Université Ibn Tofaïl

Faculté des sciences-Kénitra

TP2
Interface graphique et interaction avec l'utilisateur sous Android Studio

License DAWM
Pr. EL AZAMI-Année 2023

1. Découverte des Ressources, Layouts et Vues

On va partir d'un projet minimal, type «Empty Activity», nommez-le TP2.

1.1. Ressources

Ce sont les fichiers XML qui sont dans *projet*/src/main/res.

- res/layout : ce sont les définitions d'écrans des activités, par exemple activity_main.xml. Les balises XML et leurs attributs définissent les éléments affichés : boutons, zones de saisie, etc. Dans les classes Java, ce layout sera désigné par R.layout.activity_main.
- res/values : le fichier strings.xml contient les textes de votre application : labels, messages, etc. Ces textes sont identifiés par l'attribut name et référencés dans les layouts par @string/nom, et par R.string.nom dans les sources Java.
  Si vous voulez traduire ces textes en allemand, faites une copie du dossier values en values-de et traduisez tous les textes sans changer les attributs name.
- res/mipmap-*dpi et res/drawable-*dpi : vous y trouverez les icônes dans différentes tailles. Chaque image PNG devient une ressource identifiée par R.drawable.son_nom en Java et @drawable/son_nom dans un layout. C'est le système qui choisit automatiquement, selon la densité de votre écran, celle qu'il faut afficher.

Les fichiers XML peuvent être affichés soit sous forme d'un formulaire ou de manière graphique, soit sous forme XML brute. Voyez l'onglet juste en bas de l'éditeur, ex: Design ou Text. Le mode XML offre un contrôle total, mais il faut savoir ce qu'on fait.

Actuellement, dans layout/activity_main.xml, il y a un message en dur dans android:text. C'est signalé par un avertissement dans Android Studio, car normalement, il faut placer tous les messages

dans res/values/strings.xml, afin qu'ils puissent être facilement traduits dans d'autres langues. Alors, dans ce dernier fichier, créez un nouvel élément < string name="message">Salut tout le monde</string> puis référencez cette chaîne dans le layout avec android:text="@string/message". Constatez que le message a été pris en compte. Pour certains layouts, il faut identifier les vues. Ça consiste à rajouter un attribut android:id="@+id/nom" :

```
< TextView android:id="@+id/texte"
    ... />
```

Ensuite, on peut référencer cet identifiant avec une syntaxe subtilement différente, sans le + :

```
< Button android:layout_below="@id/texte"
    ... />
```

1.2. Composants à connaître

Android contient plusieurs dizaines de composants d'interface :

- des vues : TextView, Button, EditText...
- des groupes : LinearLayout, RelativeLayout, TableLayout...

Il faut savoir que les composants évoluent et peuvent être déclarés obsolètes, par exemple DigitalClock, Gallery...

2. Tutoriel sur les groupements de vues

On va s'intéresser aux vues qui permettent d'arranger d'autres vues. Elles dérivent de la classe ViewGroup. Pour simplifier, elles ont des vues enfants et décident de leur placement, taille et position, en fonction de propriétés présentes sur les enfants. Voici les quatre à connaître :

- LinearLayout pour former des lignes ou des colonnes,
- TableLayout pour disposer en tableau,
- RelativeLayout pour mises en page statiques plus complexes,
- ConstraintLayout pour des dispositions dynamiques complexes.

2.1. Mise en page avec des LinearLayout

C'est le ViewGroup le plus simple et le plus facile à employer.

copiez ce document XML et appelez-le res/layout/linearlayout.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    < Button android:text="OK"
        android:layout_width="wrap_content" android:layout_height="wrap_content"/>
```

```
    < Button android:text="Annuler"
          android:layout_width="wrap_content" android:layout_height="wrap_content"/>
    < Button android:text="Annuler tout"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"/>

< /LinearLayout >
```

Examinez la mise en page en comparant avec les propriétés des vues, puis faites les changements suivants :

- Mettez match_parent pour android:layout_width de l'un des boutons, puis annulez le changement.
- Mettez match_parent pour android:layout_height du deuxième bouton, regardez son effet, puis annulez.
- Mettez 200dp puis 300dp pour android:layout_height du deuxième bouton, regardez son effet, puis annulez.
- Changez android:orientation en horizontal pour le LinearLayout, laissez ce changement. Passez en mode paysage si les vues sont trop encombrantes.

On voit que les vues ont une largeur dépendant de leur contenu. On voudrait qu'elles aient la même largeur.

- Ajoutez android:layout_weight="1" pour chacune. C'est pas très visible, mais la largeur n'est pas exactement la même.
- Changez android:layout_width="0dp" pour les trois vues. En principe, c'est le meilleur résultat. La largeur nulle combinée avec un poids non nul fait un calcul correct.
- Changez android:layout_width="wrap_content" pour le LinearLayout puis annulez ce changement.

Maintenant, à vous :

- Copiez linearlayout.xml et appelez-le password.xml
- Supprimez les 2e et 3e boutons, ne laissez que le OK. Ajoutez un EditText devant le bouton OK. Remplacez la propriété android:text par android:hint et mettez la chaîne "mot de passe". Puis modifiez les largeurs pour obtenir le résultat demandé. C'est à dire que l'EditText prend presque toute la place en largeur, tandis que le bouton OK est le plus petit possible. Pour cela, il faut mettre un poids nul au bouton OK mais aussi lui donner une taille minimale.

Dans un cas général, on est amenés à combiner des LinearLayout horizontaux et verticaux. Voici un exemple.

- Copiez password.xml et appelez-le calendar.xml
- Faites en sorte d'ajouter un CalendarView au dessus des deux vues EditText et Button pour obtenir l'image ci-dessous (en mode paysage pour ce sujet, le calendrier est tronqué, mais en mode portrait il est complet). Pour y arriver, vous devrez englober le LinearLayout horizontal

du bas dans un LinearLayout vertical, et en jouant sur les poids et hauteurs pour garantir une visibilité et une taille à tout le monde :

Vous pourrez recevoir un avertissement sur le coût des calculs en cas d'imbrication des layouts portant des poids. La présence de poids oblige l'algorithme du layout à faire plusieurs parcours des vues pour les placer.

2.2. Mise en page avec un TableLayout

Ce type de vue sert à organiser les éléments comme dans un tableau HTML. Ce sont les mêmes notions codées en XML. Voici un exemple à copier sous le nom res/layout/tableau.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent" android:layout_height="match_parent">

      < TableRow >
            <Button android:text="1" android:layout_width="wrap_content"
                  android:layout_height="wrap_content"/>
            <Button android:text="col2" android:layout_width="wrap_content"
                  android:layout_height="wrap_content"/>
      < /TableRow >

      < TableRow >
            <Button android:text="2" android:layout_width="wrap_content"
                  android:layout_height="wrap_content"/>
            <Button android:text="col2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/> < /TableRow > <
/TableLayout >
```

Rajoutez par copie-collage une colonne de plus, puis deux ou trois lignes. Regardez ce que ça fait dans l'aperçu. Par défaut, dans une colonne toutes les cases font la même largeur, et même si une colonne est plus étroite, elle reste quand même assez large. Il n'y a pas grande chose à configurer pour changer l'apparence à part deux propriétés :

- Ajoutez android:shrinkColumns="0" dans la balise <TableLayout> en haut. La première colonne 0 est autorisée à devenir plus étroite si la place manque, ou carrément disparaître. Pour voir cela, allongez le texte de l'un des boutons 2e colonne.
- Ajoutez android:stretchColumns="1" dans la balise <TableLayout>. La 2e colonne va s'élargir autant que possible – allongez le texte d'un bouton. Plusieurs colonnes peuvent être étirées : android:stretchColumns="1,2".

2.3. Mise en page avec un RelativeLayout

Ce ViewGroup permet une mise en page précise et complexe, mais par contre, il est difficile d'emploi. Le principe général est de positionner une vue enfant par rapport soit aux bords du RelativeLayout, soit à des vues précédemment posées. Par exemple : «coller le bord gauche de telle vue au bord droit de telle autre». Voici un exemple à copier sous le nom res/layout/relative.xml. Il est très artificiel, destiné à montrer les propriétés de placement.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent" android:layout_height="match_parent">

      < Button android:id="@+id/button1" android:text="BTN1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:layout_alignParentTop="true"/>
      < Button android:id="@+id/button2" android:text="BTN2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_toRightOf="@id/button1"/>
      < Button android:id="@+id/button3" android:text="BTN3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/button2"
            android:layout_alignLeft="@id/button1"
            android:layout_alignRight="@id/button1"/>

< /RelativeLayout >
```

Dans un RelativeLayout les vues enfant sont positionnées et dimensionnées par quatre contraintes, une par côté, haut, bas, gauche, droite. Les côtés droit et gauche sont aussi désignés par start et end pour pouvoir tenir compte des régionalisations (écriture de droite à gauche, etc.). Il y a des avertissements dans le *layout* précédent, il faudrait par exemple rajouter des propriétés comme toEndOf quand il y a toRightOf.

Il y a deux catégories de contraintes :

- Positionnement par rapport au layout.

  La valeur de ces contraintes sont de simples booléens et on ne les mentionne que quand ils sont vrais.
  - «collé au bord du parent» : layout_alignParentXXX avec *XXX* valant Top, Bottom, Left et Right, ainsi que les variantes Start et End.

- «centré dans le parent» : layout_centerInParent, layout_centerHorizontal et layout_centerVertical.

- Positionnement par rapport à une autre vue.

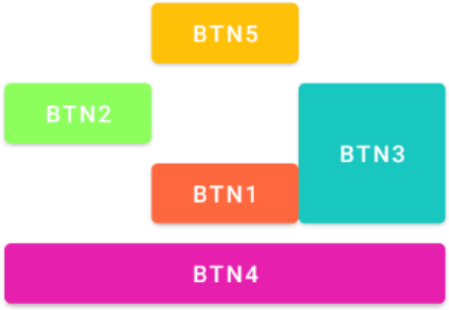
La valeur à mettre dans ces contraintes est la référence à l'identifiant d'une autre vue, @id/autre_vue. Chaque vue concernée doit donc définir un identifiant.

- «collé au bord opposé» : layout_toRightOf, layout_toLeftOf, layout_above et layout_below. Par exemple, il faut coller le bord droit de cette vue au bord gauche de l'autre vue.
- «bords alignés» : layout_alignXXX comme pour alignParent. Cette vue aligne son bord avec celui de l'autre vue.

Pour voir l'effet de ces contraintes, essayez ces changements et comprenez-les :

- dans le bouton1, changez la contrainte alignParentTop en alignParentBottom.
- dans le bouton1, allongez le titre android:text et constatez les répercussions sur le bouton3.
- dans le bouton3, allongez le titre, et constatez ce qui se passe.
- dans le bouton3, changez la valeur alignRight avec bouton2 au lieu de bouton1.

Maintenant, à vous, faites en sorte d'obtenir cette mise en page. Le bouton BTN1 est mis au centre de l'écran. Les boutons sont définis dans l'ordre de leur nom.



Pour changer la couleur d'un bouton, mettez d'abord android:backgroundTint="#888", puis cliquez sur le petit carré gris montrant cette couleur : un sélecteur de couleur apparaît. . .

2.4. Mise en page avec un ConstraintLayout

Ce layout fait partie de la bibliothèque support, celle qui permet de créer des applications fonctionnant sur de vieux smarphones. Cela impose une déclaration dans le fichier build.gradle de l'application, ainsi que l'utilisation de plusieurs *namespace* particuliers.

Ce type de ViewGroup est une extension de RelativeLayout, permettant de mettre à jour la disposition de manière plus dynamique et plus polyvalente. Les attributs pour mettre en place les contraintes ont des noms plus homogènes, mais nettement plus compliqués, obligeant à réfléchir davantage. La forme générale des contraintes est layout_constraintXXX_toYYYOf avec *XXX* et *yyy* valant Top, Bottom, Left, Right ou Baseline. Cela donne des attributs comme layout_constraintTop_toTopOf, layout_constraintRight_toLeftOf, etc. Le code Baseline fait aligner sur la base du texte des vues.

Tous ces attributs prennent pour valeur une référence d'identifiant (et non pas un booléen comme certains avec les RelativeLayout). Par exemple, on écrit layout_alignParentTop="true" dans un

RelativeLayout, mais layout_constraintTop_toTopOf="@id/id_du_layout" dans un ConstraintLayout – dans ce dernier cas, on peut aussi mettre une valeur spéciale pour indiquer le layout : "parent".

Pour positionner une vue à une certaine distance d'une autre, on utilise les attributs layout_marginXXX. Deux autres contraintes existent : layout_constraintZZZ_bias avec *ZZZ* étant Vertical ou Horizontal. Elles permettent de positionner une vue de manière proportionnellement à l'espace disponible – elles déséquilibrent une contrainte en faveur de l'un ou l'autre côté. La valeur à mettre est un réel entre 0 et 1, par défaut, c'est *0.5*.

La taille d'une vue peut être minimale wrap_content, maximale match_parent ou donnée par les contraintes de placement 0dp (aussi nommée *match_constraint*)

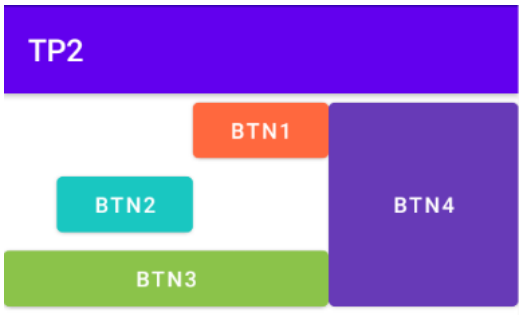
Voici maintenant un exemple à copier sous le nom res/layout/constraint.xml :

```
<?xml version="1.0" encoding="utf-8"?>
< android.support.constraint.ConstraintLayout
      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">
      < Button android:id="@+id/button1"
            android:text="BTN1"
            android:layout_height="wrap_content" app:layout_constraintTop_toTopOf="parent"
            android:layout_marginTop="32dp" android:layout_width="wrap_content"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintHorizontal_bias="0.25"/>
      < Button android:id="@+id/button2"
            android:text="BTN2"
            android:layout_height="wrap_content"
            app:layout_constraintTop_toBottomOf="@id/button1" android:layout_width="0dp"
            app:layout_constraintLeft_toLeftOf="@id/button1"
            app:layout_constraintRight_toRightOf="parent"/>

< /android.support.constraint.ConstraintLayout >
```

Le premier bouton est positionné à 32 pixels du haut et à 25% à gauche dans la largeur de son parent. Le second bouton est mis sous le premier, et comme sa largeur est nulle, elle est forcée à ce que donnent ses contraintes : du bord gauche du bouton 1 au bord droit du parent.

À vous maintenant. Essayez de reproduire cette disposition avec un ConstraintLayout :



Le bouton 1 est centré en haut. Le bouton 2 est placé en bas et à gauche du n°1. Le bouton 3 est sous le n°2 et va horizontalement du bord gauche de la fenêtre au bord droit du bouton 1. Le bouton 4 remplit l'espace à droite entre le haut de la fenêtre et le bas du bouton 3.

3. Exercices

Chacun de ces exercices consiste à créer un fichier res/layout/*exercice*.xml en mettant le titre de l'exercice à la place. Il vous suffit de changer l'appel à setContentView(R.layout.exercice) dans MainActivity.java pour passer d'un exercice à l'autre.

Par exemple, vous créez un nouveau fichier res/layout/seance.xml et vous mettez setContentView(R.layout.seance) dans la méthode onCreate de MainActivity. C'est ce layout qui sera affiché.

Utilisez le dessinateur (onglet *Design*) pour ajouter les vues, mais surveillez de temps en temps ce qui est généré en XML. Vous devrez apprendre à vous servir du dessinateur : gérer l'imbrication des vues et des containers, modifier les propriétés des vues, etc.

Dans les exercices, on demande de créer au minimum un écran qui permette de faire l'opération indiquée et au mieux, si vous avez le temps, cet écran sera ergonomique et élégant quelle que soit la taille et l'orientation du mobile: les vues bien alignées, avec un libellé en face, etc.

Également : au minimum, les textes seront en dur dans le layout, et au mieux ils seront sous forme de ressources dans strings.xml. AndroidStudio vous affiche une petite ampoule pour vous suggérer de transformer la chaîne «hard-coded» en ressource.

3.1. Saisie de repas : repas.xml

Créez une interface pour saisir un repas :

- une zone de saisie texte pour l'entrée,
- une pour le plat principal,
- deux cases à cocher pour indiquer si on a mangé du fromage et un fruit (on doit pouvoir cocher les deux).
- Il faut rajouter deux RadioButton midi et soir tels que si on coche soir, ça décoche midi et inversement. Pour cela, il faut les placer dans un RadioGroup.
- N'oubliez pas un bouton pour valider
- et éventuellement un autre pour annuler.

3.2. Saisie d'un plein de carburant : plein.xml

Créez une interface pour une application toute simple qui permet d'enregistrer des achats de carburant pour une voiture.

- Il faut au moins pouvoir saisir la quantité et le prix.
- Rajoutez aussi de quoi saisir la distance parcourue depuis le précédent plein.
- Il serait bien qu'il y ait la possibilité d'indiquer le carburant (E98, E95, E85, gasoil).
- Il faudrait également pouvoir saisir la date (EditText + Bouton qui afficherait un sélecteur de date, à ne pas faire car c'est de la programmation)
- et le lieu de l'achat.

Il faudrait voir du côté des attributs inputType des EditText pour restreindre la saisie des valeurs autorisées, nombres décimaux par exemple.

En ce qui concerne le sélecteur de carburant, si vous optez pour un Spinner, placez les différents carburants dans une ressource de type string-array, (voir le chapitre 2) et indiquez son identifiant dans l'attribut android:entries du Spinner. Voici un extrait :

```
<?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">
<Spinner
    android:id="@+id/carburant"
    android:entries="@array/carburants"...
```

avec un fichier res/values/arrays.xml à créer et contenant :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="carburants">
        <item>E98</item>
        <item>E95</item>
        ...
    </string-array>
</resources>
```

3.3. Saisie de relevés météo : meteo.xml

Créez une interface pour saisir un relevé météo :

- date,
- heure,
- température (°C),
- vitesse du vent (km/h)
- et état du ciel (parmi *dégagé*, *couvert* et *pluvieux*).

Pour la date et l'heure, il n'est pas possible d'utiliser les DatePicker et TimePicker qui sont beaucoup trop grands. Il faut simplement placer, pour chacun, un EditText collé à un bouton. Le bouton est destiné à faire apparaître une boîte de dialogue contenant un calendrier ou une horloge, mais cela, c'est pour un TP ultérieur.

Si vous avez le temps, configurez l'attribut inputType pour que les deux premières mesures n'acceptent que des nombres.

3.4. Saisie d'un livre :livre.xml

Créer un écran de saisie pour un livre : titre, auteur, année, isbn et notation. La notation, c'est à dire la note de 0 à 5 que vous attribuez au livre sera réalisée par un RatingBar. Cette vue est faite pour attribuer une note sous forme d'étoiles, mais elle est un peu déroutante à utiliser :

- Ne lui imposez pas une largeur autre que "wrap_content" sinon le nombre d'étoiles dépendra de la largeur effective,
- Pour fixer le nombre d'étoiles, affectez sa propriété android:numStars, mettez-en 5,

- Vous pouvez aussi fixer le pas, c'est à dire la granularité des notes, affectez par exemple android:stepSize="0.5"qui signifie qu'on peut mettre des notes de 0.5 en 0.5. Si vous avez le temps, configurez l'attribut inputType pour que l'année et le code isbn n'accepte que des chiffres.

4. Travail à rendre

Avec le navigateur de fichiers, descendez dans le dossier app/src de votre projet, probablement dans C:\AndroidStudioProject\TP2. Cliquez droit sur le dossier main, choisissez Compresser..., format ZIP, cliquez sur Creer. Déposez l'archive main.zip sur la classroom et sur moodle.