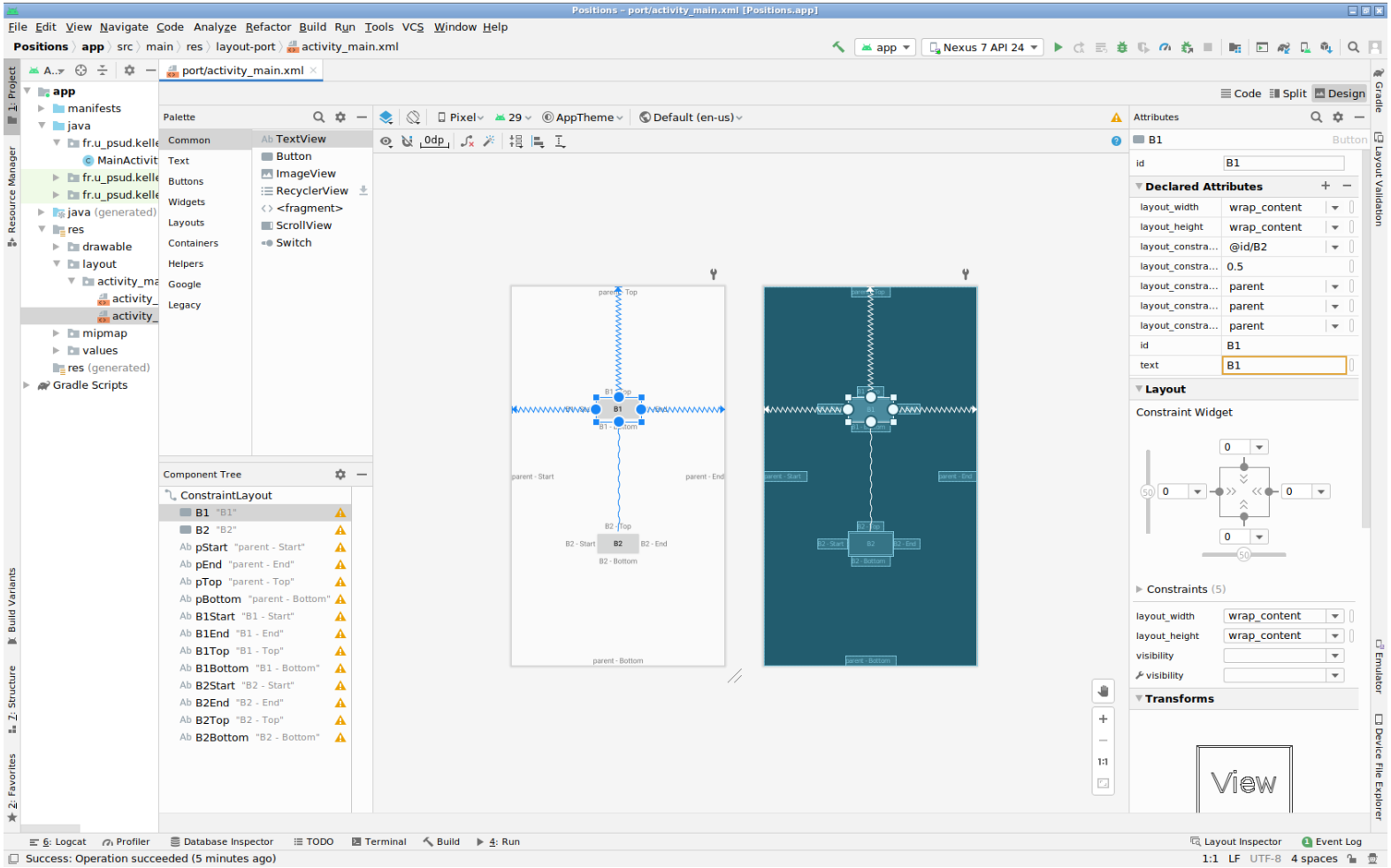
Placement des objets : *responsive design*



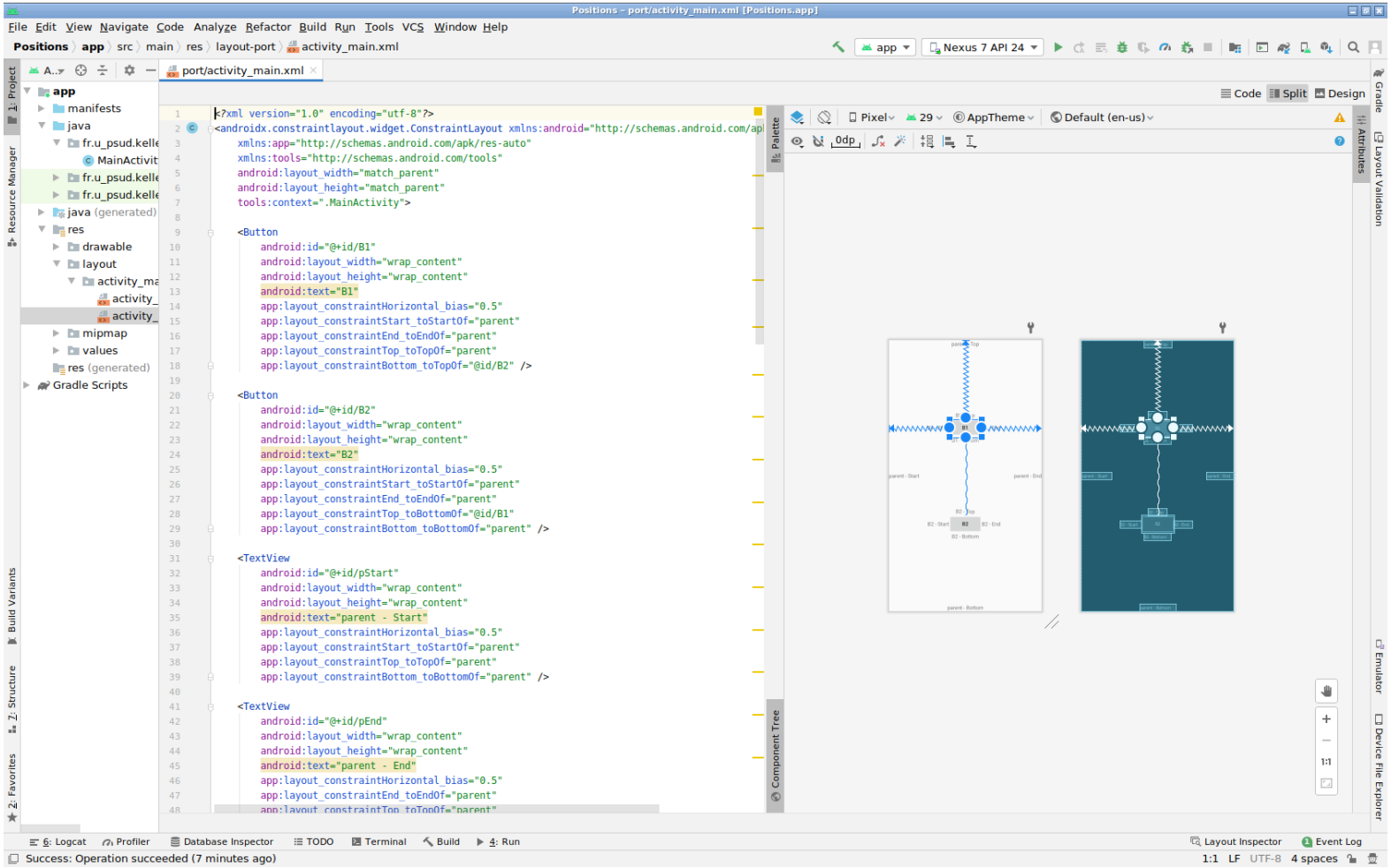
Boîte la plus courante : *ConstraintLayout*

- un seul nœud : ConstraintLayout
- à l'intérieur, les fils sont contraints les uns par rapport aux autres ou par rapport au parent
- XML + support graphique dans AndroidStudio

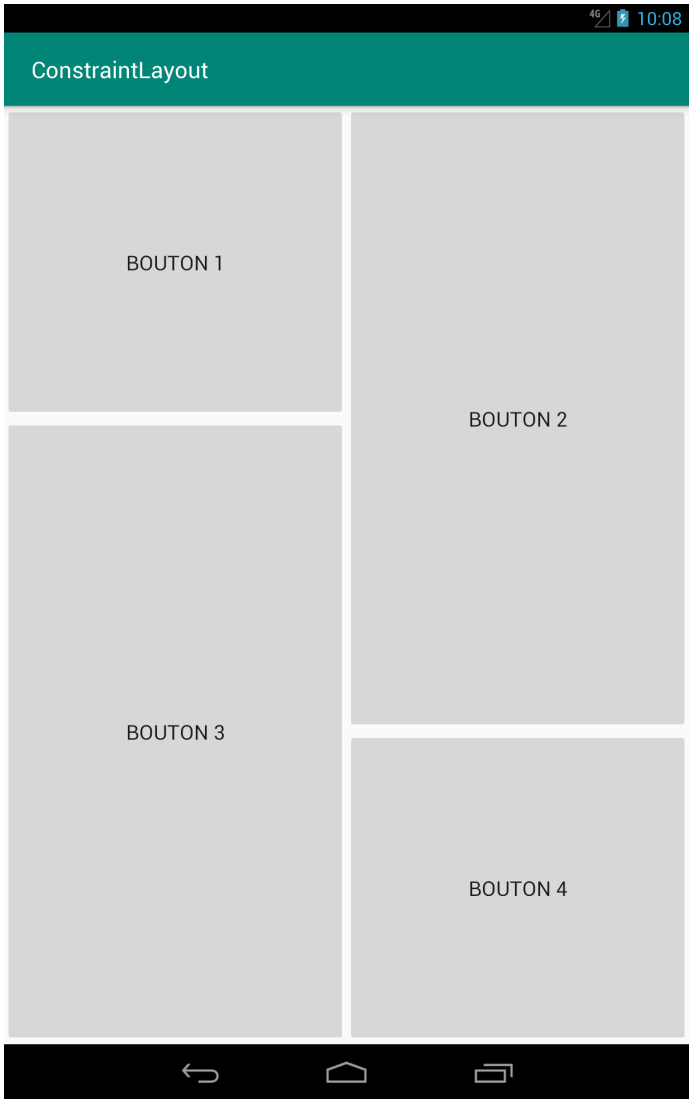
AndroidStudio : support graphique



AndroidStudio : XML

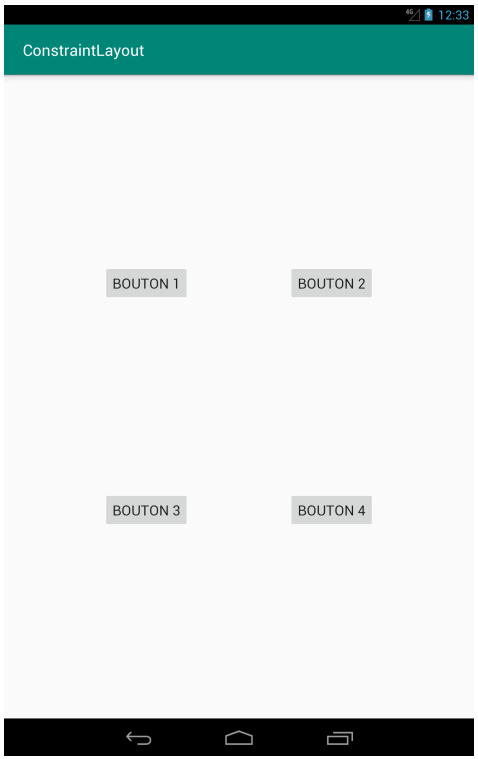


Pas à pas



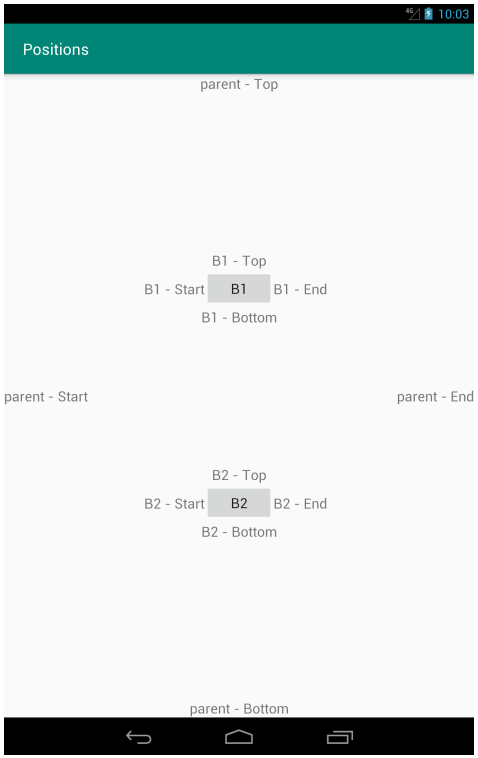
1. Placement des objets

```
<androidx.constraintlayout.widget.ConstraintLayout>

    <Button
        android:id="@+id/bouton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/bouton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 2"
        app:layout_constraintStart_toEndOf="@id/bouton1"
        app:layout_constraintTop_toTopOf="parent" />

    ...

</androidx.constraintlayout.widget.ConstraintLayout>
```



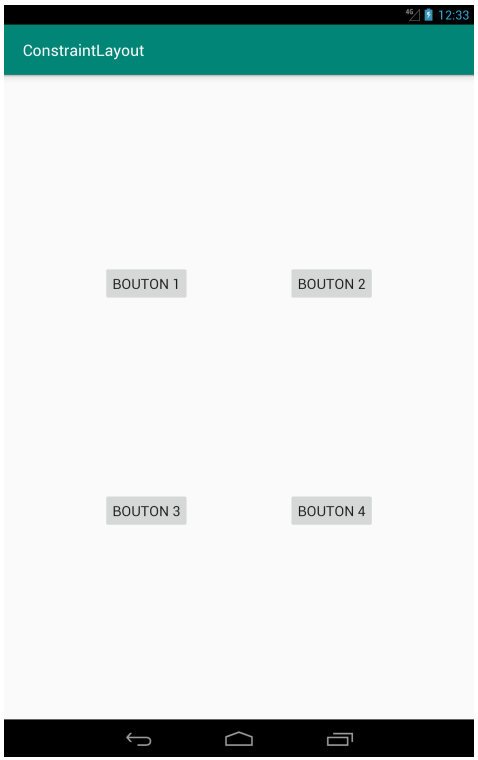
Précisions sur les positions

```
<androidx.constraintlayout.widget.ConstraintLayout>

    <Button
        android:id="@+id/B1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/B2" />

    <Button
        android:id="@+id/B2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B2"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/B1"
        app:layout_constraintBottom_toBottomOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



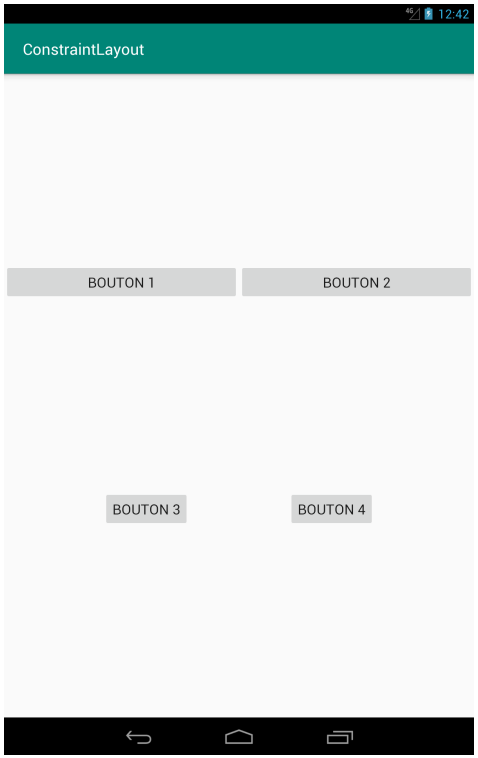
2. Chaîne : les deux sens des contraintes

```
<androidx.constraintlayout.widget.ConstraintLayout>

    <Button
        android:id="@+id/bouton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/bouton2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/bouton3" />

    <Button
        android:id="@+id/bouton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bouton 2"
        app:layout_constraintStart_toEndOf="@id/bouton1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/bouton4" />

    ...

</androidx.constraintlayout.widget.ConstraintLayout>
```

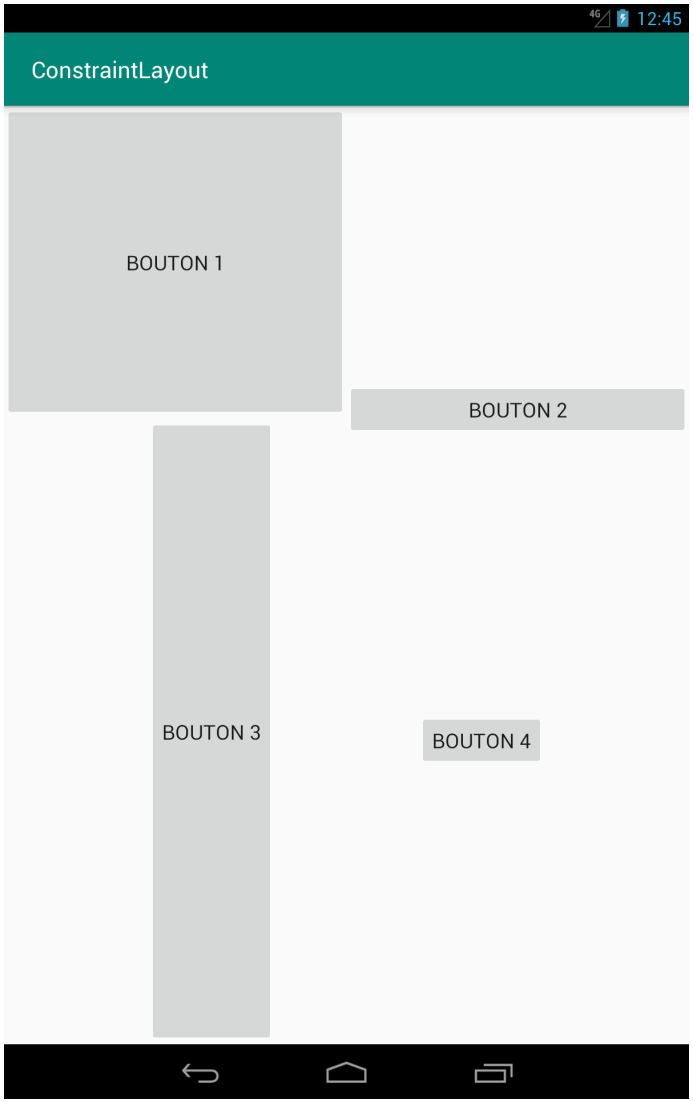


3. Réglage des dimensions : **poids**

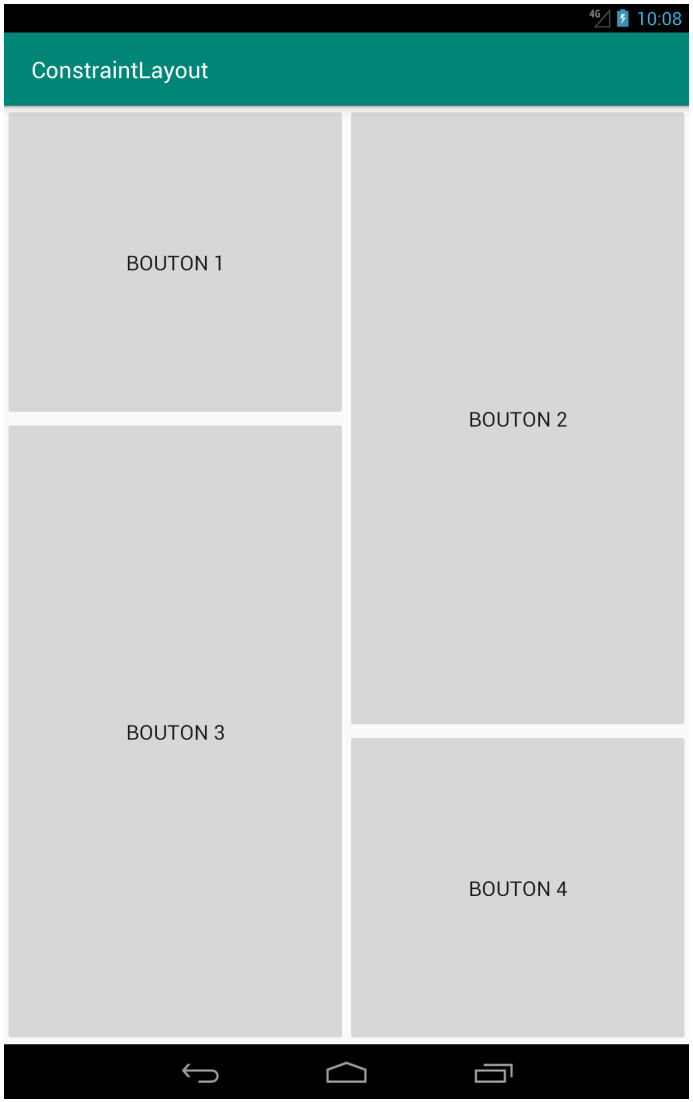
```
<androidx.constraintlayout.widget.ConstraintLayout>

    <Button
        android:id="@+id/bouton1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintHorizontal_weight="1"
        android:text="Bouton 1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toStartOf="@id/bouton2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/bouton3" />

    <Button
        android:id="@+id/bouton2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintHorizontal_weight="1"
        android:text="Bouton 2"
        app:layout_constraintStart_toEndOf="@id/bouton1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@id/bouton4" />

    ...

</androidx.constraintlayout.widget.ConstraintLayout>
```



Exercice



Exercice à la maison



Conclusion sur la vue

Interface en XML :

- feuilles : objets de base, attributs pour les caractéristiques, identifiant
- nœuds : *layouts*, *ConstraintLayout* est très souvent approprié
- 3 possibilités pour les tailles (*layout_width* et *layout_height*) :
  - *wrap_content* : la taille minimale pour le contenu
  - *match_parent* : la taille totale de la fenêtre
  - *0dp* + poids (*layout_constraintHorizontal_weight* ou *layout_constraintVertical_weight*) : taille calculée par rapport aux autres objets de la chaîne

jamais de taille absolue

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 = 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

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 = 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*)