Institut Supérieur d'Informatique

Université de Tunis el Manar

TP3 : Composants Android

Programmation Mobile – 2ème Licence – Systèmes Embarqués

Année Universitaire : 2011/2012

MME. LILIA SFAXI

TP3 : Composants Android

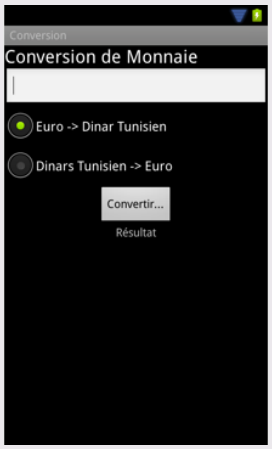
Programmation Mobile

Objectifs du TP

Ce TP a pour objectif de vous initier aux différents composants importants d'Android.

Nous allons dans ce TP réaliser progressivement une petite application de conversion.

TAF-1 **:** Créer un nouveau projet intitulé *Conversion*, qui contient le package *isi.conversion*. Créer les éléments nécessaires pour que l'interface soit comme la figure suivante :



I. Boutons, boutons radios et cases à cocher

Dans le TP précédent, nous avons vu comment gérer le comportement des boutons. Ce n'est pas la seule manière.

I. 1. Associer un comportement à un bouton

1. Créer une méthode dans le code Java de l'activité qui définit le comportement du bouton. On l'appellera

```
public void auClicMethode(View v){…}
```

Cette méthode doit obligatoirement être publique, retourner *void* et avoir un paramètre de type *android.view.View*.

2. Créer un bouton dans le fichier *layout* en utilisant la partie graphique.

3. Modifier son identifiant et son texte à votre guise

> **Remarque** : Il vous est possible de modifier ces informations graphiquement. Pour cela, ouvrir la partie graphique du fichier *layout*, clic-droit sur l'élément à configurer, et choisir *Edit Text*, *Edit ID*…

4. Dans le code xml de votre bouton, ajouter l'attribut :

```
android:onClick = "@string/auClic"
```

5. Créer dans le fichier *strings.xml* un nouveau string dont le nom est *auClic* et la valeur est *auClicMethode* (qui est le nom de la méthode que vous avez créé dans 1.)

> ***TAF-2*** **:** Créer une méthode appelée *convertir* et l'associer au bouton *Convertir* de votre interface.

I. 2. Gérer les boutons radios

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 *radio1*).
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 (radio1.isChecked() ){
        //traitement
}
```

TAF-3 **:**

1. Créer deux méthodes : dinarToEuro et euroToDinar, qui prennent de convertir une valeur en entrée :

```
private float dinarsToEuro(float valeurDinar) {
        return (float) (valeurDinar * 1.9919);
}

private float euroToDinar(float valeurEuro) {
        return (float) (valeurEuro * 0.5020);
}
```

2. Implémenter la méthode *convertir* pour qu'elle fasse la conversion nécessaire, selon le bouton radio qui est coché. Mettre le résultat dans le champs de texte *Resultat*.

Indication : La valeur lue dans le champs de saisie (ici appelé ***edt****) doit être convertie en float pour être manipulée. Pour cela, utiliser le code suivant :*

```
EditText edt = (EditText) findViewById(R.id.edit_float);
float number = Float.valueOf(edt.getText().toString());
```

D'autre part, pour extraire la chaîne de caractères associée à une variable float (appelée ici ***floatVar****), utiliser le code suivant :*

```
String s = String.valueOf(floatVar) ;
```

I. 3. Gérer les cases à cocher

Tout comme les boutons radio, les cases à cocher ont deux états : coché ou décoché. Cependant, on peut avoir plusieurs cases qui sont cochées en même temps, et elle sont la plupart du temps indépendantes.

Pour gérer l'état d'une case à cocher, il faut suivre les étapes suivantes :

1. Créer un attribut de type *CheckBox* dans votre activité (par exemple *check1*).
2. L'associer à la case à cocher appropriée de votre interface en utilisant la méthode *findViewById*.
3. Pour tester l'état de votre case à cocher, appeler la méthode *isChecked()*. Par exemple :

```
if (check1.isChecked() ){
        //traitement
}
```

4. Pour modifier l'état de la case à cocher, utiliser la méthode *setChecked(boolean etat)*. Par exemple :

```
check1.setChecked(false) ;        //pour décocher la case
check1.setChecked(true) ;         //pour cocher la case
```

II. Menus

Sur Android, les menus permettent d'ajouter des fonctionnalités à une application en fournissant des opérations supplémentaires, initialement cachées à l'utilisateur.

Il existe deux types de menus :

- Un menu *d'options* : déclenché par le bouton matériel *Menu* sur le téléphone.
- Un menu *contextuel* : déclenché par un évènement sur un élément de l'interface, par exemple un *long clic*.

Ces deux types de menus peuvent contenir :

- Du texte
- Des icônes
- Des boutons radios
- Des cases à cocher
- Des sous-menus
- Des raccourcis…

II. 1. Menu contextuel

Pour définir un menu contextuel, qui sera déclenché suite à un long clic sur un élément (que j'appelle ici *element :* il peut être un bouton, un textView, ou même le layout en entier) :

1. Créer un attribut pour l'élément auquel on veut ajouter un menu contextuel, et l'associer à l'élément graphique avec la méthode *findViewById(…).*
2. Ajouter un écouteur pour le long clic, de la même manière que nous avons ajouté un écouteur pour le clic dans le TP précédent. Cet écouteur(*Listener),* va ordonner l'affichage du menu contextuel quand on fait un long clic sur *element* :

```
element.setOnLongClickListener(new OnLongClickListener() {

	@Override
	public boolean onLongClick(View v) {
		v.showContextMenu();
		return false;
	}});
element.setOnCreateContextMenuListener (this) ;
```

3. Indiquer le comportement de ce menu. Pour cela, générer la méthode *onCreateContextMenu* dans votre activité. Elle est appelée quand un menu contextuel va être affiché.
4. Dans le corps de cette méthode, on peut ajouter des menus. Pour cela, utiliser la méthode :

```
menu.add(groupID, itemID, ordre, "nom du menu") ;
```

Cette méthode permet de créer un nouveau menu. Les paramètres nécessaires sont les suivants :

- *groupID* : identifiant du groupe. Il est possible de regrouper les éléments, mais dans notre cas, on n'en a pas besoin, on lui donne donc la valeur 0 ;
- *itemID* : identifiant de ce menu. Il nous sera utilise pour identifier ce menu parmi les autres. On doit donner un identifiant différent à chaque menu (1, 2, 3… par exemple).
- *ordre* : associer un ordre d'affichage au menu. On donnera toujours la valeur 0.
- *Nom du menu* : chaîne qui représente le titre du menu.

5. Nous devons dire quel est le comportement à faire quand on clique sur cet élément du menu. Pour cela, générer la méthode :

```
public boolean onContextItemSelected(MenuItem item){…}
```

Dans le corps de cette méthode, indiquer le comportement à adopter, si on clique sur le menu 1 ou 2 :

```
switch(item.getItemId()){
	case 1:
		//traitement 1
	case 2:
		//traitement 2
}
```

TAF-4 **:** Créer un menu contextuel sur les deux boutons radios. Il doit contenir deux menus :

1. « *Taux dinar -> euro* » : affiche dans un Toast le taux de conversion du dinar vers l'euro.
2. « *Taux euro -> dinar* » : affiche dans un Toast le taux de conversion de l'euro vers le dinar.

II. 2. Menu d'options

Le menu d'options se gère presque de la même manière qu'un menu d'options, mais sans la gestion de l'élément graphique auquel est associé le menu contextuel.

1. Générer la méthode *onCreateOptionsMenu* dans votre activité. Cette méthode permet de déterminer le comportement de l'application quand le bouton *Menu* de votre téléphone est appuyé. Elle prend comme paramètre un objet de type *Menu* : c'est le menu que vous allez manipuler.
2. Dans le corps de la méthode *onCreateOptionsMenu* (avant l'instruction *return*), ajouter les différents menus dont vous avez besoin.
3. Générer la méthode *onOptionsItemSelected* pour définir le comportement au clic d'un menu.

TAF-5 : Créer un menu d'options qui contient deux menus :

1. *Conversion C <-> F* : qu'on laissera vide pour l'instant
2. *Quitter* : permet de quitter l'application.

Indication : Pour quitter l'application, il faut appeler la méthode ***finish()***.

III. Messages d'Alerte

Pour ajouter un message d'alerte, il suffit d'insérer le code suivant :

```
AlertDialog alertDialog;
alertDialog = new AlertDialog.Builder(this).create();
alertDialog.setTitle("titre de l'alerte");
alertDialog.setMessage("Message à afficher !!!");
alertDialog.setButton("OK", new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
      }});
alertDialog.show();
```

TAF-6 : Créer une alerte appelée si l'utilisateur clique sur le bouton *conversion* alors que le champs en entrée est vide.

IV. Homework

- Créer une nouvelle activité dans le même projet, qui s'appelle *ConversionTemperature*. Cette activité présente une interface similaire à l'activité précédente, et permet de convertir entre le Celcius et le Farenheit.

Indication :
Tc = (5/9)*(Tf-32)
Tf = (9/5)*Tc+32; avec Tc=température en Celcius et Tf=température en Farenheit

- Implémenter le menu d'option Conversion C <-> F de la première activité pour qu'il ouvre la deuxième
- Créer un menu d'options dans la deuxième activité qui permet de :
  - revenir à la conversion euro <-> dinar
  - quitter