TP3 : Interface graphique (Les composants standards d'une activité)

Objectifs :

- Se familiariser avec les composants graphiques
- Réaliser une interface graphique en XML ou en java
- Utiliser Toast, setContentView, findViewbyId, onClick

I. View et ViewGroup

I.1 Le Composant View

La classe **View** représente la classe de base pour la création d'une interface graphique en Android. C'est la classe mère de tous les widgets (éléments graphiques), utilisés pour créer des composants graphiques interactifs (boutons, champs texte, champs de saisie…).
Une vue occupe un espace rectangulaire sur l'écran, responsable de la gestion des évènements initiés par l'utilisateur.
Une sous-classe **ViewGroup** est définie par Android, représentant la classe de base pour tous les layouts (dispositions), qui sont des conteneurs invisibles qui rassemblent plusieurs **View** ou **ViewGroup**, et définissent leur emplacement dans l'écran.

I.2 Les Layouts

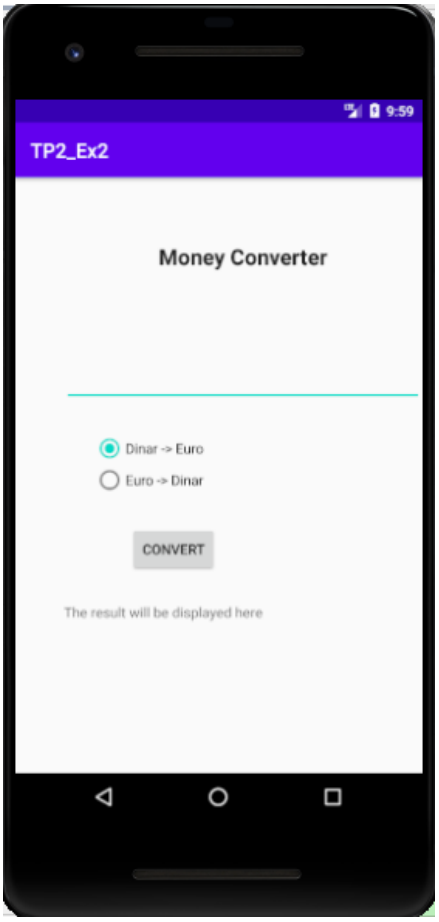
Chaque élément d'une interface graphique est soit un objet **View**, soit **ViewGroup**. Les dispositions de ces éléments dans l'écran sont précisées principalement par des Layouts.
Un Layout est un **ViewGroup** qui regroupe un ensemble de widgets ou d'autres layouts, permettant ainsi de définir leur disposition sur l'écran de la fenêtre principale. Parmi les layouts existants, on cite le **FrameLayout**, **LinearLayout**, **RelativeLayout, ConstraintLayout**…

II. Exercices

Exercice 1 : Composants Graphiques de Base

1. Objectif

L'objectif de cette première partie est de réaliser une application simple de conversion de monnaie, ayant une interface semblable à ce qui suit (veiller à respecter les tailles et emplacements des éléments graphiques) :



2. Comportement d'un EditText

Un **EditText** est un objet graphique qui permet à l'utilisateur de saisir une chaîne de caractères, utilisable par l'application.

De même que pour le bouton (ainsi que tous les éléments graphiques que nous désirons utiliser dans notre application), nous devons définir et initialiser un objet Java associé au champs **EditText** :

1. Créer un attribut dans votre activité de type **EditText**:

```
private EditText eEntry;
```

2. Dans la méthode **onCreate()**, initialiser l'attribut **eEntree** en lui associant le champs de saisie créé dans le main.xml :

```
this.eEntry = (EditText) this.findViewById(R.id.e_entry) ;
```

3. Définir son comportement. Pour un champs de saisie, les principales fonctionnalités sont :
   - La lecture du contenu : pour cela on utilise la méthode :

```
String s = eEntry.getText().toString();
```

   - La modification du contenu :

```
eEntry.setText("Nouveau texte");
```

3. Comportement d'un TextView

Un **TextView** est un objet graphique qui permet d'afficher une chaîne de caractères non-éditable par l'utilisateur.
Le **TextView** peut être utilisé par l'application exactement de la même manière qu'un **EditText**.

4. Comportement d'un Bouton Radio

Un bouton **radio** est un bouton à deux états qui peut être soit coché (**checked**) ou décoché (**unchecked**). Les boutons radios sont en général utilisés dans un groupe **RadioGroup**. Au sein d'un même groupe, un seul bouton radio peut être coché.

Pour gérer l'état d'un bouton radio, il faut suivre les étapes suivantes :

1. Créer un attribut de type **RadioButton** dans votre activité (par exemple rDinarEuro).
2. L'associer au bouton radio approprié de votre interface en utilisant la méthode **findViewById**.
3. Pour tester l'état de votre bouton radio, appeler la méthode **isChecked()**. Par exemple :

```
if (rDinarEuro.isChecked() ){

        //traitement
}
```

4. Pour mettre à jour l'état du bouton radio, utiliser la méthode **setChecked**(boolean etat).

Par exemple, si on veut cocher l'élément radio1 et décocher radio2, on peut faire comme suit :

```
radio1.setChecked(true);
radio2.setChecked(false);
```

Remarque :

1. Pour utiliser les boutons radios, il est inutile de définir des variables en Java pour le **RadioGroup**.
2. Dans l'exemple précédent (4), si radio1 et radio2 se trouvent dans un **RadioGroup**, il est inutile de changer leurs deux états : le changement de l'état de l'un va automatiquement changer l'autre, puisqu'un seul peut être coché à la fois.

5. Comportement du bouton

Soit le bouton défini dans le fichier main.xml, dont l'identifiant est b_convert. Pour manipuler ce bouton, trois méthodes principales sont en général utilisées:

5.1 Méthode 1 : Surcharge du Listener du bouton

En Java, un **Listener** (écouteur) est un objet permettant au programmeur de réagir à la suite des actions de l'utilisateur (clic de souris, touche du clavier, etc.). Dans notre cas, nous avons besoin d'un écouteur pour le clic sur le bouton, appelé **onClickListener**. **Listener** est une interface, qui fournit des méthodes qui doivent être implémentées par le programmeur.

Par exemple, le **onClickListener** contient une méthode **onClick**, qu'on doit implémenter pour définir le comportement de notre bouton.

1. Créer un attribut dans votre activité de type **Button** :

```
private Button bConvert;
```

2. Dans la méthode **onCreate()**, initialiser l'attribut **bAfficher** en lui associant le bouton créé dans le main.xml :

```
this.bConvert = (Button) this.findViewById(R.id.b_convert) ;
```

3. Utiliser le code suivant pour définir le comportement du bouton bConvert (de préférence à l'intérieur de la méthode onCreate).

```
this.bConvert.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
//comportement de votre bouton
}
});
```

5.2 Méthode 2 : Définition d'une méthode propre au bouton

Il existe une manière moins encombrante de définir le comportement du bouton, mais qui ne peut pas s'appliquer à tous les évènements, seulement au clic.

• Dans l'élément XML du bouton, ajouter l'attribut :

```
android:onClick="convert"
```

Cet attribut indique qu'il existe une méthode dans le code de l'activité, appelée **convert**, qui définit le comportement du bouton au clic.

• Dans le code de l'activité, ajouter la méthode suivante :

```
public void convert(View v){
        // comportement du bouton
}
```

La méthode **convert** ainsi définie a une signature particulière : elle doit obligatoirement être publique, retourner **void** et prendre comme paramètre un objet de type **android.view.View** (vous remarquerez que cette méthode a la même signature que la méthode **onClick**, surchargée dans la première méthode). Il faut noter également que la vue v passée en paramètre, correspond à l'objet cliqué.

Remarque : Si vous utilisez cette solution, il est inutile de définir une variable de type **Button** en Java, et de l'associer au bouton défini dans le fichier XML.

5.3 Méthode 3 : Implémentation de l'interface OnClickListener

Il est possible d'utiliser l'héritage pour surcharger la méthode **onClick**, sans passer par l'appel à la méthode **setOnClickListener**. Il suffit de suivre les étapes suivantes :

1. Votre **activity** doit implémenter l'interface **OnClickListener**. Ceci est réalisé en transformant la signature de votre classe activité comme suit, par exemple :

```
public class MoneyConverter extends Activity implements OnClickListener {…}
```

2. Créer l'attribut **bConvert** de type **Button** et l'associer à l'élément XML **b_convert**, comme dans la méthode 1.

3. Définir l'activité courante comme étant l'écouteur du clic sur le bouton **bConvert** :

```
bConvert.setOnClickListener(this);
```

4. Ajouter la méthode **onClick** dans votre activité, comme suit :

```
public void onClick(View v) {
        if (v.getId()==R.id.b_convert){
```

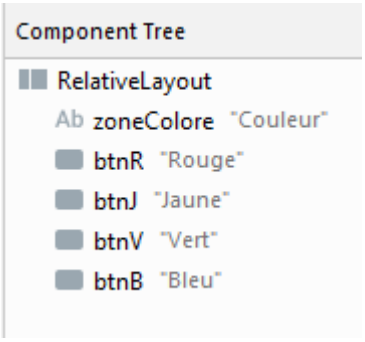
```
                //Comportement du bouton b_convert
                }
        }
```

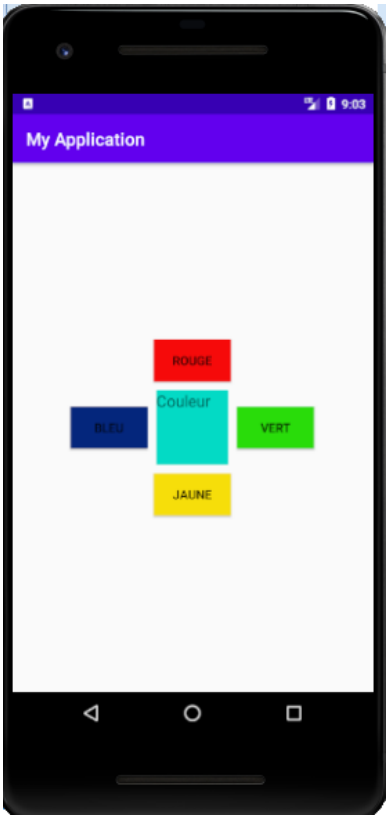
Attention, cette méthode sera commune à tous les éléments cliquables, il faut donc distinguer le comportement selon l'identifiant de l'élément cliqué.
Cette méthode peut s'avérer utile dans le cas où on voudrait par exemple regrouper les implémentations de tous les boutons de l'interface dans la même méthode, ou si plusieurs boutons partageaient une partie de leur comportement.

Question. Définir maintenant le comportement des différents composants de votre interface en utilisant la méthode de votre choix.
Ajouter un autre bouton « Annuler » qui permet de remettre les champs à l'état initial.

Exercice 2 : RelativeLayout

L'objectif est de réaliser l'interface graphique suivante avec RelativeLayout :





La zone de milieu est une zone de texte (TextView) située au milieu de l'écran, entourée par 4 boutons (Button) respectivement bouton rouge, vert, jaune, bleu.

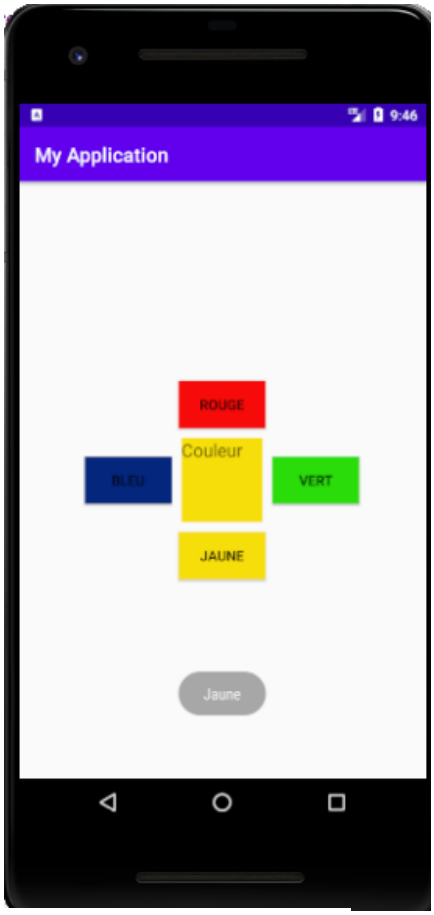
1. Compléter le fichier XML nommé activity_couleur.xml

- Ajouter des couleurs en utilisant colors.xml
- Ajouter les contraintes de positionnement de la zone de texte au milieu, ensuite respectivement les boutons Rouge, Jaune, Vert et Bleu.

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


    <TextView
        android:id="@+id/zoneColore"
        android:layout_width="82dp"
        android:layout_height="85dp"
        android:layout_margin="10dp"
        android:text="Couleur"
        android:background="@color/colorAccent"
        android:textSize="18sp"
          . . . />

    <Button
        android:id="@+id/btnR"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@color/Rouge"
        android:text="Rouge"
        android:onClick="rouge"
          . . . />

    <Button
        android:id="@+id/btnJ"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Jaune"
        android:background="@color/Jaune"
        android:onClick="jaune"
          . . . />

    <Button
        android:id="@+id/btnV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@color/Vert"
        android:text="Vert"
        android:onClick="vert"
          . . . />

    <Button
        android:id="@+id/btnB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@color/Bleu"
        android:text="Bleu"
        android:onClick="bleu"
```

```
            . . . />
    </RelativeLayout>
```

2. Compléter le code suivant dans le fichier couleurActivity.java. Au clic sur chaque bouton la zone au milieu sera colorée avec la couleur mentionnée sur le bouton, et un message sera affiché avec la couleur choisie.




```
public class CouleurActivity extends
AppCompatActivity {

    EditText zone;

    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(. . .);
        //Initialisation
        zone= . . .;

    }

    public void rouge(View v)
    {
        zone. . .

     }
. . .
}
```

3. Refaire le même projet en utilisant ConstraintLayout.