

Travaux pratiques Développement pour Android

Objectifs du TP

L'objectif de ces exercices est de réaliser une application illustrant les différents aspects du développement Android abordés dans le cours.

L'environnement de développement utilisé sera **Android Studio**. Le test de l'application se fera aux choix soit avec l'émulateur **AVD**, soit avec **Genymotion** soit sur un appareil réel exécutant Android.

L'application finale à réaliser s'appelle « **La Vache Fait Meuh** ». Son fonctionnement est illustré par la vidéo montrée en séance de cours et dont ci-dessous un lien. La vidéo est silencieuse, mais il faut noter que le clic sur la photo d'un animal doit produire le son que fait cet animal.

Lien vers la vidéo montrant l'application : <https://youtu.be/jdJNEX3Mihw>

La réalisation de l'application se fera d'une manière incrémentale sous la forme de 4 exercices. Le point d'entrée de chacun des exercices 2, 3 et 4 sera l'application réalisée dans l'exercice précédent.

Exercice 01**Réalisation d'une application avec un Splash Screen contenant**

L'objectif de cet exercice est de réaliser l'écran d'accueil de l'application « **La Vache Fait Meuh** ». Le point de départ est un nouveau projet Android avec une activité blanche. Cette activité sera modifiée pour contenir une image et deux zones de texte. Ensuite l'image sera animée.

Les concepts manipulés pendant cet exercices sont :

- Activités
- Layouts
- View et Widgets
- Ressources
- Animations
- Changement de l'icône de l'application

Question 01 : Créer un nouveau projet Android nommé « **La Vache Fait Meuh** » avec une activité blanche que l'on appellera **SplashActivity** ayant un layout nommé **activity_splash.xml**. Vous pouvez choisir le nom de domaine **enis.tn** pour votre projet.

NB : Faire attention lors du choix des N° d'API minimum. Il faut que cela concorde avec la version du système Android installé sur l'émulateur ou l'appareil sur lequel vous testerez votre appareil. Noter aussi qu'il est possible de changer ce paramètre après la création du projet en se rendant au fichier **build.gradle** et en modifiant le paramètre **minSDKVersion**.

Tester le bon fonctionnement de votre application pour s'assurer que l'environnement de développement est bien mis en place.

Question 02 : Si le layout généré dans le fichier **activity_splash.xml** est un **RelativeLayout**, il faut le modifier pour qu'il soit de type **LinearLayout** avec un arrangement **vertical** des widgets et une valeur « **match_parent** » pour la largeur et la hauteur.

LinearLayout permet d'arranger les widgets qu'il contient l'un à la suite de l'autre (verticalement ou horizontalement) tandis-que **RelativeLayout** arrange les widgets les uns relativement aux autres. Pour notre application, **LinearLayout** convient le mieux.

Question 03 : Par défaut, l'application contient une zone de texte **TextView** affichant un message « **Hello World** ». Le message est référencé par une ressource **@string/hello**. Il s'agit d'un identifiant d'une chaîne de caractère située dans le fichier **res/values/strings.xml**.

Modifier le message affiché pour qu'il affiche le nom de l'application (en utilisant un autre identifiant déjà existant dans **strings.xml**). Vous pouvez désormais supprimer l'identifiant **hello** du fichier **strings.xml** car il n'est plus nécessaire.

Question 04 : On souhaite maintenant ajouter une image au dessus du texte. Dans les ressources fournies pour cet exercice, copier le fichier **splash_icon.png** et le coller dans le dossier **res/drawable** du projet Android. Ceci est possible en cliquant avec le bouton droit sur le dossier **drawable** et en choisissant la commande « **paste** ». Cette ressource pourra désormais être accessible avec l'identifiant

@drawable/splash_icon dans les fichiers XML.

Ajouter un widget de type **ImageView** au dessus du **TextView** dans le layout **activity_splash** avec les propriétés suivantes :

```
<ImageView
    android:id="@+id/image_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/splash_icon"/>
```

Question 05 : Embellir l’affichage du *Splash Screen* en centrant les différents widgets au milieu de l’écran. Ceci se fait en :

- Ajoutant une marge en haut (**paddingTop**) de **100dp** au **LinearLayout**
- Déplaçant l’image au centre du layout et en modifiant sont **layout_width** à **match_parent**. Ne pas changer **layout_height** pour éviter la distorsion de l’image
- Changeant la taille du texte à **24dp**
- Alignant le texte au centre (attribut **gravity** et valeur **center_horizontal**)
- Éloignant le texte un peu de l’image en ajoutant une marge de haut de **40dp** au widget **TextView**.

Question 06 : On désire maintenant animer l’image. Il existe 3 types d’animations dans Android :

- **OpenGL** : Pour effectuer des animations avancées en 2D et 3D
- **Tween Animation** : en effectuant et combinant des transformation simple (rotation, translation, échelle, transparence) sur une image
- **Frame Animation** : en affichant une séquence d’images.

Pour notre application nous effectuerons une animation **Tween**. Afin de pouvoir manipuler l’image dans le code Java, il faut récupérer une référence vers la **TextView** correspondante. Pour ce faire, insérer à la fin de la méthode **onCreate** le code suivant :

```
ImageView splashIcon = (ImageView) findViewById(R.id.image_view);
```

On effectuera dans cette question une animation prédéfinie simple. Implanter les étapes suivantes et tester le fonctionnement de l’application :

- Créer un objet de type **Animation** en utilisant la méthode statique **loadAnimation** de la classe **AnimationUtils** en lui passant l’activité courante (**this**) et **android.R.anim.slide_in_left** comme paramètres. Il est possible de tester avec d’autres types d’animations que **slide_in_left** (**fade_in** par exemple...)
- Fixer la durée de l’animation créée à **3 secondes** (méthode **setDuration**).
- Invoquer la méthode **startAnimation** sur l’objet **ImageView** avec comme paramètre l’animation créée
- Exécuter l’application et vérifier le bon déroulement de l’animation.

Question 07 : On souhaite désormais implanter une animation plus complexe. Ceci se fait en décrivant l’animation sous la forme d’une ressource **XML** :

- Créer un nouveau dossier « **anim** » dans le dossier **res/** et y copier le fichier **custom_anim.xml** fourni
- Modifier le code de la question précédente pour charger la nouvelle animation au lieu de **android.R.anim.xxxx**. Ne pas oublier de supprimer l’invocation de

- **setDuration** car c'est déjà décrit dans le fichier **XML**
- Tester à nouveau le fonctionnement de l'application.

Question 08 : On désire maintenant perfectionner l'écran d'accueil. Effectuer les actions suivantes :

- Changer l'arrière plan du layout principal pour qu'il soit défini par l'image **arriere_plan.png** fournie (utiliser un attribut adéquat du layout)
- Changer la couleur du texte en blanc pour qu'il soit plus lisible avec le nouvel arrière plan. Les couleurs sont des ressources prédéfinies (**@android:color/...**)
- Ajouter une nouvelle zone de texte au-dessous de la première indiquant la version de l'application (nouvelle ressource à ajouter dans **strings.xml** et valant « **1.0** » pour cet exercice)
- Changer l'icône de l'application en écrasant les 4 versions du fichier **ic_launcher.png** situé dans le dossier **res/mipmap** par les 4 versions fournies pour cet exercice. Noter que chacun de ces fichiers doit s'appeler **ic_launcher.png**
- Tester le fonctionnement de l'application.

Exercice 02

Ajout d'une activité principale

L'objectif de cet exercice est de réaliser une activité principale pour l'application « **La Vache Fait Meuh** ». Le point de départ est l'application réalisée à la fin de l'**Exercice 01**. Dans un premier temps, cette activité sera affichée après l'écran d'accueil (*Splash Screen*). Ensuite, l'image d'un animal (une vache dans pour cet exercice) couvrira tout l'écran de la nouvelle activité. Enfin, le clic sur l'image déclenchera le son que fait la vache.

Les concepts manipulés pendant cet exercices sont :

- Ajout d'une nouvelle activité
- Lancement d'une activité à l'aide d'un **Intent**
- Utilisation d'un **MediaPlayer**

Question 01 : Planter les actions suivantes pour ajouter l'activité principale :

- Ajouter une nouvelle classe nommée **MainActivity** à votre projet et la faire hériter de la classe **android.app.Activity**. Il est possible aussi d'utiliser l'assistant d'*Android Studio* pour créer une nouvelle activité (cette solution vous épargnera la question 03)
- Ajouter une méthode **onCreate** pour cette activité dans laquelle on fixe le contenu au layout **R.layout.activity_main**. On peut insérer un squelette de code pour **onCreate** en cliquant avec le bouton droit dans la classe → *generate* → *Override Methods*, puis choisir la méthode voulue. Ceci fait gagner beaucoup de temps et diminue le risque d'erreur
- Dupliquer le fichier **XML activity_splash.xml** en un second fichier que l'on nommera **activity_main.xml**
- Modifier le contenu du fichier **activity_main.xml** pour qu'il contienne uniquement une **ImageView**
- Supprimer le paramètre **paddingTop** du fichier **activity_main.xml** pour que l'image occupe tout le layout depuis le haut
- Changer le paramètre **id** de l'**ImageView** pour qu'il puisse être référencé dans le code Java par **R.id.animal**
- Ajouter l'image **vache.jpg** fournie, aux ressources de votre projet
- Modifier le paramètre **src** de l'**ImageView** pour le pointer vers la nouvelle image **vache.jpg**
- Tester le bon fonctionnement de votre application pour s'assurer que l'exécution de l'application ne diffère pas trop de l'**Exercice 01**.

Question 02 : À la fin de la question précédente, nous avons remarqué que le simple ajout d'une nouvelle activité avec son layout n'a pas trop modifié l'exécution de l'application. Dans cette question nous allons voir comment utiliser un **Intent** pour lancer la nouvelle activité **MainActivity** après l'affichage de l'écran d'accueil :

- Dans la classe **SplashActivity**, à la fin de la méthode **onCreate** (après l'invocation de **startAnimation**), créer une nouvelle instance de **Intent**. La classe **Intent** comporte plusieurs constructeurs. On utilisera celui prenant deux paramètres :
 - **Context (this)**
 - **Class (MainActivity.class)**.
- Invoquer **startActivity** avec le nouvel **Intent** comme paramètre
- Tester le fonctionnement de l'application. Que remarque-t-on?

Question 03 : À la fin de la question précédente, si vous avez créé l'activité

MainActivity manuellement, l'exécution de l'application se termine avec une erreur. Dans la console d'*Android Studio*, on peut voir que l'erreur est causée par une exception **android.content.ActivityNotFoundException**. Cette erreur est causée par l'absence de déclaration de l'activité **MainActivity** dans le fichier **AndroidManifest.xml**.

Dans le fichier **AndroidManifest.xml**, juste après la déclaration de l'activité **SplashActivity**, ajouter la balise suivante :

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
</activity>
```

Tester le bon fonctionnement de l'application. Quelles anomalies peut-on détecter?

Question 04 : Deux anomalies sont détectées à la fin de la question précédente :

1. L'écran d'accueil n'a pas l'air d'apparaître (en réalité, il est très vite masqué par l'image de **MainActivity**)
2. L'image de l'activité **MainActivity** n'occupe pas tout l'écran.

Dans cette question, la première anomalie sera corrigée. Pour ce faire, nous devons attendre la fin de l'animation pour lancer l'activité **ActivityMain**.

À la fin de la méthode **onCreate** de **SplashActivity**, insérer le code suivant :

```
customAnimation.setAnimationListener(this);
```

Compléter le code de la classe **SplashActivity** et déplacer le code de la création d'**Intent** et l'invocation de **startActivity** à un endroit plus adéquat (méthode **onAnimation???).**

Question 05 : Dans cette question, on va modifier l'attribut de l'**ImageView** montrant la photo de l'animal pour qu'elle occupe tout l'écran. Dans le fichier **activity_main.xml** ajouter un attribut **scaleType** à l'**ImageView** et lui donner une valeur pour que l'image soit **centrée** et **coupée (cropped)** si besoin. Changer aussi la valeur de l'attribut **layout_width** de l'image à « **match_parent** ».

Question 06 : Dans cette question, on va implanter la lecture d'un son quand l'utilisateur clique sur l'image de l'animal. Implanter les actions suivantes :

- Créer un dossier **raw** dans le dossier **res**
- Copier le fichier **vache.ogg** fourni dans le dossier **raw**. Désormais cette ressource audio sera accessible dans le code Java à l'aide de son identifiant **R.raw.vache**
- Dans la classe **MainActivity**, à la fin de la méthode **onCreate**, créer une variable **animalImage** et utiliser **findViewById** pour la faire pointer vers l'**ImageView** correspondante du fichier **activity_main.xml**
- Ajouter **this** comme **OnClickListener** à **animalImage**
- Ajouter un attribut privé **mp** de type **MediaPlayer** à la classe **MainActivity**
- Ajouter la méthode suivante à la classe **MainActivity** après s'être assuré qu'elle implémente l'interface **OnClickListener** :

```
@Override
public void onClick(View v) {
    if (mp != null) {
        mp.release();
    }
    mp = MediaPlayer.create(this, R.raw.cow);
    mp.start();
}
```

- Tester le bon fonctionnement de l'application

Question 07 : Perfectionner l'application en implantant les actions suivantes :

- Insérer à la fin de **onCreate** la ligne suivante qui permet à l'utilisateur de régler le volume Média quand dans l'application :
`setVolumeControlStream (AudioManager.STREAM_MUSIC);`
- Augmenter le numéro de version de l'application pour qu'il soit égal à « **2.0** ».
- Causer l'arrêt du son produit quand l'utilisateur quitte l'application (autrement dit quand l'activité **MainActivity** est suspendue)
- Terminer l'activité **SplashActivity** après le lancement de l'activity **MainActivity** (méthode **finish**).

Exercice 03 ***Plus d'Animaux et de Sons***

L'objectif de cet exercice est d'enrichir l'application de l'**exercice 02** en ajoutant plus de photos d'animaux et de sons. Le clic sur la photo d'un animal provoquera l'émission du son que cet animal fait et causera le changement de la photo pour afficher un autre animal. Ainsi de suite, à la fin, on boucle vers le premier animal. Les différentes photos et les différents fichiers audio sont fournis.

Nous souhaitons également changer l'application pour qu'une nouvelle exécution affiche la dernière image à laquelle est arrivé l'utilisateur lors de l'exécution précédente. Ceci requiert le stockage **persistant** d'une information sur l'appareil.

Les concepts manipulés pendant cet exercices sont :

- Stockage de l'état de l'application en utilisation **SharedPreferences**
- Changer l'image d'une **ImageView** depuis le code

Question 01 : Ajouter les images et les sons fournis pour cet exercice comme nouvelles ressources dans votre projet « **La Vache Fait Meuh** ».

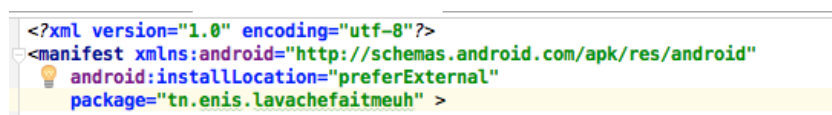
Question 02 : Pour pouvoir manipuler facilement les images et les sons, nous avons besoin de déclarer les ressources correspondantes sous la forme de deux tableaux comme suit. Le texte des deux tableaux est fournis dans le fichier **tableaux.txt** :

```
private static int[] imagesIds = {
    R.drawable.ours,      R.drawable.chat,      R.drawable.poule,      R.drawable.chimpanse,
    R.drawable.vache,     R.drawable.chien,     R.drawable.ane,        R.drawable.elephant,
    R.drawable.grenouille, R.drawable.chevre,    R.drawable.oie,        R.drawable.cheval,
    R.drawable.chaton,    R.drawable.lion,      R.drawable.singe,      R.drawable.cochon,
    R.drawable.coq,       R.drawable.foque,     R.drawable.mouton,     R.drawable.tigre,
    R.drawable.dinde,     R.drawable.baleine,   R.drawable.loup
};

private static int[] sonsIds = {
    R.raw.ours,    R.raw.chat,    R.raw.poule,    R.raw.chimpanse,    R.raw.vache,
    R.raw.chien,   R.raw.ane,     R.raw.elephant, R.raw.grenouille,   R.raw.chevre,
    R.raw.oie,     R.raw.cheval, R.raw.chaton,   R.raw.lion,         R.raw.singe,
    R.raw.cochon,  R.raw.coq,    R.raw.foque,    R.raw.mouton,       R.raw.tigre,
    R.raw.dinde,   R.raw.baleine, R.raw.loup
};
```

Ajouter les deux tableaux ci-dessus dans la classe **MainActivity.java**

Question 03 : Étant donné que la taille de notre application devient importante, il serait judicieux de préférer son installation sur le support externe d'un appareil Android. Pour implanter cela, il suffit de modifier l'attribut **android:installLocation** du fichier **AndroidManifest.xml**. Fixer la valeur de cet attribut à « **preferExternal** » comme illustré par l'image ci-dessous :



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:installLocation="preferExternal"
    package="tn.enis.lavachefaitmeuh" >
```

Question 04 : Pour pouvoir stocker l'index de la dernière photo affichée d'une manière persistante, nous allons utiliser les **SharedPreferences** qui permettent de stocker des données privée de types primitifs (entier, réel, booléen, chaîne de caractère, etc) sous la forme de couples (*clefs, valeur*). Elles sont généralement utilisées pour partager des informations entre deux activités différentes ou bien entre deux instances d'exécution différentes de la même application.

La classe **SharedPreferences** fournit un framework qui permet de stocker et de retrouver des couples (*clef, valeurs*) d'une manière persistante. Elle est utilisée de cette manière :

- Récupérer une instance de type **SharedPreferences**. Ceci peut être effectué de deux manières différentes :
 - En utilisant la méthode **getSharedPreferences** de la classe **Activity**. Cette méthode est utilisée dans le cas où l'application possède plusieurs ensembles de préférences. Le nom de l'ensemble sera spécifié dans le premier paramètre
 - En utilisant la méthode **getPreferences** de la classe **Activity**. Cette méthode est utilisée dans le cas où l'application possède un seul ensemble de préférences. C'est cette méthode qui sera utilisée dans cet exercice
- Lire une valeur de préférence. Pour ce faire, il faut utiliser une des nombreuses méthodes de la classe **SharedPreferences** (**getInt**, **getBoolean**, **getString**, etc.). Ces méthodes prennent deux paramètres :
 - Le nom de la clef sous la forme d'une chaîne de caractères
 - Une valeur par défaut qui sera retournée si aucune information stockée n'a été trouvée

Pour enregistrer une information, il faut :

- Récupérer un **Editor** en invoquant la méthode **edit** sur l'instance de **SharedPreferences**
- Utiliser l'**Editor** en invoquant une des nombreuses méthodes fournies (**putInt**, **putBoolean**, **putString**, etc.)
- Invoquer la méthode **commit** sur l'**Editor**. Sans cette dernière invocation, l'information ne sera pas stockée d'une manière persistante.

Implanter les actions suivantes dans la classe **MainActivity** :

- Créer un attribut privé **preferences** qui sera initialisé dans la méthode **onCreate** avec la méthode **getPreferences**. Le seul paramètre à fournir à cette méthode est le de partage de la préférence (choisir un mode parmi **MODE_PRIVATE**, **MODE_WORLD_READABLE** ou **MODE_WORLD_WRITEABLE**).
- Déclarer les attributs suivants dans la classe **MainActivity**. Ils serviront à stocker respectivement la valeur de la clef pour la préférence, la position de départ et la position courante :

```
private static final String POSITION_COURANTE = "positionCourante";
private static final int POSITION_DEPART = 0;
private int positionCourante;
```

Question 05 : Après l'initialisation de **positionCourante**, qui représentera un index dans le tableau **imagesIds**, on va changer l'image affichée par l'**ImageView** pour afficher l'image correspondant à cet index. Pour cela, on va implanter une nouvelle méthode dans la classe **MainActivity** qui simplifiera cette tâche. Ajouter la méthode suivante dans **MainActivity** :

```
private Drawable getDrawableByIndex (int index) {
    return getResources().getDrawable(imagesIds[index]);
}
```

Selon votre configuration de l'API minimale lors de la création du projet, il se peut que Android Studio vous signale que la méthode **getDrawable** est obsolète comme c'est le cas dans la figure ci-dessus. Ce n'est pas très grave pour le TP. Mais pour un projet professionnel, il faut remplacer l'utilisation de

toute méthode obsolète par son remplaçant tout en gardant une compatibilité avec les versions antérieures.

Question 06 : À la fin de la méthode **onCreate** de la classe **MainActivity**, récupérer la valeur de la dernière position courante enregistrée et spécifier l'image courante avec l'index obtenu :

```
positionCourante = preferences.getInt (POSITION_COURANTE, POSITION_DEPART);  
animalImage.setImageDrawable (getDrawableByIndex (positionCourante));
```

Question 07 : Modifier les méthodes **onClick** et **onCompletion** de la classe **MainActivity** pour réaliser le comportement suivant :

- Jouer le son correspondant à l'animal dont l'image est affichée
- Passer à l'image suivante quand la lecture du son se termine. Arrivé à la dernière image, l'image suivante sera la première
- Enregistrer la nouvelle position à l'aide des **SharedPreferences**
- Modifier le numéro de version de l'application pour le passer à « **3.0** ».

Tester le bon fonctionnement de l'application.

Exercice 04 **Défilement Manuel des Images**

L'objectif de cet exercice est d'implanter le défilement manuel entre les photos des animaux. Le point d'entrée est l'application « **La Vache Fait Meuh** » résultat de l'**exercice 03**. On modifiera cette application pour réaliser le défilement de droite à gauche et de gauche à droite des photos des animaux.

Comme d'habitude, le clic sur une photo produira le son que fait l'animal correspondant. Pour ce faire, le widget **ImageView** sera remplacé par un widget **ImageSwitcher** car ce dernier supporte les actions de défilement (événement **OnTouch**). On utilisera aussi deux animation personnalisées pour simuler une sortie et une entrée d'images lors du défilement.

Les concepts manipulés pendant cet exercices sont :

- Utilisation d'un **ImageSwitcher**
- Gestion des événements **OnTouch**
- Utilisation d'animations personnalisées pour simuler une transition d'images

Le widget **ImageSwitcher** permet à une activité d'ajouter une animation entre deux widgets **ImageView**. Il utilise une instance de **ViewFactory** pour contrôler les différents widgets **ImageView**. Par conséquent, pour initialiser correctement un **ImageSwitcher**, il faut lui associer une **ViewFactory** et utiliser sa méthode **makeView** pour contrôler l'image.

Question 01 : Modifier le fichier de layout **activity_main.xml** en remplaçant le widget **ImageView** par un widget **ImageSwitcher** comme suit :

```
<ImageSwitcher
    android:id="@+id/image_switcher"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clickable="true"/>
```

À ce stade, des erreurs de syntaxes devraient apparaître dans la méthode **onCreate** dues essentiellement à la différence dans le nom d'identifiant. Ces erreurs seront progressivement corrigées dans les questions qui suivent.

Question 02 : À la fin de l'activité **MainActivity**, insérer le code de la classe ci-dessous, qui jouera le rôle de factory pour l'**ImageSwitcher** ajouté :

```
private class MyImageSwitcherFactory implements ViewSwitcher.ViewFactory {
    public View makeView() {
        ImageView imageView = new ImageView(MainActivity.this);
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setLayoutParams(
            (new ImageSwitcher.LayoutParams
             (ImageSwitcher.LayoutParams.MATCH_PARENT,
              ImageSwitcher.LayoutParams.MATCH_PARENT));
        return imageView;
    }
}
```

Comme on peut le remarquer, il s'agit d'une manière « **Java** » pour coder dynamiquement l'insertion de widgets d'une similaire à la méthode « **XML** ».

Question 03 : Dans la méthode **onCreate**, récupérer une référence vers l'**ImageSwitcher** et utiliser la méthode **setFactory** pour lui attribuer une nouvelle instance de la classe **MyImageSwitcherFactory**. Enfin, ajuster le reste du code de la méthode **onCreate** pour corriger les différentes erreurs.

À ce stade, l'application obtenue est fonctionnellement identique à celle de l'**exercice 03**. Tester son fonctionnement.

Question 04 : Dans cette question (la plus dure de l'exercice), on remplacera l'écoute des événements *Click* par une écoute d'événements *Touch*. Modifier l'entête de la classe **MainActivity** en remplaçant **onClickListener** par **onTouchListener** et modifier le reste de la classe pour corriger les erreurs survenues.

La méthode **onTouch** ajoutée à la place de la méthode **onClick** comporte deux paramètres : un **View** et un **MotionEvent**. Le second servira à connaître la nature exacte de l'évènement et de décider de l'action à effectuer. Les méthodes qui nous intéressent pour le cadre de cet exercice sont :

- **public final int** `getAction()` qui retourne l'action effectuée. Les valeurs qui nous intéressent sont :
 - `MotionEvent.ACTION_DOWN` : Doigt enfoncé
 - `MotionEvent.ACTION_UP` : Doigt levé
- **public final float** `getX()` qui retourne l'abscisse du pointeur lors du déclenchement de l'évènement.

Implanter le pseudo algorithme suivant dans la méthode **onTouch** :

- Connaître l'action correspondant à l'évènement
- Si l'action est **ACTION_DOWN**, enregistrer l'abscisse
- Si l'action est **ACTION_UP**, enregistrer l'abscisse puis :
 - Si la différence (en valeur absolue) entre les deux abscisses enregistrées est supérieure à un certain seuil minimal (50dp par exemple), alors défiler vers l'image correspondante selon le signe de la différence, tout en gérant les **SharedPreferences** comme pour l'**exercice 03**.
Il faudra, comme d'habitude, boucler vers la première image si on atteint la dernière et, c'est nouveau, **aller à la dernière si on recule depuis la première**.
 - Si la différence (en valeur absolue) est inférieure au seuil fixé, traiter évènement comme un clic produire le son correspondant.

Tester le fonctionnement de l'application.

Question 05 : Dans cette question, on va animer le défilement entre les images. Ajouter les ressources **XML** fournies, **entree_droite.xml**, **entree_gauche.xml**, **sortie_droite.xml** et **sortie_gauche.xml** dans le projet comme animations personnalisées. Ces ressources permettront de simuler une transition selon le défilement voulu par l'utilisateur.

Modifier le code de la méthode **onTouch** pour invoquer les méthodes **setInAnimation** et **setOutAnimation** sur l'**ImageSwitcher** pour réaliser le comportement suivant :

- En cas de défilement **droite → gauche** : l'image sortante sortira par la gauche et l'image entrante entrera par la droite
- En cas de défilement **gauche → droite** : l'image sortante sortira par la droite et l'image entrante entrera par la gauche

Question 06 : Peaufiner l'application en implantant les actions suivantes :

- Arrêt du son lors d'un défilement ou d'un nouveau clic
- Modification du numéro de version « **4.0** »

Tester le bon fonctionnement de l'application.