Chapitre IV :
Interface utilisateur (UI)

Introduction

- **L'Interface Utilisateur (UI)** dans Android désigne les composants visuels et interactifs d'une application Android qui permettent aux utilisateurs d'interagir avec l'application et d'effectuer diverses tâches.
- L'UI est une partie essentielle de toute application Android, car elle détermine la manière dont les utilisateurs interagissent avec l'application et comment ils percevront sa fonctionnalité et sa facilité d'utilisation.
- L'interface utilisateur (UI) d'Android est guidée par les principes de Material Design de Google, qui fournissent des normes pour concevoir des interfaces claires et attrayantes sur tous les appareils Android.
- En suivant ces directives, les développeurs créent des expériences utilisateur harmonieuses et esthétiquement engageantes, facilitant une interaction intuitive et plaisante.

Layouts

- **Les layouts** sont un élément crucial de la conception de l'interface utilisateur (UI) Android. Un layout détermine la structure visuelle de l'interface utilisateur dans une application Android, comme l'emplacement des éléments UI. Autrement dit, les layouts définissent l'apparence de l'application.
- Android propose une variété de types de layouts parmi lesquels les développeurs peuvent choisir pour concevoir l'UI de leur application. Chaque layout possède ses propres propriétés uniques, et choisir le bon layout peut grandement influencer l'expérience utilisateur globale. Comprendre les différents types de layouts et savoir comment les utiliser efficacement est essentiel pour créer des applications bien conçues et faciles à utiliser

Les types de Layouts

Il existe plusieurs types de layouts disponibles dans Android, chacun ayant ses propres caractéristiques uniques et cas d'utilisation prévus. Voici quelques-uns des layouts les plus couramment utilisés :

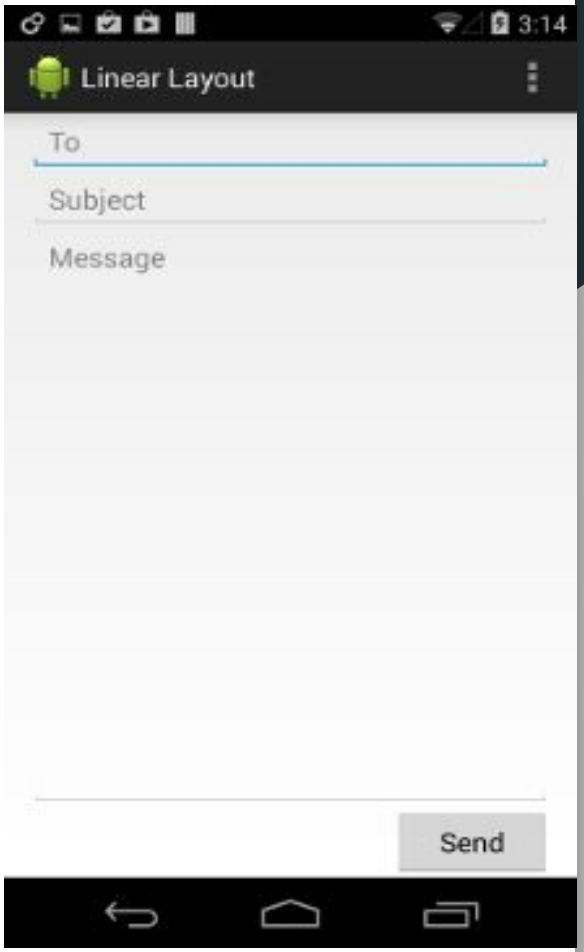
Type de Layout	Description	Cas d’utilisation
Linear Layout	Dispose les éléments de l'interface utilisateur en une seule rangée ou colonne.	Formulaires, listes, conceptions d'interface utilisateur simples.
Relative Layout	Dispose les éléments de l'interface utilisateur les uns par rapport aux autres ou par rapport au conteneur parent.	Conceptions d'interface utilisateur complexes nécessitant un positionnement précis des éléments.
Constraint Layout	Semblable à un RelativeLayout mais permet un positionnement plus précis des éléments de l'interface utilisateur.	Conceptions d'interface utilisateur complexes avec des éléments qui doivent être contraints les uns aux autres.

Les types de Layouts

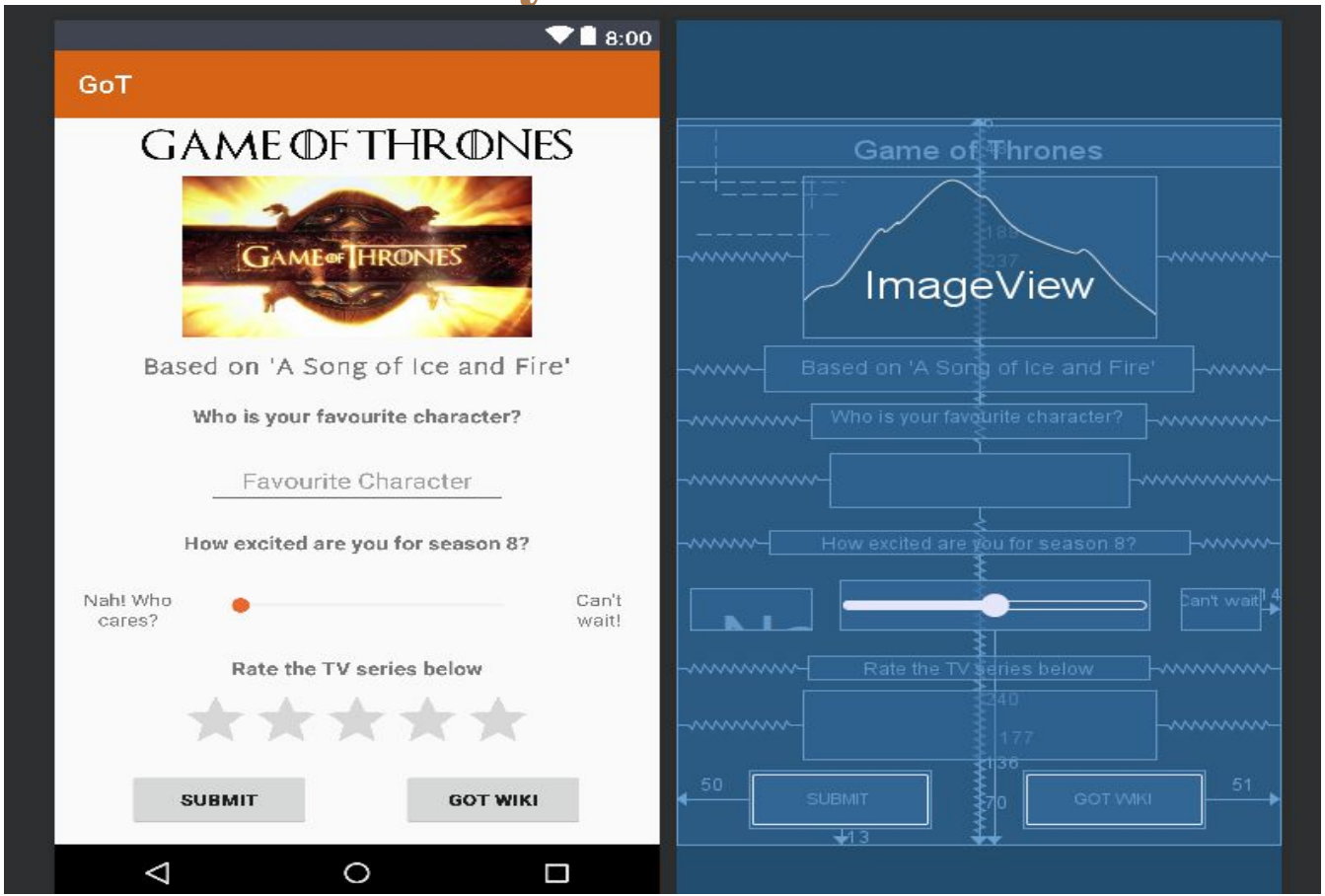
Table Layout	Organise les éléments de l'interface utilisateur en lignes et en colonnes, de manière similaire à un tableau HTML	Affichage des données tabulaires
Grid Layout	Organise les éléments de l'interface utilisateur selon une structure en grille, avec un nombre fixe de lignes et de colonnes.	Affichage de plusieurs éléments dans un modèle de type grille
Frame Layout	Permet d'afficher une seule vue enfant à la fois.	Affichage d'une vue unique qui occupe tout l'écran

Exemple: Linear Layout

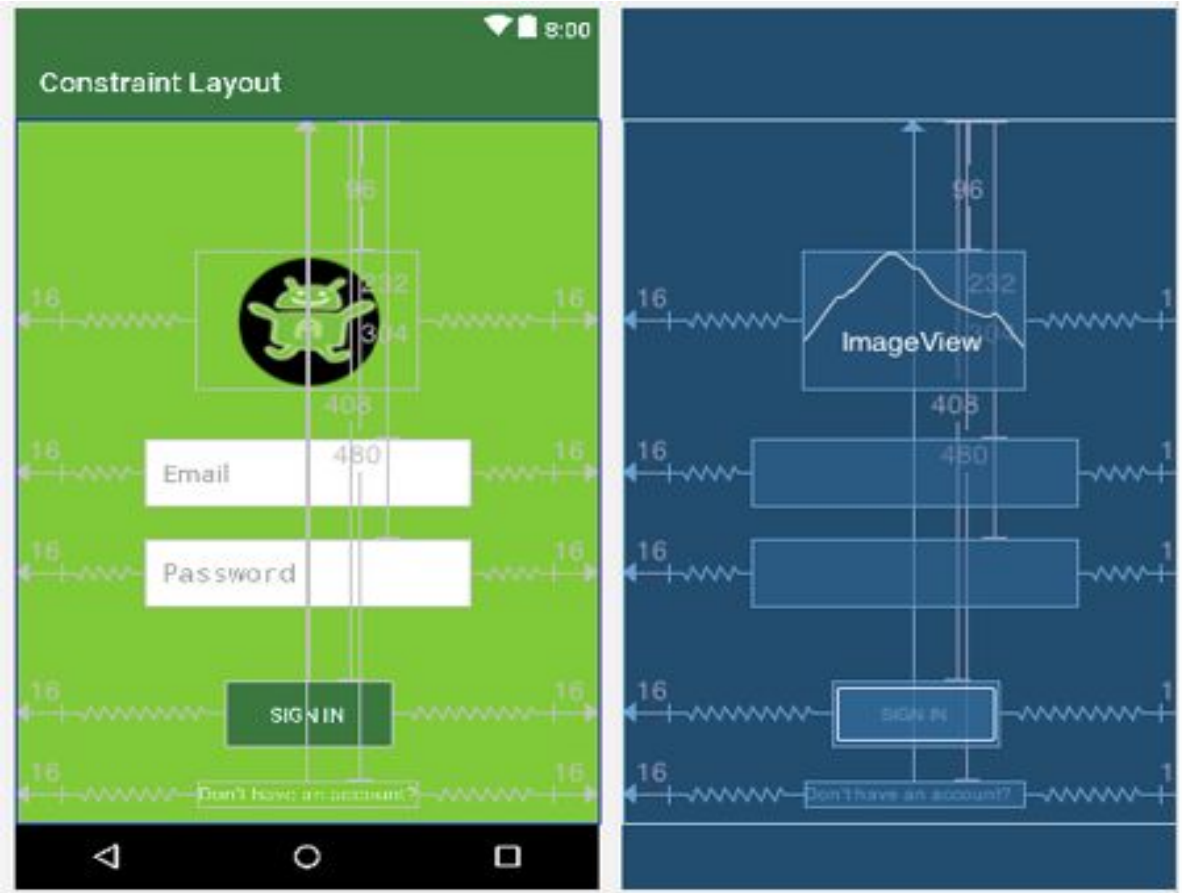
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```



Exemple: Relative Layout



Exemple: Constraint Layout

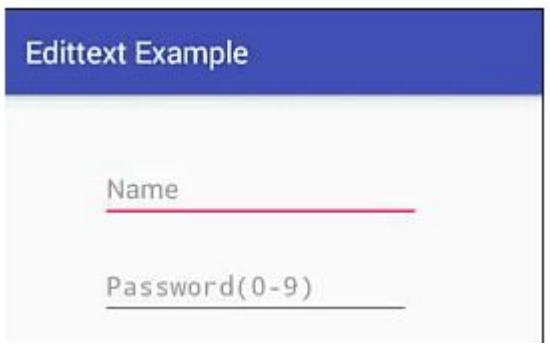


Views (Vues)

- Dans le développement d'applications Android, les "Views" désignent les éléments visuels ou composants qui constituent l'interface utilisateur (UI) d'une application.
- Les Views peuvent être des éléments simples tels que du texte, des images et des boutons, ou des éléments plus complexes comme des listes, des grilles et des vues web.
- Les Views sont implémentées en utilisant la classe View et ses sous-classes telles que TextView, EditText, Button, ImageView, ListView, et d'autres.
- Chaque View possède ses propres propriétés et méthodes qui définissent son comportement et son apparence, et elles peuvent être personnalisées pour répondre à des exigences spécifiques.

EditText

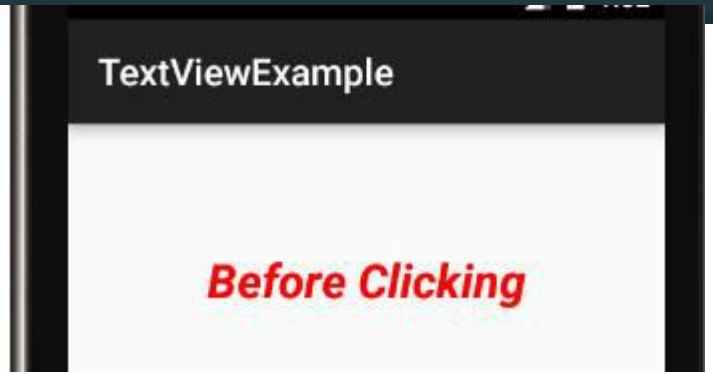
```
<EditText android:id="@+id/editText1“
    android:layout_width="wrap_content“
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="name"
    android:inputType="textPersonName" />
<EditText android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="password_0_9"
    android:inputType="numberPassword" />
```



- L'attribut "**android:hint**" permet d’afficher un commentaire
- L'attribut "**android:inputType**" permet de choisir un type d’entrée (text, mot de passe, nombre, date, ….)
- La méthode **getText()** permet de récupérer le texte saisi
- La méthode **setText()** permet de placer un texte

TextView

```
<TextView android:id="@+id/TextView1“
    android:layout_width="wrap_content“
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:text="Before Clicking"
    android:textColor="#f00"
    android:textSize="25sp"
    android:textStyle="bold|italic"
    android:layout_marginTop="50dp" />
```



- L'attribut " **android:textStyle** " permet de choisir le style du texte, peut prendre les valeurs “**normal**" , "**bold**" , "**italic**"
- L'attribut "**android:typeface** " permet de choisir la police de caractères, peut prendre les valeurs "**sans**", "**serif**", "**monospace**"
- L'attribut " **android:textSize** " permet de choisir la **taille** du texte
- L'attribut " **android:textColor** " permet de choisir la **couleur** du texte

Button/ImageButton

```
<Button android:id="@+id/But1 "
        android:layout_width="wrap_content "
        android:layout_height="wrap_content"
        android:gravity= " center"
        android:text= " Android" />
<ImageButton android:id="@+id/ImgBut1 "
        android:layout_width="wrap_content "
        android:layout_height="wrap_content"
        android:gravity= " center"
        android:srcCompat= " @mipmap/ic_launcher" />
```

CheckBox

```
<CheckBox android:id="@+id/ChBox1"
      android:layout_width="“wrap_content "
      android:layout_height="wrap_content"
      android:text= " Android"
      android:textColor="#44f“
      android:textStyle="bold|italic" />
<CheckBox android:id="@+id/ChBox2"
      android:layout_width="“wrap_content "
      android:layout_height="wrap_content"
      android:text= " Java"
      android:textColor="#f44“
      android:textStyle="bold|italic" />
```

- Pour vérifier si un CheckBox est coché:

```
CheckBox ch= (CheckBox) this.findViewById (R.id.ChBox1);
boolean b=ch.isChecked();
```

RadioButton

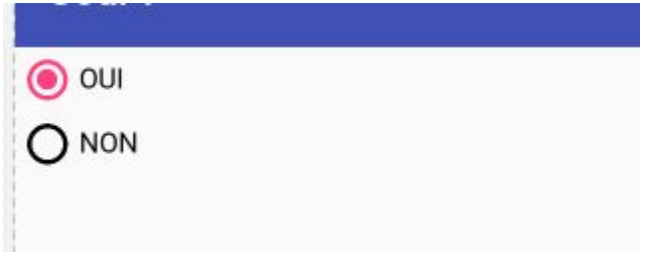
```
<RadioButton android:id="@+id/RBut1 "
        android:layout_width=“match_parent "
        android:layout_height="wrap_content"
        android:text= " I am a radiobutton" />
```

- Pour vérifier si un RadioButtonest coché:

```
RadioButton ch= (RadioButton) this.findViewById (R.id.RBut1);
boolean b=ch.isChecked();
```

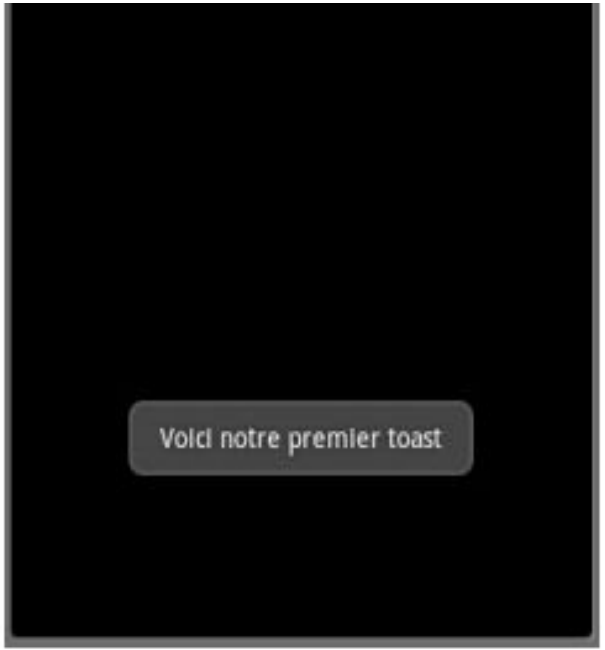
RadioGroup

```
<RadioGroup
        android:layout_width=“wrap_content "
        android:layout_height="wrap_content"
        android:orientation= " vertical" >
RadioButton
        android:layout_width=“wrap_content"
        android:layout_height="wrap_content"
        android:text= "OUI"
        android:checked= “true" />
RadioButton
        android:layout_width=“wrap_content"
        android:layout_height="wrap_content"
        android:text= “NON" />
</RadioGroup>
```



Toast

- Android Toast peut être utilisé pour afficher des informations à l'utilisateur pour une courte période de temps.
- Un toast contient un message à afficher rapidement et disparaît après un certain temps sans intervention de l'utilisateur.
- Vous pouvez également créer un toast personnalisé, par exemple un toast affichant une image.
- Un "toast" est une instance de la classe "Toast" (android.widget.Toast) Toast




```
Toast toast= Toast.makeText(this,"Voici notre premier toast",
Toast.LENGTH_LONG);
toast.show();
```

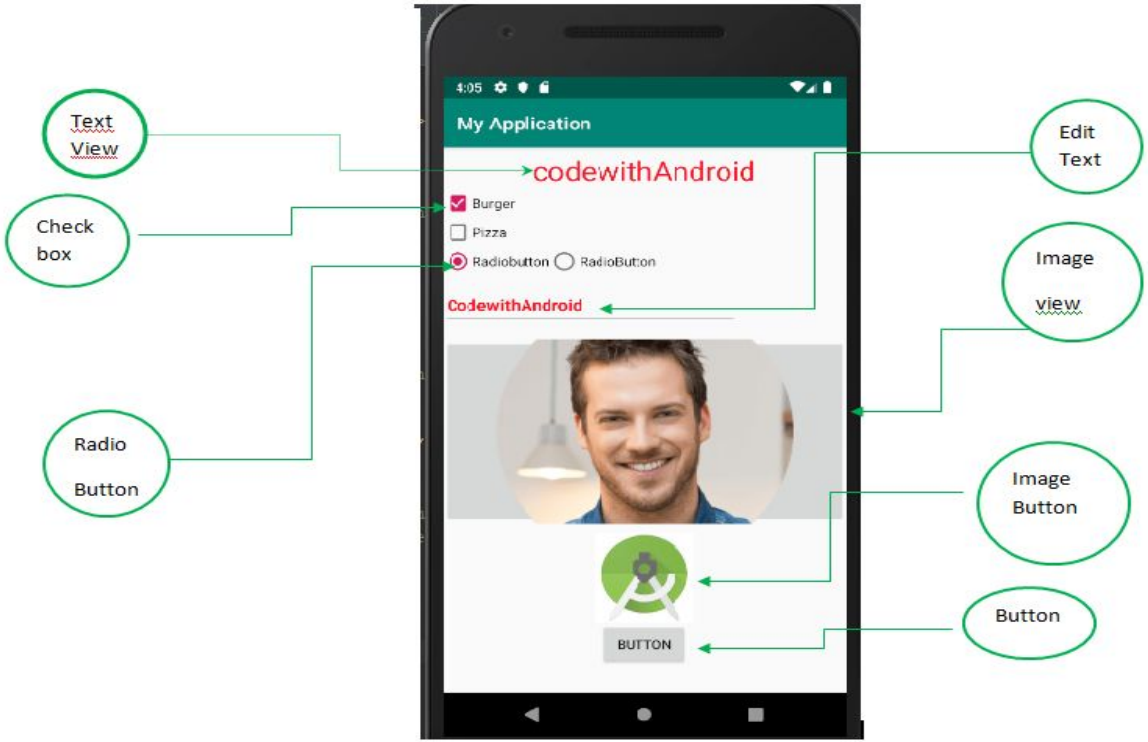
Types de Vues

Vue	Description
Button	Un bouton cliquable qui exécute une action lorsqu'il est cliqué
TextView	Une vue de texte non modifiable qui affiche du texte à l'utilisateur
EditText	Une vue qui permet à l'utilisateur de saisir du texte
ImageView	Une vue qui affiche une ressource image
CheckBox	Une vue qui permet à l'utilisateur de sélectionner une ou plusieurs options parmi un ensemble
RadioButton	Une vue qui permet à l'utilisateur de sélectionner une option parmi un ensemble
Spinner	Une vue qui affiche une liste déroulante d'options que l'utilisateur peut sélectionner
SeekBar	Une vue qui permet à l'utilisateur de sélectionner une valeur dans une plage de valeurs en faisant glisser un curseur le long d'une piste horizontale

Types de Vues

SeekBar	Une vue qui permet à l'utilisateur de sélectionner une valeur dans une plage de valeurs en faisant glisser un curseur le long d'une piste horizontale
RatingBar	Une vue qui permet à l'utilisateur d'évaluer quelque chose en sélectionnant un nombre d'étoiles
ProgressBar	Une vue qui affiche un indicateur de progression, généralement utilisé pour indiquer l'avancement d'une tâche en arrière-plan
Switch	Une vue qui permet à l'utilisateur de basculer un paramètre binaire
ToggleButton	Une vue qui permet à l'utilisateur de basculer un paramètre binaire, semblable à un interrupteur mais avec une apparence différente
WebView	Une vue qui affiche du contenu web dans l'application.

Exemple de Vues



Manipulation de l'interface utilisateur (UI)

Manipuler l'interface dans le code Java implique d'accéder aux différents éléments de l'UI et de modifier leurs propriétés, telles que le texte, la visibilité et le comportement. Cela peut être réalisé en obtenant des références aux éléments de l'UI via leurs ID attribués et en utilisant les méthodes fournies par le SDK Android. Par exemple, pour modifier le texte d'un élément TextView, nous pouvons utiliser la méthode setText(), comme illustré dans l'extrait de code suivant :

Manipulation de l'interface utilisateur (UI)

Manipuler l'inte ents éléments de l'UI et de modifi e comportement.

Cela peut être UI via leurs ID attribués et en u r exemple, pour modifier le text thode setText(), comme illustré dans l'extrait de code suivant :

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="15dp"
    android:text="TextView"
    tools:layout_editor_absoluteX="115dp"
    tools:layout_editor_absoluteY="334dp" />
```

```
TextView textView = findViewById(R.id.textViewId);
textView.setText("New Text");
```

Manipulation de l'interface utilisateur (UI)

De même, pour définir la visibilité d'un élément View comme caché, nous pouvons utiliser la méthode setVisibility() avec la constante View.INVISIBLE, comme ceci :

```
View view = findViewById(R.id.viewId);
view.setVisibility(View.INVISIBLE);
```

Manipulation de l'interface utilisateur (UI)

Pour gérer les entrées utilisateur, nous pouvons attacher des écouteurs d'événements aux éléments de l'interface utilisateur en utilisant la méthode setOnClickListener(), comme ceci :

```
Button button = findViewById(R.id.buttonId);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Handle button click event
    }
});
```

Écouteurs d'événements dans Android

Écouteurs d'événements	Description
OnClickListener	Réagit aux clics sur une vue
OnLongClickListener	Réagit aux clics longs sur une vue.
OnTouchListener	Réagit aux événements tactiles sur une vue.
OnKeyListener	Réagit aux pressions de touches.
OnFocus ChangeListener	Réagit aux changements de focus.
OnCheckedChangeListener	Réagit aux changements d'état coché d'un bouton composite.
OnSeekBarChangeListener	Réagit aux changements de progression d'une barre de recherche.
OnItemSelectedListener	Réagit à la sélection d'un élément dans un spinner.
TextWatcher	Réagit aux changements dans le texte d'une vue modifiable.
OnEditorActionListener	Réagit aux actions de l'éditeur, telles que "terminé" ou "suivant".

Écouteurs d'événements dans Android

GestureDetector.OnGestureListener	Réagit aux gestes, tels que les lancers, les balayages et les tapotements.
GestureDetector.OnDoubleTapListener	Réagit aux gestes de double tapotement.
ScaleGestureDetector.OnScaleGestureListener	Réagit aux gestes de mise à l'échelle.
View.OnDragListener	Réagit aux événements de glissement et de dépôt.
View.OnLayoutChangeListener	Réagit aux changements dans la mise en page d'une vue.
ViewTreeObserver.OnGlobalLayoutListener	Réagit aux changements de disposition globale, comme les changements de taille ou de position d'une vue.
ViewTreeObserver.OnScrollChangedListener	Réagit aux changements de position de défilement d'une vue.
ViewTreeObserver.OnPreDrawListener	Réagit avant qu'une vue ne soit dessinée, permettant des modifications à l'arbre de vue.
ViewTreeObserver.OnWindowFocusChangeListener	Réagit aux changements d'état de focus d'une fenêtre.
View.OnSystemUiVisibilityChangeListener	Réagit aux changements de visibilité de l'UI système, comme lorsque la barre d'état ou la barre de navigation est affichée ou cachée.