



WEBFORCE
BE THE CHANGE



TRAVAUX PRATIQUES – FILIÈRE DÉVELOPPEMENT DIGITAL

Option – Applications mobiles

M209 – Acquérir les bases de développement Android



70 heures



SOMMAIRE

1. DÉCOUVRIR LE DÉVELOPPEMENT MOBILE

- Etude de cas : Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique

2. MAÎTRISER LA PLATEFORME ANDROID

- Activité n°1 : Lancer une application mobile sur smartphone
- Activité n°2 : Configurer les flavors et installer plusieurs applications sur smartphone

3. CODER EN JAVA

- Activité n°1 : Créer une application mobile JAVA en appliquant la POO
- Activité n°2 : Créer une application mobile avec interface graphique simple
- Activité n°3 : Utiliser Git pour manipuler le code des deux applications créées

MODALITÉS PÉDAGOGIQUES



WEBFORCE
BE THE CHANGE



1

LE GUIDE DE SOUTIEN
Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF
Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life



3

DES CONTENUS TÉLÉCHARGEABLES
Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life



4

DU CONTENU INTERACTIF
Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life



5

DES RESSOURCES EN LIGNES
Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage



WEBFORCE
BE THE CHANGE



PARTIE 1

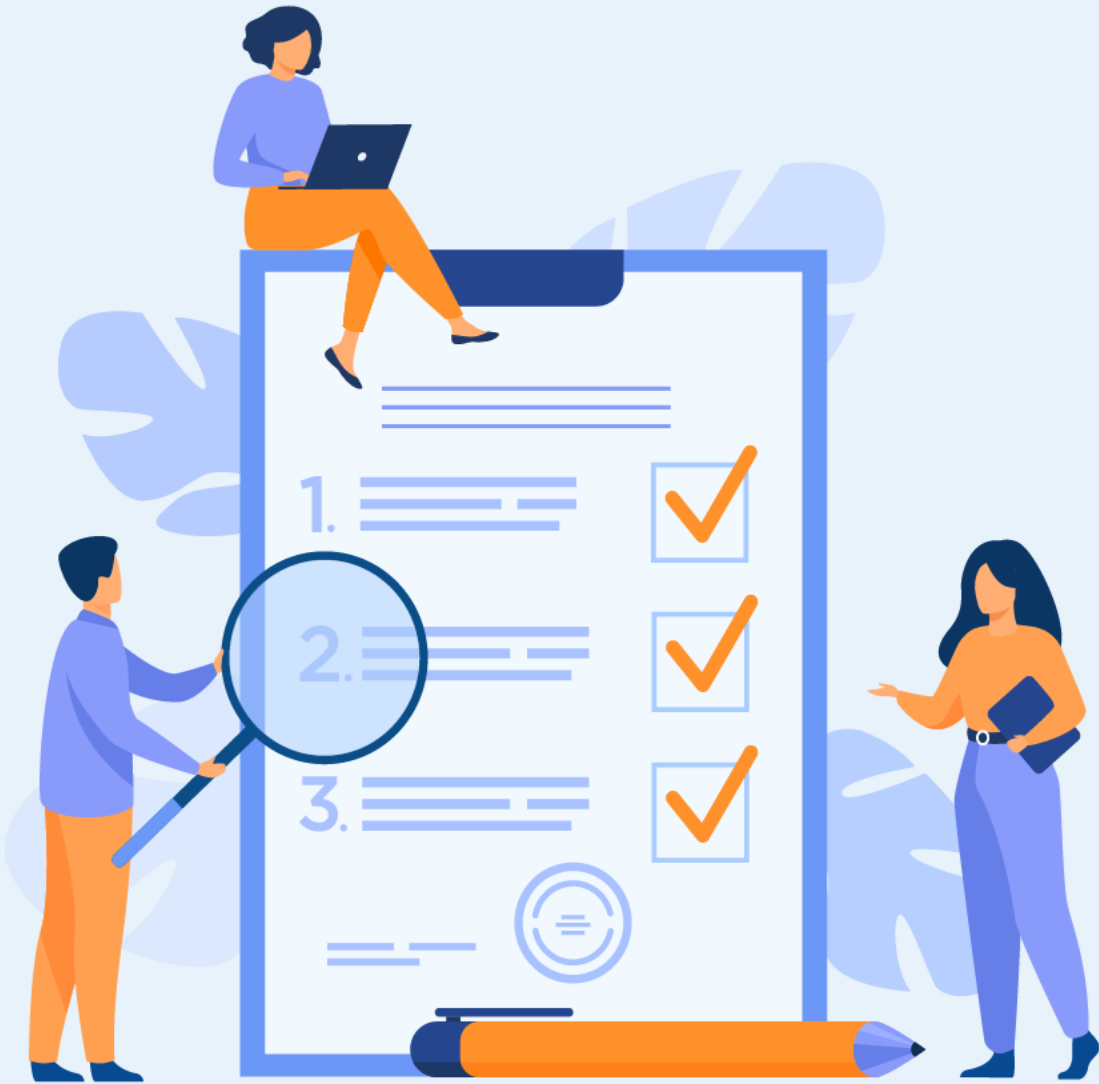
Découvrir le développement mobile

Dans ce module, vous allez :

- Explorer les bases du développement mobile
- Acquérir des connaissances sur les approches de développement mobile
- Découvrir l'écosystème de développement mobile



8 heures



Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique

Compétences visées :

- Découvrir le développement mobile
- Maîtriser les approches de développement mobile
- Concevoir les maquettes d'un projet mobile

Recommandations clés :

- Lire attentivement l'énoncé de l'étude de cas
- Utiliser le résumé théorique



8 heures

CONSIGNES

Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours
- Demander aux apprenants de réaliser le travail de synthèse et de faire une présentation
- Constituer des groupes de travail

Pour l'apprenant :

- S'autoformer sur un outil pour concevoir les maquettes d'une application mobile
- Collaborer dans un groupe de 4 à 5 stagiaires

Conditions de réalisation :

- Support de résumé théorique accompagnant
- Installation d'un outil pour concevoir les maquettes (exemple: Balsamiq)

Critères de réussite :

- Le stagiaire est-il capable de :
 - Choisir l'approche de développement mobile adéquat pour la mise en place d'une application spécifique
 - Distinguer entre les types d'applications mobiles
 - Concevoir les maquettes d'une application mobile



01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique



Etude de cas

Une société de services souhaite mettre en place une application mobile pour ses employés, et elle a fait recours à votre expertise, afin de proposer l'approche et l'environnement technique les plus adéquats pour sa mise en œuvre.

But de projet :

Le but de ce projet est de développer une application de géolocalisation, cette dernière permet de localiser la position des contacts du répertoire téléphonique situés dans un rayon donné, à l'aide d'une carte. Elle offre également la possibilité de communiquer avec d'autres personnes connectées au réseau avec la même application, en échangeant des messages texte et des fichiers média (ex : photos, vidéos), et enfin elle donne la possibilité de prendre des photos et de les transmettre via l'application à d'autres contacts.

01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique



Etude de cas

Questions :

1. Citer les besoins fonctionnels à développer ;
2. Citer les besoins non fonctionnels ;
3. Quelles sont les APIs natives auxquelles l'application aura accès ;
4. Citer les approches candidates pour développer cette application. Justifier votre réponse ;
5. En appliquant l'arbre de décision, quelle est la meilleure approche ? Justifier votre réponse pour chaque étape ;
6. Proposer un environnement technique selon l'approche sélectionnée.

01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique

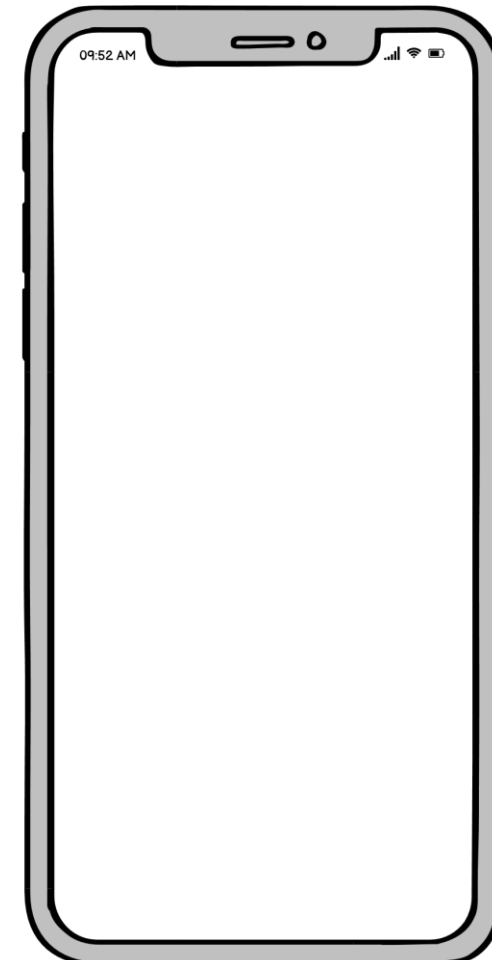


Etude de cas

7. Elaborer les maquettes pour l'application mobile.

- Vous pouvez choisir un outil de cette liste ou autre :

- Balsamiq
- Uizard
- Figma
-



01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique



Solution

1. Besoins fonctionnels :

- Notification par alerte et vibration ;
- Localiser la position d'un contact ;
- Communiquer avec les autres contacts ;
- Partager les fichiers (audio, vidéo) ;
- Prise des photos ;
- Partage des photos.

2. Besoins non fonctionnels :

- Disponible sur Android et iOS ;
- Développement à faible coût ;
- Déployable sur les app stores.

3. APIs natives :

- Localisation (GPS) ;
- Contacts ;
- Téléphonie ;
- Caméra ;
- Média.

4. Approches candidates :

- Native
- Multiplateforme

Justification :

- Puisque l'application sera publiée dans les app stores : on élimine l'approche web.
- Puisque l'application fait recours à plusieurs API natives : on élimine l'approche hybride.

01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique



Solution

5. Pour trancher entre l'approche native et le multiplateforme, vous devez poser la question suivante :

- Est-ce que la société dispose de développeurs mobile Android et iOS ?
 - Si oui , on opte pour l'approche native.
 - Si non, vous devez évaluer les Framework pour le développement multiplateforme, pour voir s'ils offrent un accès aux fonctionnalités natives mentionnées dans la question 3.

6. Environnement de développement :

Native	Multiplateforme
<p>Android :</p> <ul style="list-style-type: none">• Java / Kotlin• Android Studio <p>iOS</p> <ul style="list-style-type: none">• Swift / Objective C• Xcode	<p>Flutter :</p> <ul style="list-style-type: none">• Dart• Android Studio

01 – Etude de cas

Choisir l'approche de développement mobile adéquate pour la mise en place d'une application mobile spécifique



Solution

7. Chaque groupe doit proposer ses maquettes pour :

- Authentification ;
- Liste des contacts ;
- Visualisation de la position d'un contact ;
- Prise de photo ;
- Partage des fichiers ;
- Espace de chat ;
- Visualisation les notifications.



PARTIE 2

Maîtriser la plateforme Android

Dans ce module, vous allez :

- Préparer l'environnement de développement
- Vous initier à Android Studio
- Maîtriser la génération d'une application mobile



16 heures



ACTIVITÉ n° 1

Lancer une application mobile sur smartphone

Compétences visées :

- Installer et utiliser l'IDE Android Studio
- Utiliser le processus de développement pour créer des applications Android
- Créer un projet Android à partir d'un modèle
- Intégrer des messages de journal à l'application à des fins de débogage

Recommandations clés :

- Révision générale du résumé théorique
- Suivre les instructions du TP et organiser le dossier de travail



8 heures



WEBFORCE
BE THE CHANGE

CONSIGNES

1. Pour le formateur

- Demander aux stagiaires de suivre les étapes de l'activité.

2. Pour l'apprenant

- Dans cette activité, vous apprendrez comment :
 - installer Android Studio, l'environnement de développement Android
 - lancez votre première application Android, Hello World, sur un émulateur et sur un périphérique physique

3. Conditions de réalisation :

- Un ordinateur sous Windows ou Linux ou un Mac sous MacOS
- Accès Internet ou autre moyen de charger les dernières installations d'Android Studio et Java sur votre ordinateur.

4. Critères de réussite :

- Le stagiaire doit être capable de :
 - Installer l'environnement de développement Android Studio
 - Créer un émulateur (périphérique virtuel) pour exécuter votre application sur votre ordinateur
 - Créer et exécuter l'application Hello World sur les périphériques virtuels et physiques
 - Explorer la disposition du projet et le fichier AndroidManifest.xml
 - Générer et afficher les messages de journalisation à partir de votre application



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

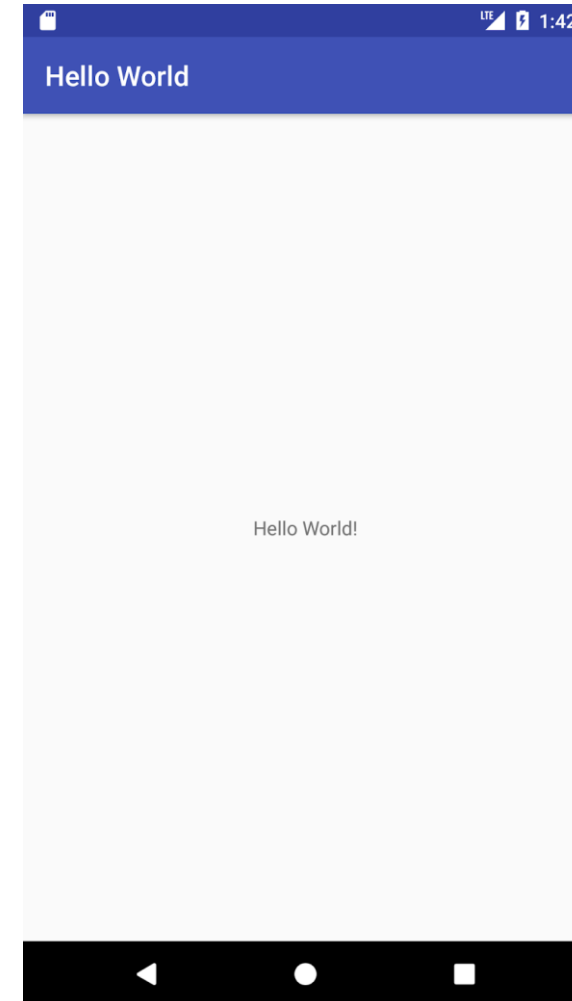
Android Studio et Hello World



Aperçu de l'application

Une fois Android Studio installé avec succès, vous créez, à partir d'un modèle, un nouveau projet pour l'application Hello World. Cette application simple affiche la chaîne "Hello World" sur l'écran du périphérique virtuel ou physique Android.

Voici à quoi ressemblera l'application finie :



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Android Studio et Hello World



Installer Android Studio

Android Studio est disponible pour les ordinateurs fonctionnant sous Windows ou Linux et pour les Mac exécutant macOS. Le plus récent OpenJDK (Java Development Kit) est livré avec Android Studio.

Pour être opérationnel avec Android Studio, commencez par vérifier [la configuration système requise](#) pour vous assurer que votre système les respecte. L'installation est similaire pour toutes les plateformes. Toutes les différences sont notées ci-dessous :

1. Accédez au [site des développeurs Android](#) et suivez les instructions pour télécharger et [installer Android Studio](#) ;
2. Acceptez les configurations par défaut pour toutes les étapes et assurez-vous que tous les composants sont sélectionnés pour l'installation ;
3. Une fois l'installation terminée, l'assistant d'installation télécharge et installe des composants supplémentaires, notamment le SDK Android. Soyez patient, cela peut prendre un certain temps en fonction de votre débit Internet, et certaines étapes peuvent sembler redondantes ;
4. Une fois le téléchargement terminé, Android Studio démarre et vous êtes prêt à créer votre premier projet.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World

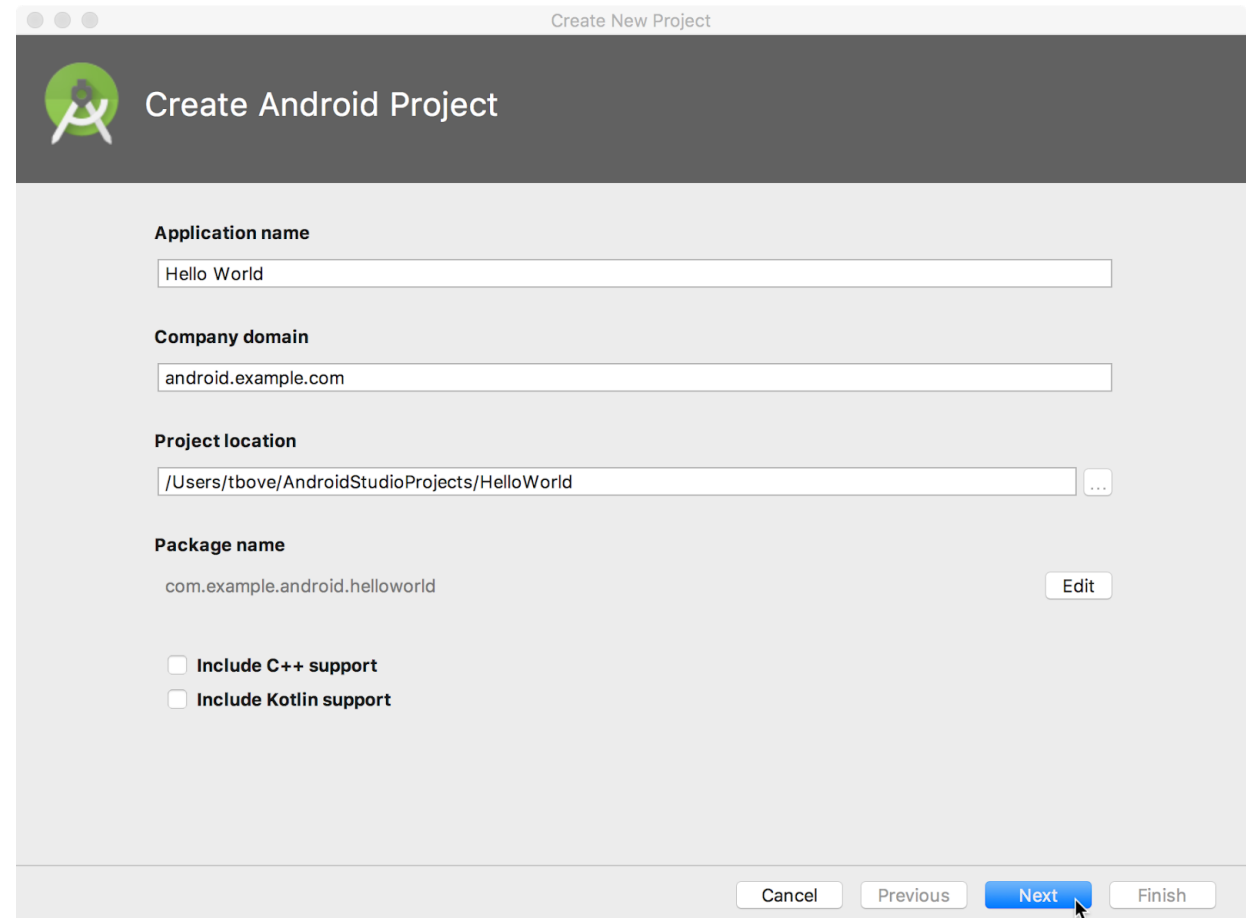


Etape 1 : Créer le projet d'application

Dans cette tâche, vous allez créer une application qui affiche "Hello World" pour vérifier que le studio Android est correctement installé et pour apprendre les bases du développement avec Android Studio.

Étapes à suivre :

1. Ouvrez Android Studio s'il n'est pas déjà ouvert ;
2. Dans la fenêtre principale **Welcome to Android Studio**, cliquez sur **Start a new Android Studio project** ;
3. Dans la fenêtre **Create Android Project**, entrez **Hello World** pour le nom de l'application.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 1 : Créer le projet d'application

4. Vérifiez que **Project location** par défaut est l'emplacement où vous souhaitez stocker votre application Hello World et d'autres projets Android Studio, ou modifiez-le à votre répertoire préféré ;

5. Acceptez le package **android.example.com** par défaut pour **Company Domain** ou créez un domaine de société unique.

Create New Project

Create Android Project

Application name
Hello World

Company domain
android.example.com

Project location
/Users/tbove/AndroidStudioProjects/HelloWorld

Package name
com.example.android.helloworld Edit

Include C++ support
 Include Kotlin support

Cancel Previous Next Finish



Remarques

Si vous ne prévoyez pas de publier votre application, vous pouvez accepter la valeur par défaut. Sachez que la modification ultérieure du nom du package de votre application représente un travail supplémentaire.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 1 : Créer le projet d'application

6. Ne cochez pas les options **Include C ++ support** et **Include Kotlin support**, puis cliquez sur **Next** ;
7. Sur l'écran **Target Android Devices, Phone and Tablet** doit être sélectionné. Assurez-vous que **l'API 21: Android 5.0 (Lollipop)** est défini comme SDK minimal; Si ce n'est pas le cas, utilisez le menu contextuel pour le définir.

Au moment d'écrire ces lignes, ces paramètres rendent votre application Hello World compatible avec **98,6%** des appareils Android actifs sur le Google Play Store.

New Project

Empty Activity

Creates a new empty activity

Name: Hello World

Package name: com.example.android.helloworld

Save location: C:\Users\Lachgar\AndroidStudioProjects\HelloWorld

Language: Java

Minimum SDK: API 21: Android 5.0 (Lollipop)

i Your app will run on approximately **98,6%** of devices.
[Help me choose](#)

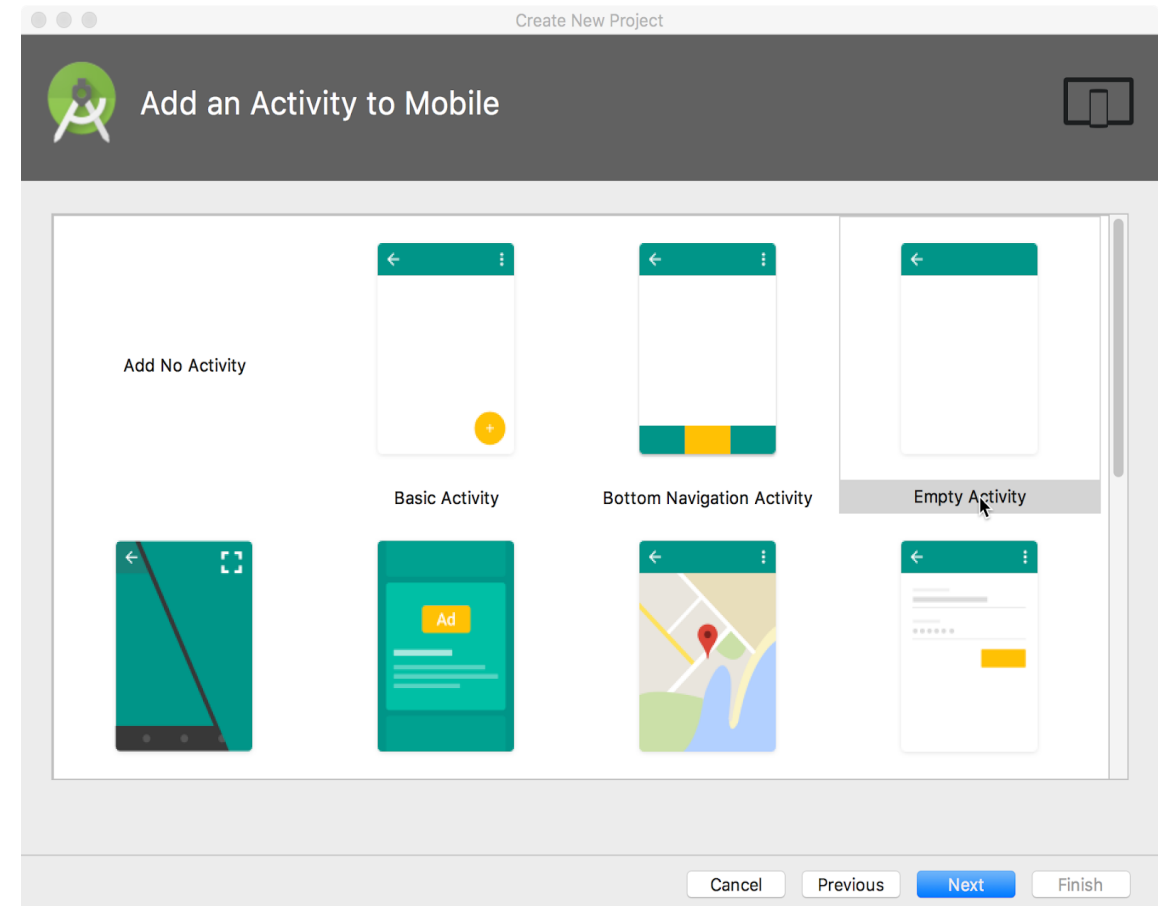
Previous Next Cancel **Finish**

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World

Etape 1 : Créer le projet d'application

- Ne cochez pas la case **Include Instant App support** et toutes les autres options. Puis cliquez sur **Next**. Si votre projet nécessite des composants supplémentaires pour le SDK cible choisi, Android Studio les installera automatiquement ;
- La fenêtre **Add an Activity** apparaît. Une **Activity** est une tâche unique et ciblée que l'utilisateur peut faire. C'est un élément crucial de toute application Android. Une **Activity** est généralement associée à une présentation qui définit la manière dont les éléments de l'interface utilisateur apparaissent sur un écran. Android Studio fournit des modèles d'Activity pour vous aider à démarrer. Pour le projet Hello World, choisissez **Empty Activity** comme indiqué ci-dessous, puis cliquez sur **Next**.



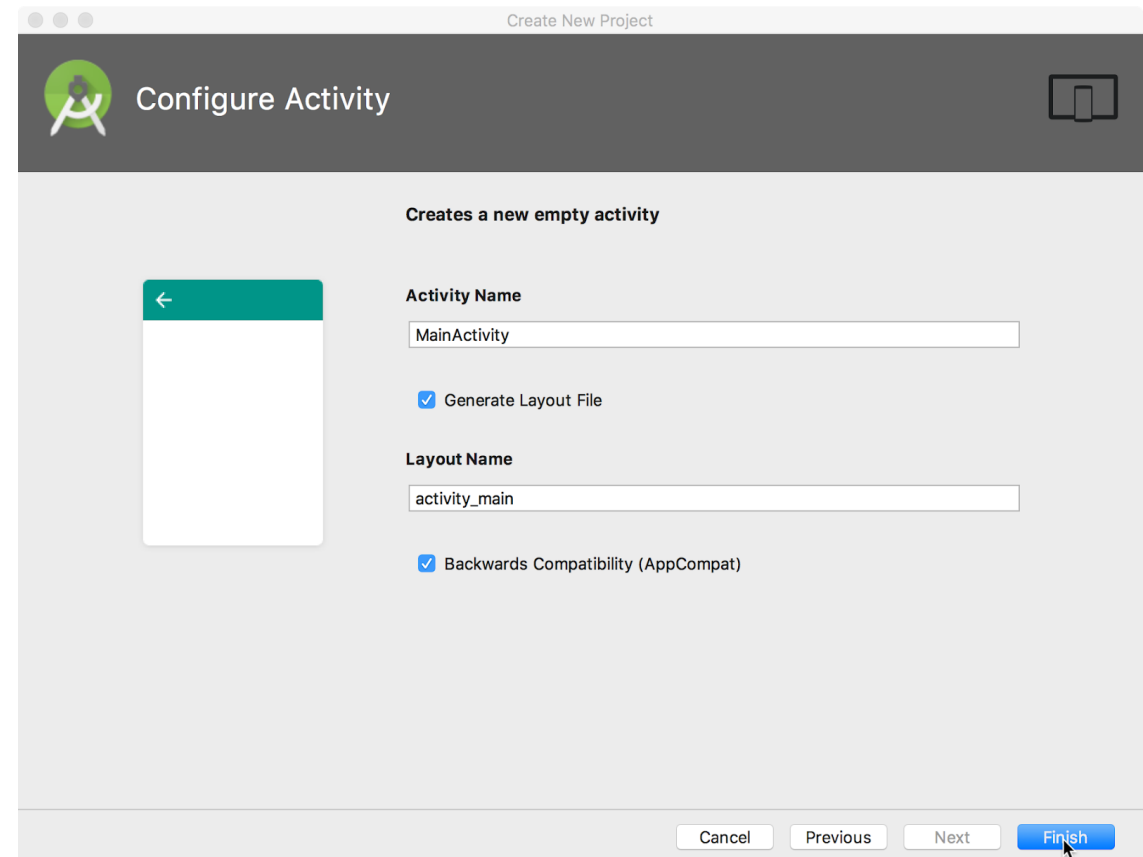
ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 1 : Créer le projet d'application

10. L'écran **Configure Activity** apparaît (selon le modèle que vous avez choisi à l'étape précédente). Par défaut, l'activité vide fournie par le modèle s'appelle **MainActivity**. Vous pouvez changer cela si vous voulez, mais cette leçon utilise **MainActivity** ;
11. Assurez-vous que l'option **Generate Layout file** est cochée. Le nom de la présentation par défaut est **activity_main**. Vous pouvez changer cela si vous voulez, mais cette leçon utilise **activity_main** ;
12. Assurez-vous que l'option **Backwards Compatibility** (App Compat) est cochée. Cela garantit que votre application sera rétrocompatible avec les versions précédentes d'Android.
13. Cliquez sur **Finish**.



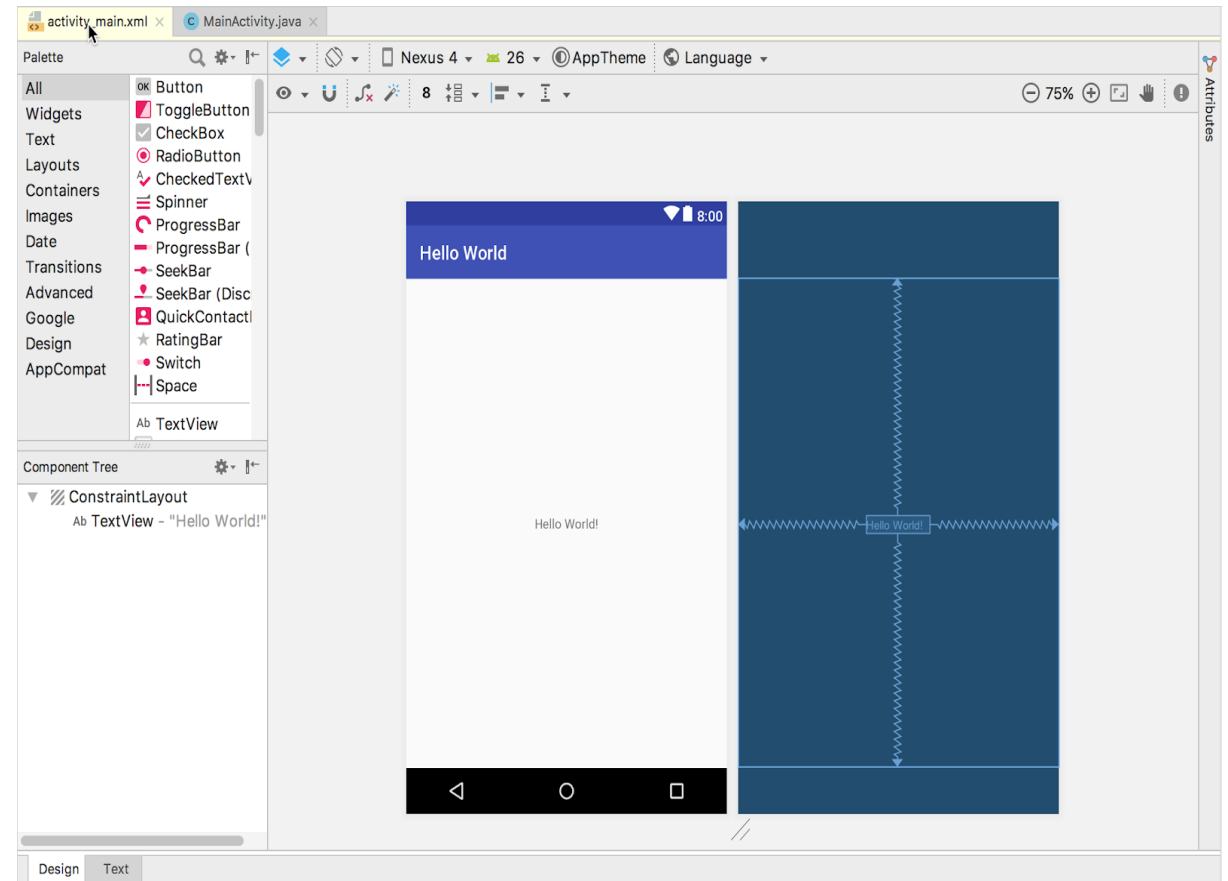
ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World

Etape 1 : Créer le projet d'application

L'éditeur Android Studio apparaît. Suivez ces étapes:

14. Cliquez sur l'onglet **activity_main.xml** pour afficher l'éditeur de « layout » ;
15. Cliquez sur l'onglet **Design** de l'éditeur de présentation, s'il n'est pas déjà sélectionné, pour afficher un rendu graphique de la présentation, comme indiqué ci-dessous.



Remarques

- Android Studio crée un dossier pour vos projets et le construit avec [Gradle](#) (cela peut prendre quelques instants);
- Voir la page [Configure your build](#) pour des informations détaillées.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 1 : Créer le projet d'application

16. Cliquez sur l'onglet **MainActivity.java** pour afficher l'éditeur de code, comme indiqué dans la figure.

```
1 package com.example.android.helloworld;
2
3 import ...
4
5
6 public class MainActivity extends AppCompatActivity {
7
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13 }
14
```

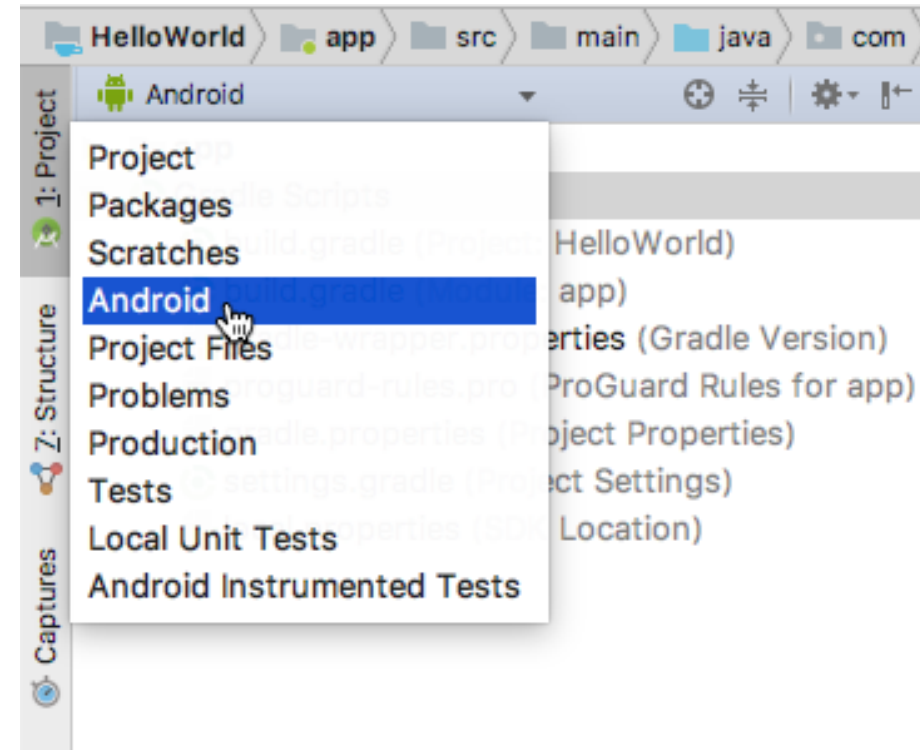

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World

Etape 2 : Explorer le volet Projet > Android

Dans cette pratique, vous explorerez comment le projet est organisé dans Android Studio.

1. Si ce n'est déjà fait, cliquez sur l'onglet **Project** dans la colonne d'onglets verticale du côté gauche de la fenêtre d'Android Studio. La fenêtre **Project** apparaît ;
2. Pour afficher le projet dans la hiérarchie de projets Android standard, choisissez **Android** dans le menu contextuel en haut de la sous-fenêtre **Project**, comme indiqué ci-dessous.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 3 : Explorer le dossier Gradle Scripts

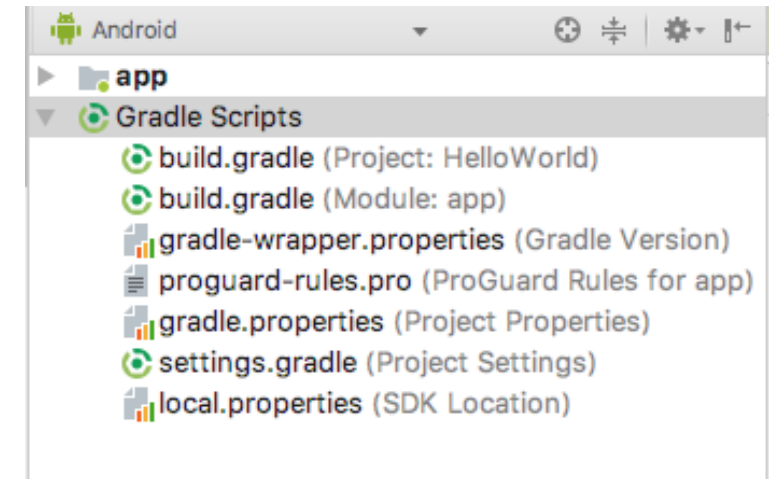
Le système de génération **Gradle** dans Android Studio facilite l'inclusion de binaires externes ou d'autres modules de bibliothèque dans votre génération en tant que dépendances.

Lorsque vous créez un projet pour la première fois, le volet **Projet > Android** apparaît avec le dossier **Gradle Scripts** développé comme indiqué ci-dessous.

Suivez ces étapes pour explorer le système **Gradle** :

1. Si le dossier **Gradle Scripts** n'est pas ouvert, cliquez sur le triangle pour l'ouvrir. Ce dossier contient tous les fichiers nécessaires au système de construction ;
2. Recherchez le fichier **build.gradle (Project: HelloWorld)**.

Vous y trouverez les options de configuration communes à tous les modules composant votre projet. Chaque projet Android Studio contient un seul fichier de construction Gradle de niveau supérieur. La plupart du temps, vous n'avez pas besoin de modifier ce fichier, mais il est toujours utile de comprendre son contenu.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 3 : Explorer le dossier Gradle Scripts

Par défaut, le fichier de construction de niveau supérieur utilise le bloc **buildscript** pour définir les référentiels **Gradle** et les dépendances communs à tous les modules du projet. Lorsque votre dépendance est autre chose qu'une bibliothèque locale ou une arborescence de fichiers, **Gradle** recherche les fichiers dans les référentiels en ligne spécifiés dans le bloc référentiels de ce fichier.

Par défaut, les nouveaux projets Android Studio déclarent que mavenCentral et Google (qui inclut le référentiel Google Maven) sont les emplacements du référentiel (Voir le fichier **settings.gradle**) :

```
repositories {
    gradlePluginPortal()
    google()
    mavenCentral()
}
```

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 3 : Explorer le dossier Gradle Scripts

3. Rechercher le fichier **build.gradle (Module: app)**.

En plus du fichier **build.gradle** au niveau du projet, chaque module possède son propre fichier **build.gradle**, qui vous permet de configurer les paramètres de construction de chaque module spécifique (l'application HelloWorld ne comporte qu'un seul module). La configuration de ces paramètres de génération vous permet de fournir des options de packaging personnalisées, telles que des types de construction et des variantes de produit supplémentaires. Vous pouvez également remplacer les paramètres du fichier **AndroidManifest.xml** ou du fichier **build.gradle** de niveau supérieur.

Ce fichier est le plus souvent le fichier à modifier lors de la modification de configurations au niveau de l'application, telles que la déclaration de dépendances dans la section **dependencies**. Vous pouvez déclarer une dépendance de bibliothèque à l'aide de l'une des différentes configurations de dépendance. Chaque configuration de dépendance fournit à Gradle différentes instructions sur l'utilisation de la bibliothèque.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 3 : Explorer le dossier Gradle Scripts

Voici le fichier **build.gradle (Module: app)** de l'application HelloWorld:

4. Cliquez sur le triangle pour fermer **Gradle Scripts**.

```
plugins {
    id 'com.android.application'
}
android {
    compileSdk 31

    defaultConfig {
        applicationId "ma.projet.android.helloworld"
        minSdk 21
        targetSdk 31
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
            "androidx.test.runner.AndroidJUnitRunner"
    }
}
```

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
            'proguard-rules.pro'
    }
}
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
dependencies {
    implementation 'androidx.appcompat:appcompat:1.4.2'
    implementation 'com.google.android.material:material:1.6.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World

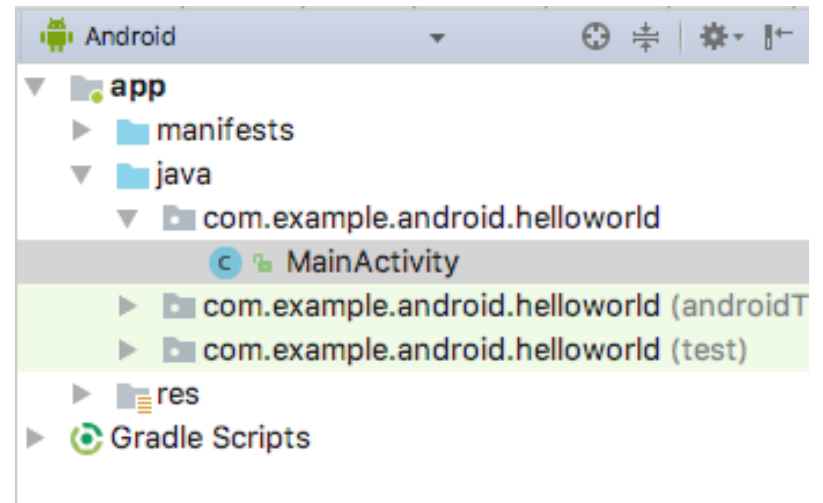


Etape 4 : Explorer les dossiers "app" et "res"

Tout le code et toutes les ressources de l'application se trouvent dans les dossiers **app** et **res**.

1. Ouvrir le dossier **app**, le dossier **java** et le dossier **com.example.android.helloworld** pour afficher le fichier java **MainActivity**. Double-cliquez sur le fichier pour l'ouvrir dans l'éditeur de code.

Le dossier **java** comprend des fichiers de classe Java dans trois sous-dossiers, comme indiqué dans la figure ci-dessus. Le dossier **com.example.android.helloworld** (ou le nom de domaine que vous avez spécifié) contient tous les fichiers d'un package d'application. Les deux autres dossiers sont utilisés pour les tests et décrits dans une autre leçon. Pour l'application Hello World, il n'y a qu'un seul package et il contient **MainActivity.java**. Le nom de la première **Activity** (écran) visible par l'utilisateur, qui initialise également les ressources de l'application, est généralement appelé **MainActivity** (l'extension de fichier est omise dans le volet **Projet > Android**).



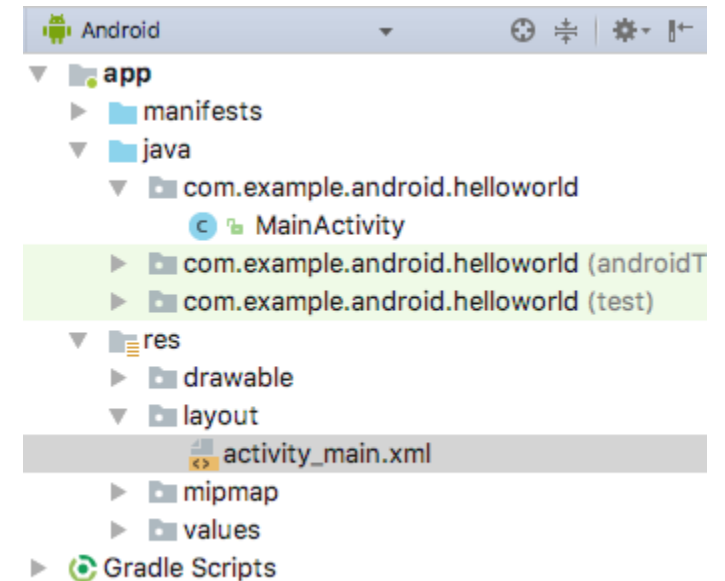
ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 4 : Explorer les dossiers "app" et "res"

- Développez les dossiers **res** et **layout**, puis double-cliquez sur le fichier **activity_main.xml** pour l'ouvrir dans l'éditeur de "layout".



Remarques

- Le dossier **res** contient des ressources, telles que des dispositions, des chaînes et des images. Une activité est généralement associée à une disposition de vues d'interface utilisateur définies sous la forme d'un fichier XML. Ce fichier porte généralement le nom de son activité.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Créer l'application Hello World



Etape 5 : Explorer le dossier "manifestes"

Le dossier **manifests** contient des fichiers qui fournissent des informations essentielles sur votre application au système Android, que le système doit posséder avant de pouvoir exécuter le code de l'application.

1. Développez le dossier **manifest** ;
2. Ouvrez le fichier **AndroidManifest.xml**.


Le fichier **AndroidManifest.xml** décrit tous les composants de votre application Android. Tous les composants d'une application, telles que les activités, doivent être déclarés dans ce fichier XML. Dans d'autres leçons, vous modifierez ce fichier pour ajouter des fonctionnalités et des autorisations de fonctionnalités. Pour une introduction, voir App [Manifest Overview](#).

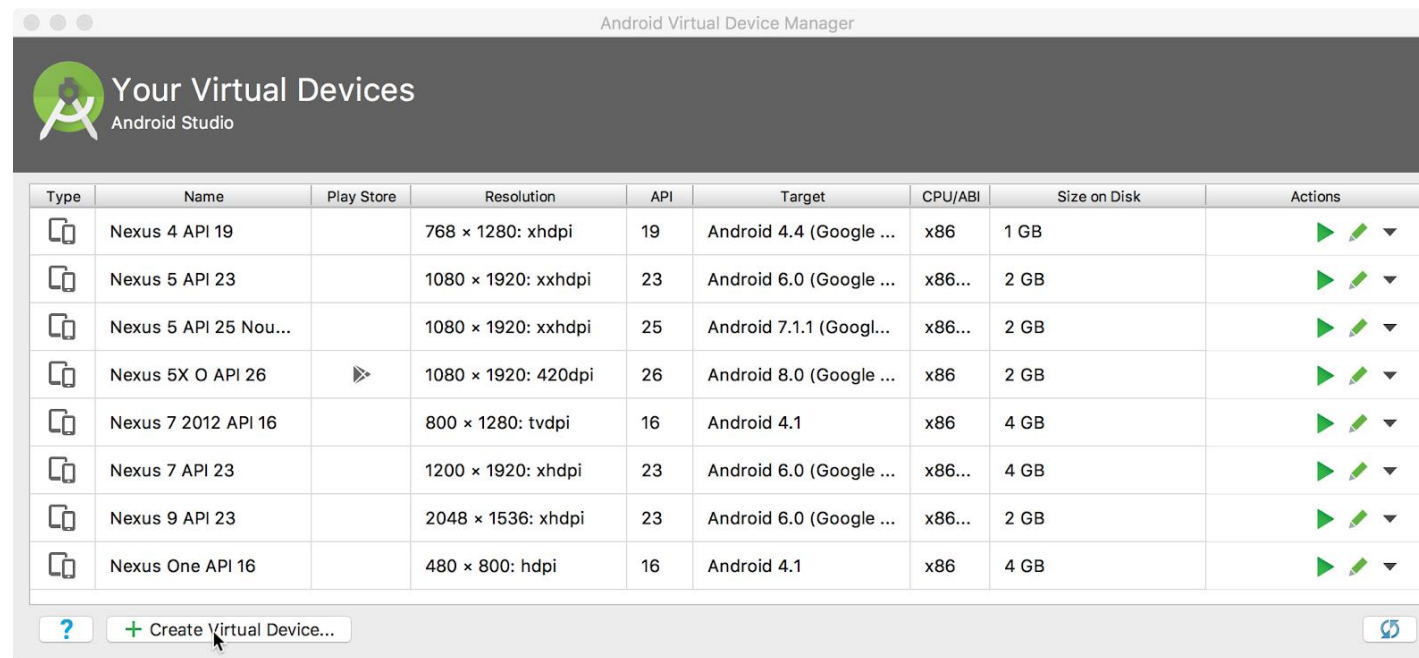
ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique virtuel (émulateur)

Etape 1 : Créer un périphérique virtuel Android (AVD)

Pour exécuter un émulateur sur votre ordinateur, vous devez créer une configuration décrivant le périphérique virtuel.

1. Dans Android Studio, sélectionnez **Tools > Android > AVD Manager** ou cliquez sur l'icône AVD Manager  dans la barre d'outils. L'écran **Your Virtual Devices** apparaît. Si vous avez déjà créé des périphériques virtuels, l'écran les affiche (comme illustré dans la figure ci-dessous). sinon, vous voyez une liste vide.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique virtuel (émulateur)

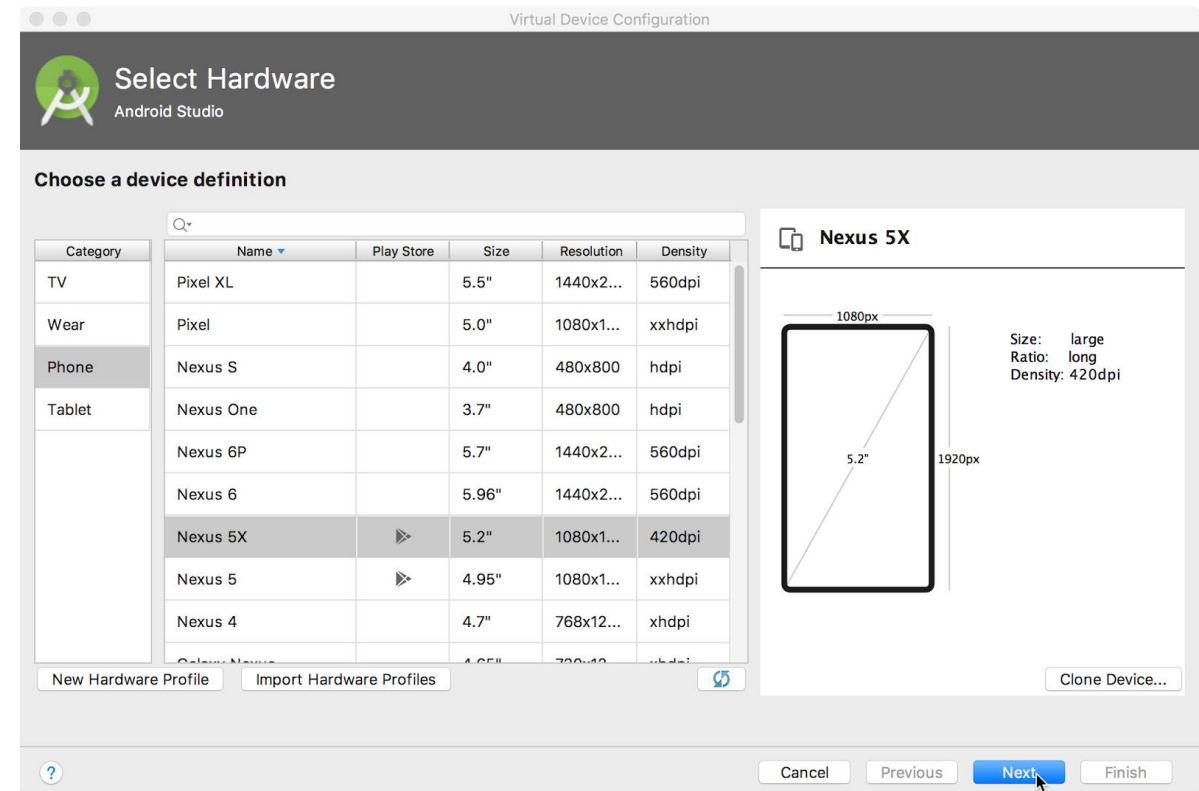


WEBFORCE
BE THE CHANGE

Etape 1 : Créer un périphérique virtuel Android (AVD)

Pour exécuter un émulateur sur votre ordinateur, vous devez créer une configuration décrivant le périphérique virtuel.

2. Cliquez sur le bouton **+Create Virtual Device**. La fenêtre **Select Hardware** apparaît avec une liste de périphériques matériels pré-configurés. Pour chaque périphérique, le tableau fournit une colonne pour sa taille d'affichage diagonale (**Size**), sa résolution d'écran en pixels (**Resolution**) et sa densité de pixels (**Density**) ;
3. Choisissez un appareil tel que **Nexus 5x** ou **Pixel XL**, puis cliquez sur **Next**.
L'écran Image système apparaît.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique virtuel (émulateur)

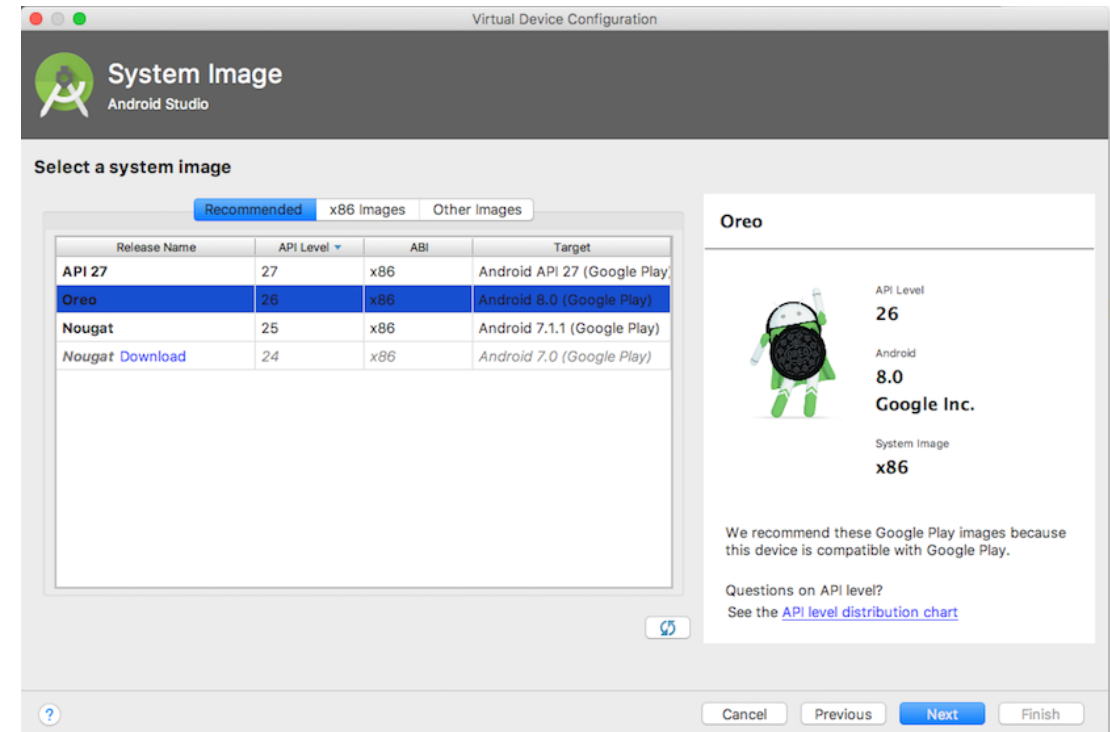


Etape 1 : Créer un périphérique virtuel Android (AVD)

4. Cliquez sur l'onglet **Recommended** s'il n'est pas déjà sélectionné et choisissez la version du système Android à exécuter sur le périphérique virtuel (telle que **Oreo**).

Il existe beaucoup plus de versions disponibles que celles indiquées dans l'onglet **Recommended**. Regardez les onglets **x86 images** et **Other Images** pour les voir.

Si un lien **Download** est visible à côté de l'image système que vous souhaitez utiliser, celle-ci n'est pas encore installée. Cliquez sur le lien pour lancer le téléchargement, puis cliquez sur **Finish** lorsque vous avez terminé.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique virtuel (émulateur)



WEBFORCE
BE THE CHANGE

Etape 1 : Créer un périphérique virtuel Android (AVD)

- Après avoir choisi une image système, cliquez sur **Next**. La fenêtre **Android Virtual Device (AVD)** apparaît. Vous pouvez également changer le nom de l'**AVD**. Vérifiez votre configuration et cliquez sur **Finish**.


ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

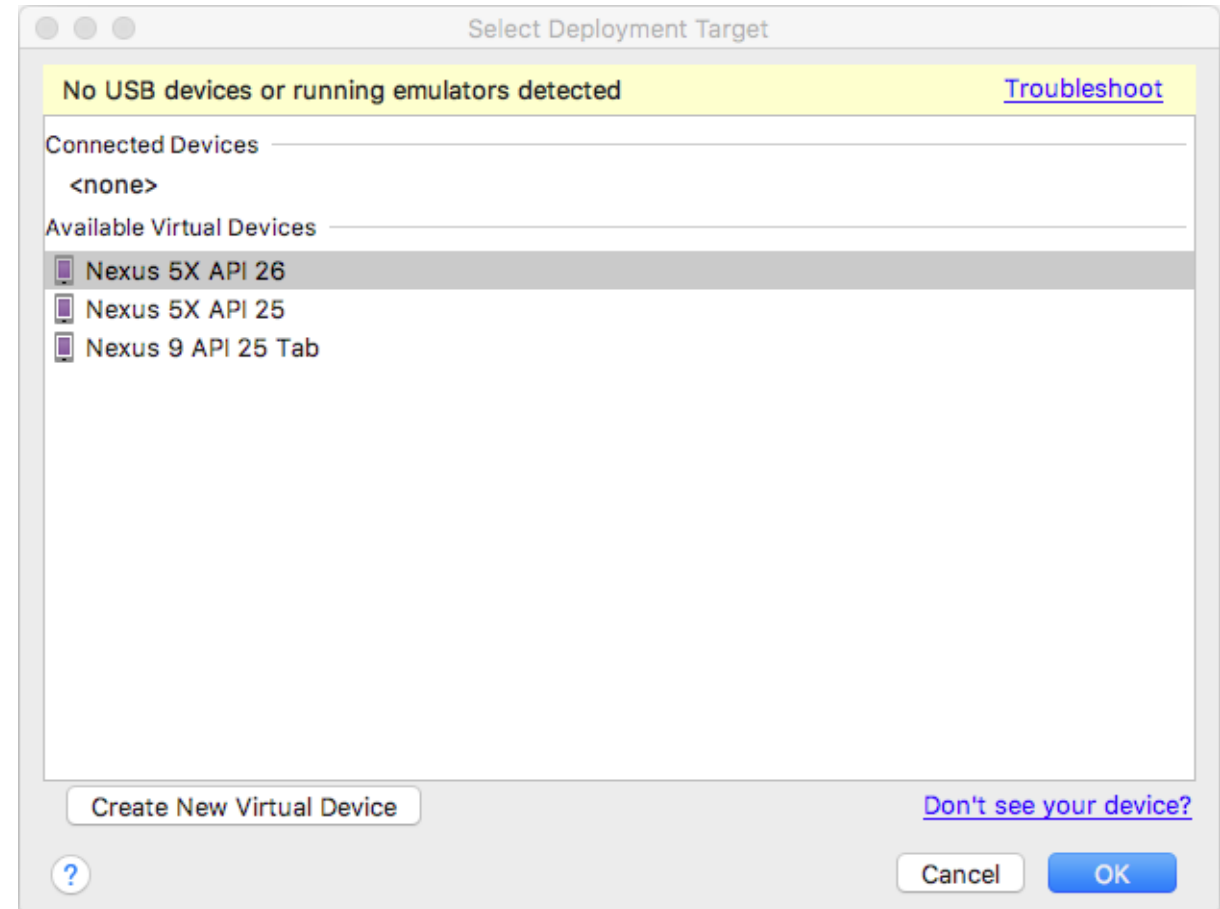
Utiliser un périphérique virtuel (émulateur)



Etape 2 : Exécuter l'application sur le périphérique virtuel

Dans cette tâche, vous allez enfin lancer votre application Hello World.

1. Dans **Android Studio**, choisissez **Run > Run app** ou cliquez sur l'icône **Run**  dans la barre d'outils ;
2. Dans la fenêtre **Select Deployment Target**, sous **Available Virtual Devices** (Périphériques virtuels disponibles), sélectionnez le périphérique virtuel que vous venez de créer, puis cliquez sur **OK**.



ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

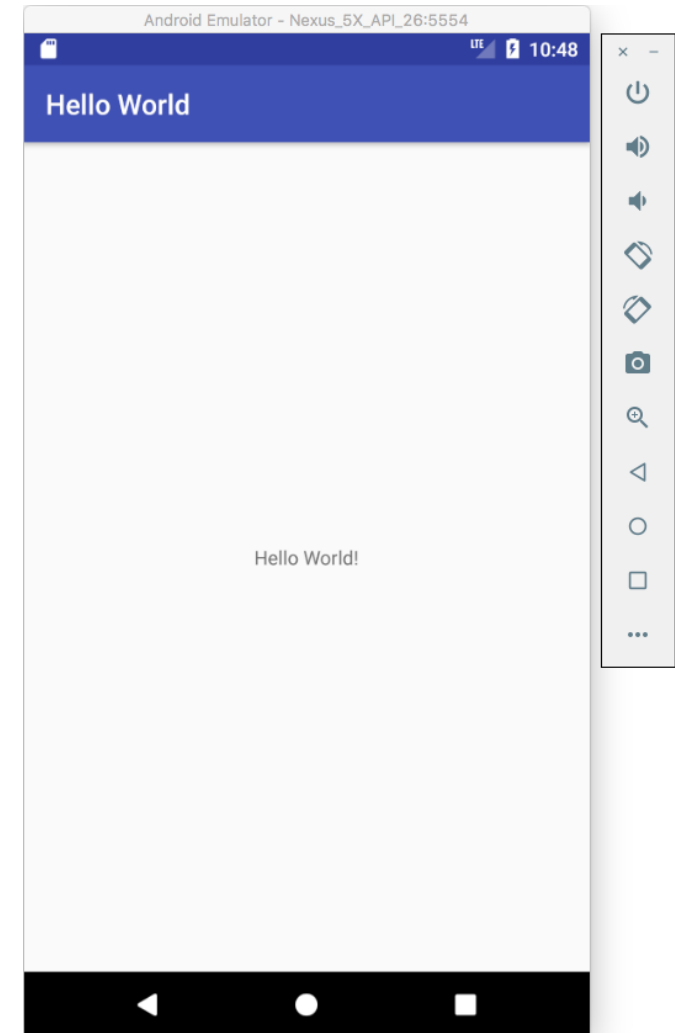
Utiliser un périphérique virtuel (émulateur)



Etape 2 : Exécuter l'application sur le périphérique virtuel

L'émulateur démarre et démarre comme un périphérique physique. Selon la vitesse de votre ordinateur, cela peut prendre un certain temps. Votre application est créée et une fois que l'émulateur est prêt, Android Studio la télécharge sur l'émulateur et l'exécute.

Vous devriez voir l'application Hello World comme indiqué dans la figure suivante.



Remarques

- Lors des tests sur un périphérique virtuel, il est recommandé de le démarrer une fois, au tout début de votre session. Vous ne devez pas la fermer avant d'avoir terminé de tester votre application, afin que celle-ci ne subisse plus le processus de démarrage de l'appareil. Pour fermer le périphérique virtuel, cliquez sur le bouton **X** en haut de l'émulateur, choisissez **Quit** dans le menu ou appuyez sur **Control-Q** sous Windows ou **Command-Q** sous macOS.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique physique



Objectif

Dans cette dernière tâche, vous exécuterez votre application sur un périphérique mobile physique tel qu'un téléphone ou une tablette. Vous devez toujours tester vos applications sur des périphériques virtuels et physiques.

Ce dont vous avez besoin :

- Un appareil Android tel qu'un téléphone ou une tablette ;
- Un câble de données pour connecter votre appareil Android à votre ordinateur via le port USB ;
- Si vous utilisez un système Linux ou Windows, vous devrez peut-être exécuter des étapes supplémentaires pour pouvoir s'exécuter sur un périphérique matériel.

Consultez [la documentation Utilisation de périphériques matériels](#). Vous devrez peut-être également installer le pilote USB approprié pour votre appareil. Pour les pilotes USB Windows, voir [OEM USB Drivers](#).

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique physique



WEBFORCE
BE THE CHANGE

Etape 1 : Activer le débogage USB

Pour permettre à Android Studio de communiquer avec votre appareil, vous devez **activer le débogage USB** sur votre **appareil Android**. Ceci est activé dans les paramètres **Developer options** de votre appareil.

Sur Android 4.2 et supérieur, l'écran **Developer options** est masqué par défaut. Pour afficher les options du développeur et activer le **USB Debugging**:

1. Sur votre appareil, ouvrez **Settings**, recherchez **About phone**, cliquez sur **About phone**, puis appuyez sur **Build number** sept fois ;
2. Retournez à l'écran précédent (**Settings / System**). **Developer options** apparaît dans la liste. Appuyez sur **Developer options** ;
3. Choisissez **USB Debugging**.


ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Utiliser un périphérique physique



Etape 2 : Exécuter votre application sur un appareil

Vous pouvez maintenant connecter votre appareil et exécuter l'application à partir d'Android Studio.

1. Connectez votre appareil à votre ordinateur de développement avec un câble USB ;
2. Cliquez sur le bouton **Run**  dans la barre d'outils. La fenêtre **Select Deployment Target** s'ouvre avec la liste des émulateurs disponibles et des périphériques connectés ;
3. Sélectionnez votre appareil et cliquez sur **OK**.

Android Studio installe et exécute l'application sur votre appareil.



Remarques

- Si votre Android Studio ne reconnaît pas votre appareil, essayez les solutions suivantes:
 1. Débranchez et rebranchez votre appareil;
 2. Redémarrez Android Studio.
- Si votre ordinateur ne trouve toujours pas le périphérique ou le déclare "unauthorized", procédez comme suit:
 1. Débrancher l'appareil ;
 2. Sur l'appareil, ouvrez **Developer Options** dans l'application **Settings** ;
 3. Appuyez sur **Revoke USB Debugging authorizations** ;
 4. Reconnectez le périphérique à votre ordinateur ;
 5. Lorsque vous y êtes invité, accordez des autorisations.
- Vous devrez peut-être installer le pilote USB approprié pour votre périphérique. Consultez [la documentation Utilisation de périphériques matériels](#).

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Modifier la configuration Gradle de l'application



Etape 1 : Changer la version minimale du SDK pour l'application

Dans cette tâche, vous allez modifier quelque chose à propos de la configuration de l'application dans le fichier **build.gradle(Module:app)** afin d'apprendre à modifier et à les synchroniser avec votre projet Android Studio.

Suivez ces étapes:

1. Développez le dossier **Gradle Scripts** s'il n'est pas déjà ouvert et double-cliquez sur le fichier **build.gradle(Module:app)** ;

Le contenu du fichier apparaît dans l'éditeur de code.

2. Au sein du bloc **DefaultConfig**, modifiez la valeur de **minSdkVersion** à 28 comme indiqué ci-dessous (il a été initialement fixé à 21) :

```
> app
  ▾ Gradle Scripts
    build.gradle (Project: Hello_World)
    build.gradle (Module: Hello_World.app)
    gradle-wrapper.properties (Gradle Ver
    proguard-rules.pro (ProGuard Rules fo
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)

8
9
10
11
12
13

You can use the Project Structure dialog to view and edit your project configuration Open (Ctrl+)

defaultConfig {
    applicationId "ma.projet.android.helloworld"
    minSdk 28
    targetSdk 31
    versionCode 1
    versionName "1.0"
```

3. L'éditeur de code affiche une barre de notification en haut avec le lien **Sync Now**.


ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Modifier la configuration Gradle de l'application



Etape 2 : Synchroniser la nouvelle configuration Gradle

Lorsque vous apportez des modifications aux fichiers de configuration de la construction dans un projet, Android Studio exige que vous synchronisez les fichiers du projet afin qu'il puisse importer les modifications et effectuer des vérifications pour vous assurer que la configuration ne créera pas d'erreurs de construction.

Pour synchroniser les fichiers du projet, cliquez sur **Sync Now** dans la barre de notification qui apparaît lorsque vous effectuez une modification (comme illustré dans la figure précédente) ou cliquez sur l'icône **Sync Project with Gradle Files**  dans la barre d'outils.

Lorsque la synchronisation de Gradle est terminée, le message Gradle build finished apparaît dans le coin inférieur gauche de la fenêtre d'Android Studio.

Pour en savoir plus sur Gradle, consultez la documentation sur [Build System Overview](#) et [Configuring Gradle Builds](#).

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Ajouter des instructions de journal (log) à votre application



Objectif

Dans cette étape, vous allez ajouter des instructions de journal (Log) à votre application, qui affichent des messages dans le volet Logcat. Les messages du journal (Log) sont un puissant outil de débogage que vous pouvez utiliser pour vérifier les valeurs, les chemins d'exécution et signaler les exceptions.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

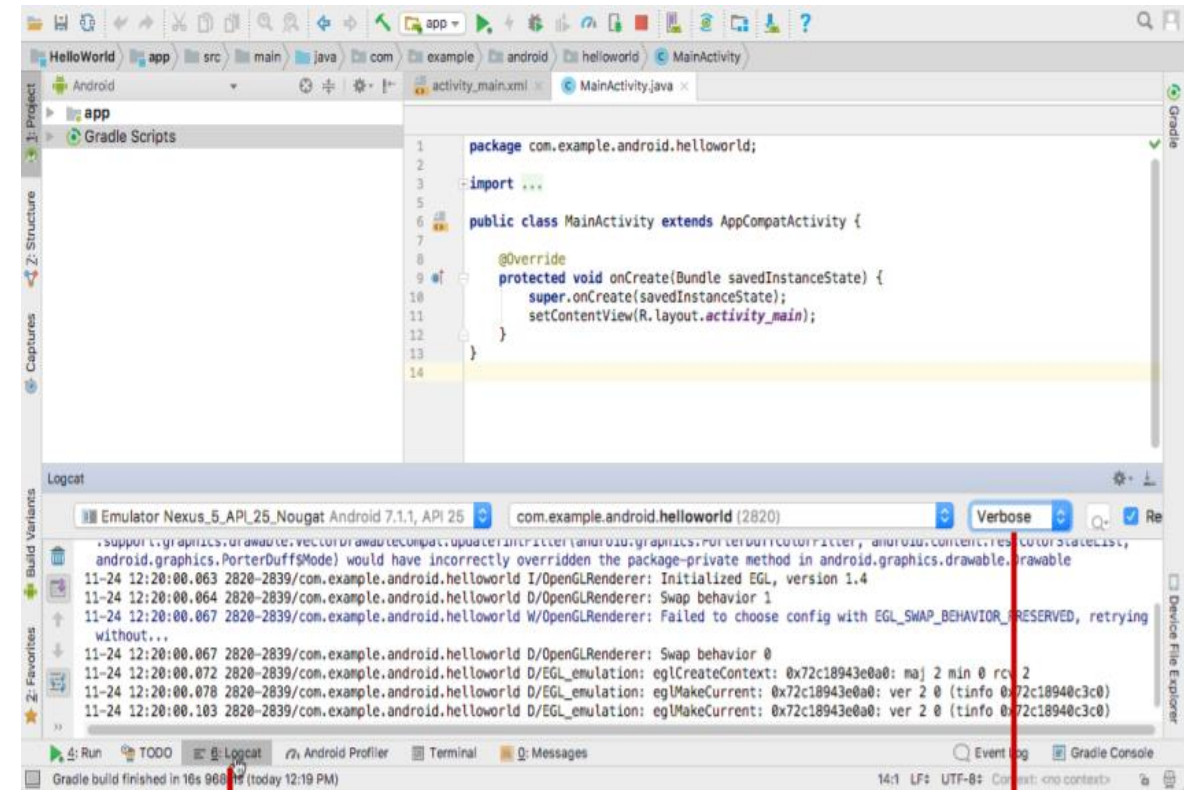
Ajouter des instructions de journal (log) à votre application

Etape 1 : Afficher le volet Logcat

Pour afficher le volet **Logcat**, cliquez sur l'onglet **Logcat** au bas de la fenêtre d'Android Studio, comme indiqué dans la figure ci-dessous.

Dans la figure ci-après :

1. L'onglet **Logcat** pour ouvrir et fermer le volet **Logcat**, qui affiche des informations sur votre application en cours d'exécution. Si vous ajoutez des instructions de journal (Log) à votre application, les messages de journal (Log) apparaissent ici ;
2. Le menu de niveau de Log est défini sur **Verbose** (valeur par défaut), qui affiche tous les messages du journal (Log). Les autres paramètres incluent **Debug**, **Error**, **Info** et **Warn**.



1

2

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Ajouter des instructions de journal (log) à votre application



Etape 2 : Ajoutez les instructions du journal (log) à votre application

Les instructions de journal (Log) dans votre code d'application affichent des messages dans le volet **Logcat**. Par exemple :

```
Log.d("MainActivity", "Hello World");
```

Les parties du message sont :

- **Log** : classe de journalisation (Log) pour l'envoi de messages de journalisation au volet Logcat ;
- **D** : le paramètre de niveau du journal de débogage (**Debug**) permettant de filtrer les messages du journal s'affiche dans le volet Logcat. Les autres niveaux de journalisation sont e pour **Error**, w pour **Warn** et i pour **Info** ;
- **"MainActivity"**: le premier argument est une balise qui peut être utilisée pour filtrer les messages dans le volet Logcat. C'est généralement le nom de l'activité (Activity) à l'origine du message. Cependant, vous pouvez faire tout ce qui vous est utile pour le débogage

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Ajouter des instructions de journal (log) à votre application



Etape 2 : Ajoutez les instructions du journal (log) à votre application

Par convention, les balises de journal sont définies en tant que constantes pour l'activité (Activity):

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();
```

- "Hello world": Le deuxième argument est le message réel.

Suivez ces étapes:

1. Ouvrez votre application Hello World dans Android Studio et ouvrez MainActivity ;
2. Pour ajouter automatiquement des importations non ambiguës à votre projet (par exemple, android.util.Log requis pour utiliser Log), choisissez **File > Settings** dans Windows ou **Android Studio > Preferences** dans macOS ;
3. Choisissez **Editor > General > Auto Import**. Cochez toutes les cases et définissez **Insert imports on paste** sur **All** ;
4. Cliquez sur Apply, puis sur **OK**.

ACTIVITÉ n° 1 – Lancer une application mobile sur smartphone

Ajouter des instructions de journal (log) à votre application



Etape 2 : Ajoutez les instructions du journal (log) à votre application

5. Dans la méthode `onCreate()` de `MainActivity`, ajoutez l'instruction suivante :

```
Log.d("MainActivity", "Hello World");
```

La méthode `onCreate()` devrait maintenant ressembler au code suivant :

```
@Override protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    Log.d("MainActivity", "Hello World");  
}
```

6. Si le volet **Logcat** n'est pas déjà ouvert, cliquez sur l'onglet **Logcat** au bas d'Android Studio pour l'ouvrir ;
7. Vérifiez que le nom de la cible et le nom du package de l'application sont corrects ;
8. Changez le niveau Log dans le volet **Logcat** sur **Debug** (ou laissez-le sur **Verbose** car il y a si peu de messages de journal) ;
9. Exécutez votre application.

Le message suivant devrait apparaître dans le volet Logcat.

```
07-24 15:06:39.001 4696-4696/? D/MainActivity: Hello World
```




ACTIVITÉ n° 2

Configurer les flavors et installer plusieurs applications sur smartphone

Compétences visées :

- Se servir de variantes de compilation
- Se servir des ensembles de sources
- Créer différentes versions de votre application à l'aide de variantes de compilation et d'ensembles de sources
- Attribuer des ID d'application uniques à vos variantes d'application

Recommandations clés :

- Révision générale du résumé théorique
- Suivre les instructions du TP et organiser le dossier de travail



8 heures

CONSIGNES

1. Pour le formateur :

- Rappeler le système de compilation Android
- Demander aux apprenants de suivre les étapes décrites dans l'activité

2. Pour l'apprenant :

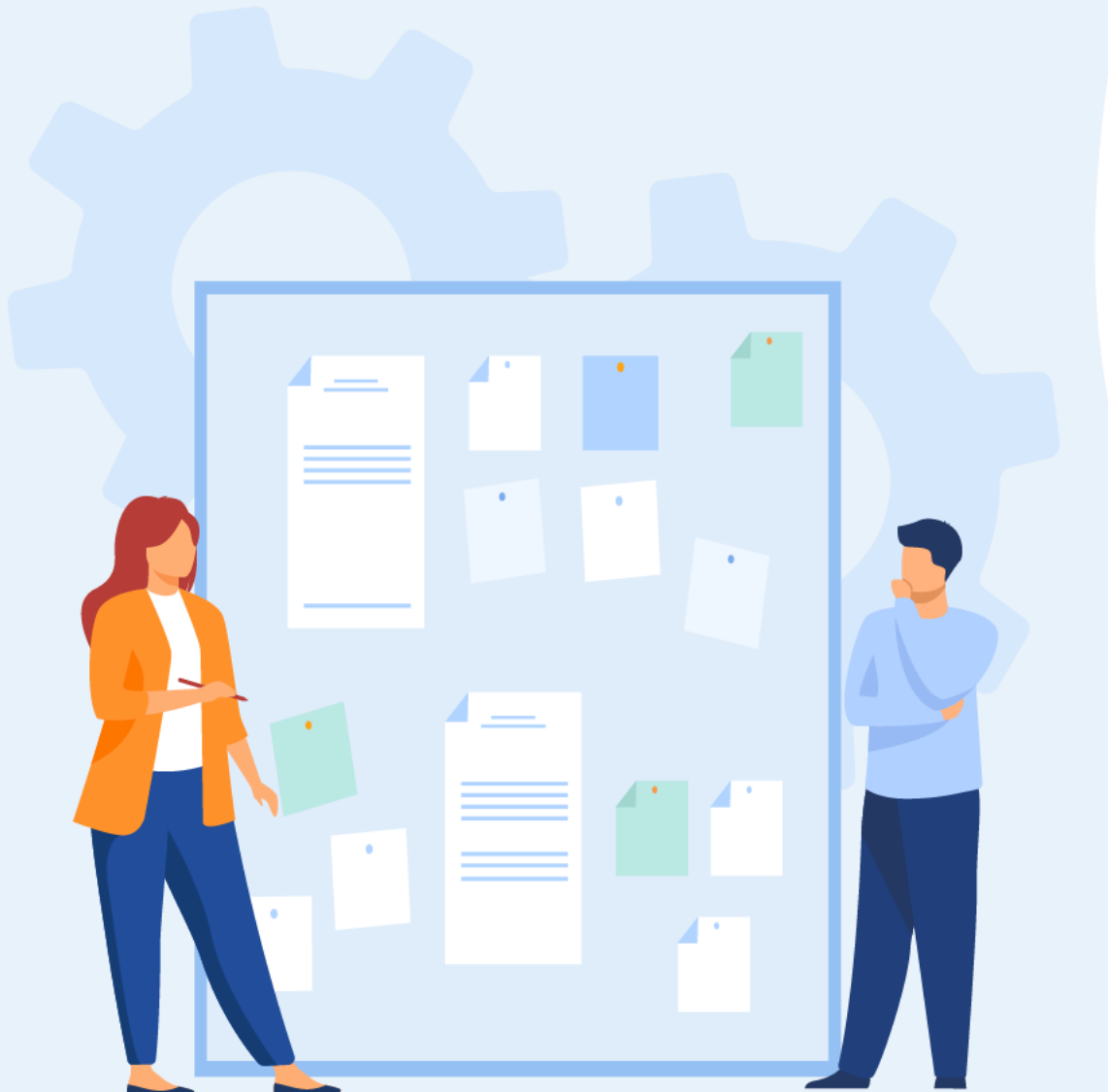
- Suivre les étapes indiquées dans l'activité

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Un ordinateur sur lequel est installée la dernière version stable d'Android Studio

4. Critères de réussite :

- Le stagiaire doit être capable de :
 - créer des types de produits et des ensembles de sources
 - personnaliser le titre de l'application selon la version
 - attribuer des ID d'application uniques à chaque version
 - générer l'APK de chaque version de l'application



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Configurer votre environnement



Utiliser la vue Projet

- Nous allons continuer sur le projet **Hello world**.
- Lorsque vous utilisez des variantes de compilation, vous devez utiliser vos fichiers de projet dans la vue **Projet** pour afficher tous les répertoires des différentes variantes. Pour ce faire, ouvrez le volet **Project** (Projet) dans Android Studio, cliquez sur le menu "View Type" (Type de vue) (défini sur **Android** par défaut), puis sélectionnez **Project** (Projet).

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



- Les variantes de compilation sont le résultat de différentes combinaisons de types de produit et de types de compilation. Vous pouvez considérer les types de produit comme des attributs côté utilisateur, et les types de compilation comme des attributs côté développeur. Vous n'allez pas configurer les variantes de compilation directement : vous allez définir un ensemble de types de produit et un ensemble de types de compilation, ce qui déterminera les variantes de compilation;
- Les variantes de compilation représentent les différentes combinaisons "type de produit/type de compilation" possibles. Elles sont donc nommées selon le schéma <product-flavor><build-type>. Par exemple, avec les types de compilation debug et release, et les types de produit demo et full, vous obtiendrez les variantes de compilation suivantes :
 - demoDebug ;
 - demoRelease ;
 - fullDebug ;
 - fullRelease.
- Configurons à présent les types de produit et les types de compilation pour l'application **HelloWorld**.

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 1 : Configurer des types de produit

Les types de produit correspondent aux attributs de votre application côté utilisateur, dans la mesure où ils représentent généralement les versions de l'application disponibles. Pour créer la version de démo et la version complète de votre application, vous devez ajouter les deux types de produit correspondants et les attribuer à un groupe de types. Pour ajouter les types de produit, ouvrez le fichier build.gradle au niveau de l'application (app > build.gradle dans la vue Project) et collez ce code dans le bloc android {}.

```
flavorDimensions "app_type"
productFlavors {
    demo {
        dimension "app_type"
    }
    full {
        dimension "app_type"
    }
}
```

Ce code effectue les opérations suivantes :

- Il crée un groupe de types appelé app_type ;
- Il crée deux types de produit, représentés par les blocs demo {} et full {} ;
- Il attribue les deux types de produit au groupe app_type (facultatif, s'il n'y a qu'un seul groupe de types).

Il s'agit du code de base nécessaire pour définir les types de produit. Vous utiliserez d'autres options dans la suite de cet atelier de programmation.

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapas 2 : Configurer des types de compilation

- Les types de compilation sont des attributs côté développeur, dans le sens où ils représentent généralement des étapes de développement (débogage, version bêta et version de production, par exemple). Android Studio configure automatiquement deux types de compilation pour vous : debug et release. Le type de compilation debug est destiné au débogage, tandis que le type de compilation release est destiné à la distribution.
- Vous pouvez créer des types de compilation et modifier leurs configurations en modifiant le bloc `buildTypes {}` dans le fichier `build.gradle` au niveau de l'application. Le bloc `buildTypes {}` par défaut se présente comme suit.

```
buildTypes {  
    release {  
        minifyEnabled false  
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
    }  
}
```

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 2 : Configurer des types de compilation

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}
```

Vous pouvez voir la configuration par défaut pour le type de compilation release (nous n'aborderons pas les propriétés `minifyEnabled` ni `proguardFiles` dans cet atelier de programmation). Le type de compilation debug n'apparaît pas par défaut dans le fichier de configuration de compilation, mais vous pouvez ajouter un bloc `debug {}` pour ajouter ou modifier certains paramètres. La configuration par défaut fonctionne pour nos besoins. Vous pouvez donc la laisser telle quelle.


ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Étapes 3 : Utiliser l'outil Build Variants

Maintenant que vous disposez de deux types de produit et de deux types de compilation, voyons quelles variantes de compilation ils permettent de créer. Pour afficher vos nouvelles variantes de compilation dans Android Studio, procédez comme suit :

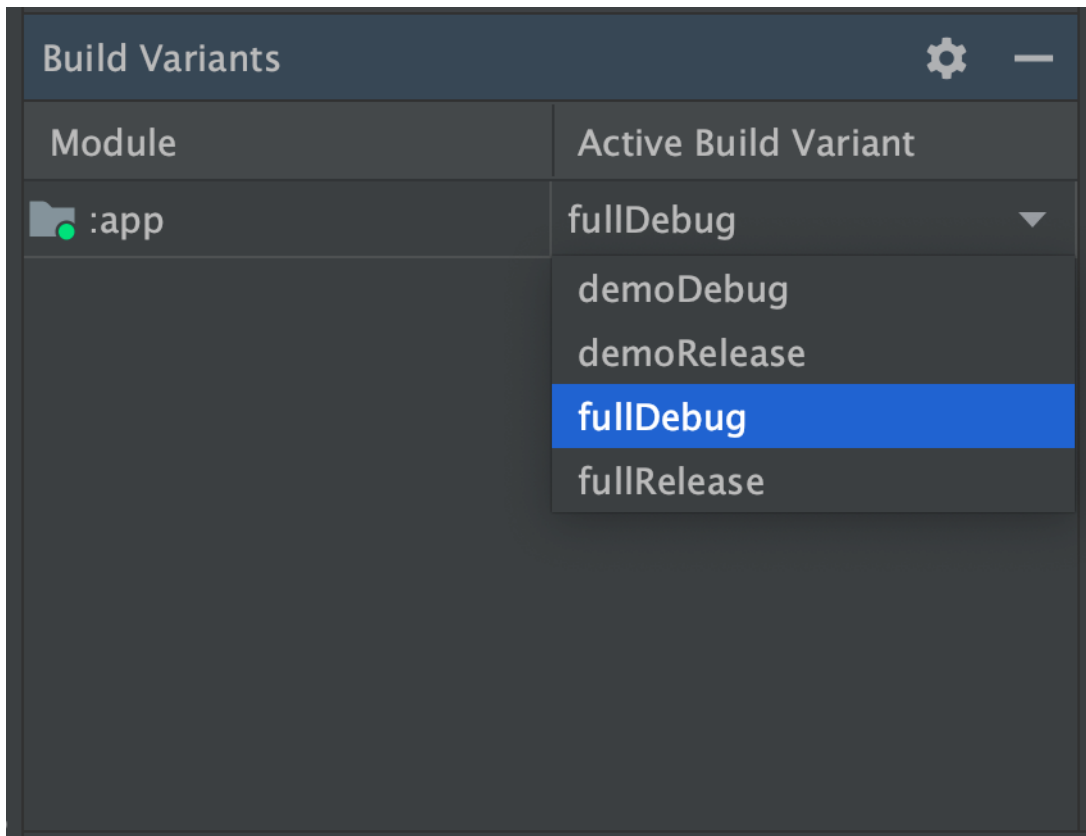
1. Dans la barre d'outils, cliquez sur **Sync Project with Gradle Files**  (Synchroniser le projet avec les fichiers Gradle). Chaque fois que vous apportez des modifications pour créer des fichiers de configuration de compilation, Studio vous invite à synchroniser vos fichiers afin d'enregistrer la nouvelle configuration de compilation et de rechercher les erreurs de compilation.
2. Cliquez sur **Build > Select Build Variant** (Compilation > Sélectionner une variante de compilation) ou sur **View > Tool Windows > Build Variants** (Vue > Fenêtres d'outil > Variantes de compilation) pour afficher la fenêtre **Build Variants** (Variantes de compilation).
3. Cliquez sur **demoDebug** dans la colonne **Active Build Variant** (Variante de compilation active) pour ouvrir un menu contenant toutes les variantes de compilation : **demoDebug**, **demoRelease**, **fullDebug**, et **fullRelease**. Ce menu vous permet de sélectionner différentes variantes de compilation à exécuter et tester.

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 3 : Utiliser l'outil Build Variants



Exécutez les variantes demoDebug et fullDebug (les variantes basées sur le type de compilation release nécessitent une configuration supplémentaire pour être exécutées ; nous ne les utiliserons donc pas dans cet atelier de programmation).

Pour l'instant, les variantes demoDebug et fullDebug ne sont pas différenciables du point de vue de l'utilisateur.

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



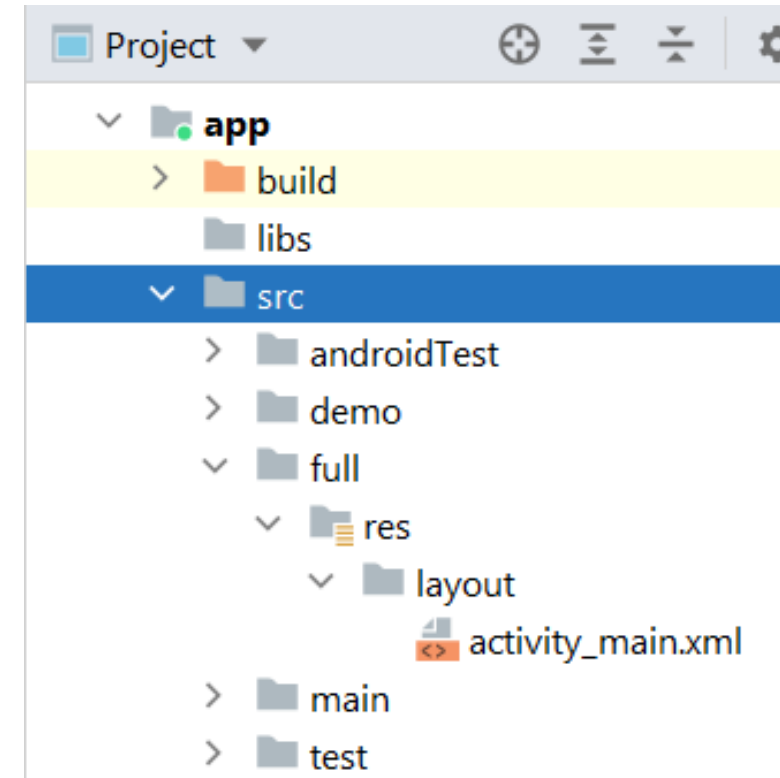
Etapes 4 : Créer des fonctionnalités différentes pour les différentes variantes

Commencez par créer l'ensemble de sources full et le répertoire de ressources Java :

1. Ouvrez le volet Project (Projet) et sélectionnez la vue Project (Projet) ;
2. Accédez au répertoire projet/app/src/ ;
3. Faites un clic droit sur le répertoire src, puis sélectionnez New > Directory (Nouveau > Répertoire) ;
4. Dans le menu situé sous Gradle Source Sets (Ensembles de sources Gradle), sélectionnez full/ressources, puis appuyez sur Enter.

Ensuite, collez le fichier activity_main.xml de l'ensemble de sources main dans l'ensemble de sources full.

1. Accédez à projet/app/src/main/res/layout ;
2. Faites un clic droit sur le fichier activity_main.xml, puis cliquez sur Copy (Copier) ;
3. Accédez à projet/app/src/full/res/layout/ ;
4. Faites un clic droit sur le répertoire layout, puis cliquez sur Paste (Coller).



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 4 : Créer des fonctionnalités différentes pour les différentes variantes

Ensuite, modifier le code de activity_main.xml de l'ensemble de sources full :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="La version payante"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



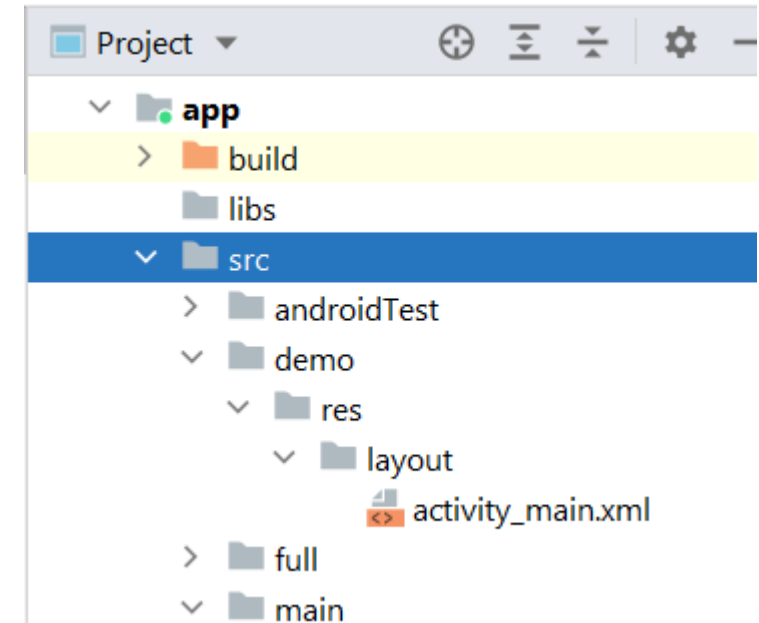
Etapes 4 : Créer des fonctionnalités différentes pour les différentes variantes

Commencez par créer l'ensemble de sources demo et le répertoire de ressources Java :

1. Ouvrez le volet Project (Projet) et sélectionnez la vue Project (Projet) ;
2. Accédez au répertoire projet/app/src/ ;
3. Faites un clic droit sur le répertoire src, puis sélectionnez New > Directory (Nouveau > Répertoire), et tapez demo/res/layout.

Ensuite, collez le fichier activity_main.xml de l'ensemble de sources main dans l'ensemble de sources demo.

1. Accédez à projet/app/src/main/res/layout ;
2. Faites un clic droit sur le fichier activity_main.xml, puis cliquez sur Copy (Copier) ;
3. Accédez à projet/app/src/full/res/layout/ ;
4. Faites un clic droit sur le répertoire layout, puis cliquez sur Paste (Coller).



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



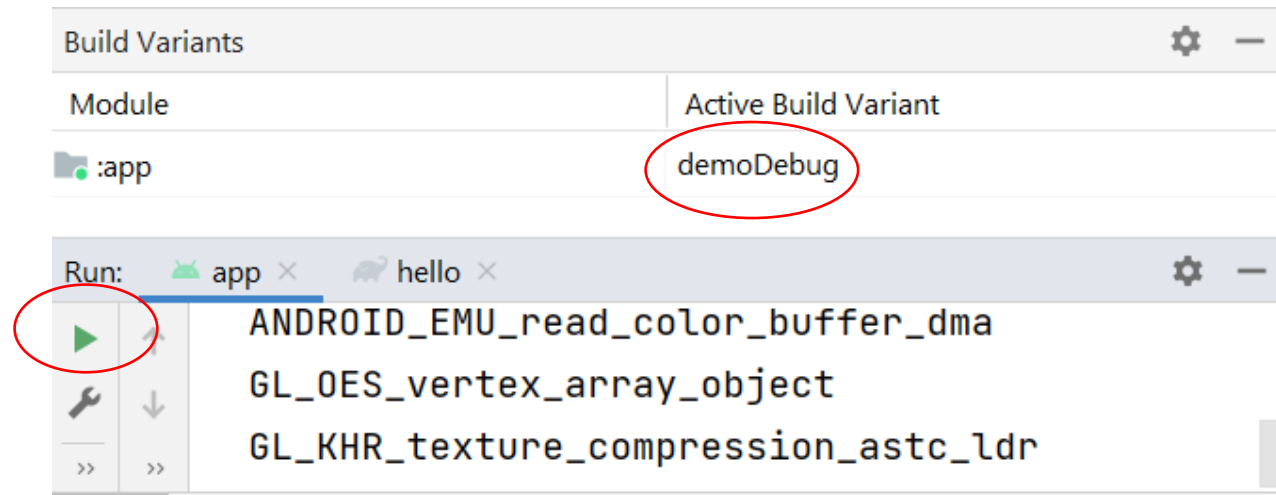
Etapes 4 : Créer des fonctionnalités différentes pour les différentes variantes

Ensuite, modifiez le code de activity_main.xml de l'ensemble de sources demo :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="La version gratuite" ←
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 4 : Créer des fonctionnalités différentes pour les différentes variantes

Exécutez les variantes de compilation **fullDebug** et **demoDebug** de votre application : sélectionnez la variante que vous souhaitez exécuter via la fenêtre de l'outil Build Variants, puis cliquez sur Run (Exécuter). La nouvelle zone ne doit s'afficher que pour la variante fullDebug. Elle ne fait rien pour le moment. Vous allez donc la programmer pour que le texte change en fonction du lancer.



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 5 : Modifier le titre des différentes versions de l'application

Par souci de clarté, vous allez spécifier la version de l'application utilisée dans le titre de celle-ci. Ce type de modification peut être effectué sans les ensembles de sources. Le titre de l'application est défini par la propriété label dans le fichier AndroidManifest.xml (**app > manifests > AndroidManifest.xml**). Pour obtenir la valeur label qui change en fonction de la variante en cours d'exécution, ouvrez le fichier AndroidManifest.xml et modifiez la ligne de libellé comme suit :

```
android:label="${appLabel}"
```

Cette action attribue une variable, appLabel, au titre de l'application.

Ensuite, pour définir la valeur de appLabel ou modifier le titre de la version de démonstration de l'application, ajoutez la ligne manifestPlaceholders au bloc demo {} que vous avez créé précédemment :

```
demo {  
    dimension "version"  
    manifestPlaceholders = [appLabel: "Version - Gratuite"]  
    applicationIdSuffix ".demo"  
}
```

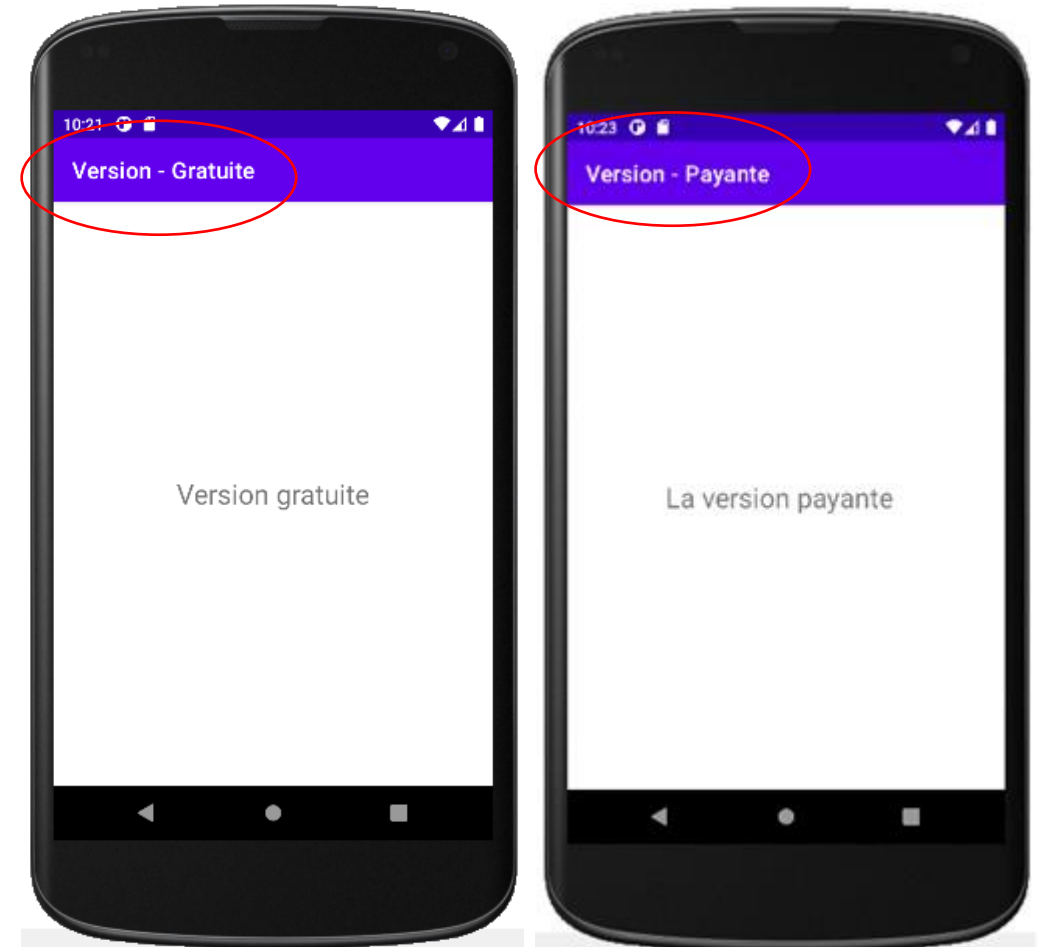
```
full {  
    dimension "version"  
    manifestPlaceholders = [appLabel: "Version - Payante"]  
    applicationIdSuffix ".full"  
}
```

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation

Étapes 5 : Modifier le titre des différentes versions de l'application

Exécutez à nouveau les variantes demoDebug et fullDebug. Les titres des différentes versions doivent s'afficher.



ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 6 : Attribuer des ID d'application uniques à vos variantes de compilation

Lorsque vous compilez un APK ou AAB pour votre application, les outils de compilation attribuent à l'application l'ID d'application défini dans le bloc `defaultConfig {}` à partir du fichier `build.gradle` au niveau de l'application (comme illustré à droite). Toutefois, si vous souhaitez créer différentes versions de votre application sous forme de fiches distinctes sur le Google Play Store, comme une version "démon" ou "complète", vous devez attribuer un ID d'application différent à chaque version. Pour chaque type de produit du bloc `productFlavors {}`, vous pouvez redéfinir la propriété `applicationId` ou ajouter un segment à l'ID d'application par défaut à l'aide de `applicationIdSuffix`, comme indiqué à droite :

```
defaultConfig {
    applicationId "com.projet.hello"
    ...
}

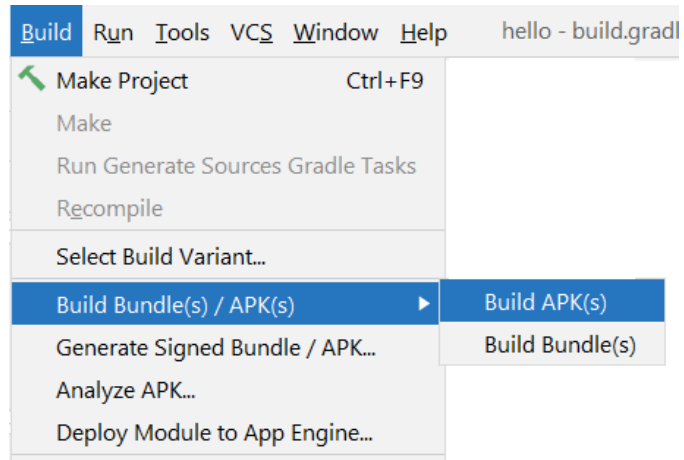
flavorDimensions "version"
productFlavors {
    demo {
        dimension "version"
        manifestPlaceholders = [appLabel: "Dice Roller - Demo"]
        applicationIdSuffix ".demo"
    }
    full {
        dimension "version"
        manifestPlaceholders = [appLabel: "Dice Roller - Full"]
        applicationIdSuffix ".full"
    }
}
```

ACTIVITÉ n° 2 – Configurer les flavors et installer plusieurs applications sur smartphone

Variantes de compilation



Etapes 7 : Générer l'APK pour chaque version



Nom	Modifié le	Type	Taille
demo	04/08/2022 19:20	Dossier de fichiers	
full	04/08/2022 20:48	Dossier de fichiers	

Nom	Modifié le	Type	Taille
app-demo-debug	04/08/2022 23:21	BlueStacks Androi...	3 642 Ko
output-metadata	04/08/2022 23:21	Fichier source JSON	1 Ko

Nom	Modifié le	Type	Taille
app-full-debug	04/08/2022 23:39	BlueStacks Androi...	3 803 Ko
output-metadata	04/08/2022 23:39	Fichier source JSON	1 Ko



WEBFORCE
BE THE CHANGE



PARTIE 3

Coder en Java

Dans ce module, vous allez :

- Vous initier au langage JAVA
- Réaliser des interfaces graphiques simples
- Maîtriser Git



46 heures



ACTIVITÉ n° 1

Créer des applications JAVA en appliquant les principes de la POO

Compétences visées :

- Manipuler des boucles, expressions de lectures et d'écritures
- Manipuler des tableaux à une seule dimension et à deux dimensions
- Créer et manipuler des classes et objets
- Implémenter correctement l'évolution des objets dans le temps (héritage, classes abstraites)
- Maîtriser le concept du polymorphisme et les interfaces
- Manipuler des exceptions, des collections et les entrées sorties

Recommandations clés :

- Il faut consacrer 2h à 3h de pratique pour chaque compétence visée



28 heures

CONSIGNES

1. Pour le formateur :

- Pour les deux premières compétences visées l'apprenant ne doit pas utiliser un IDE, doit travailler avec les commandes de base (java et javac)
- Pour le reste des compétences, l'apprenant doit utiliser Android studio, et l'affichage des résultats va se faire au niveau de la console
- Rappeler les différents concepts de la POO

2. Pour l'apprenant :

- Utiliser le résumé théorique
- Suivre les étapes qui figurent dans TPs
- Demander l'aide du formateur en cas de blocage sérieux

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Installation de JDK pour les deux premières compétences
- Installation d'Android studio pour le reste des compétences

4. Critères de réussite :

- Le stagiaire doit être capable de :
 - Manipuler les concepts de la POO en JAVA
 - Mettre en place les mécanismes de la gestion des exceptions
 - Exploiter les collections et la généricité
 - Manipuler les fichiers



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Installation de JDK

Cette tâche présente un aperçu de l'installation du JDK. Le kit de développement Java (JDK) est utile pour écrire des programmes en Java. Il est également connu sous le nom de Java Platform Standard Edition (Java SE). Le JDK est un logiciel libre fourni par Sun Microsystems, aujourd'hui racheté par Oracle, que chacun peut utiliser librement pour la programmation.

De nombreuses personnes se posent la question : c'est quoi la différence entre JDK ou JRE ?

Voyons donc à quoi sert chacun d'eux. L'environnement d'exécution Java (JRE) est nécessaire pour **exécuter** les programmes Java, tandis que le JDK est nécessaire pour **écrire** et **exécuter** les programmes. Le JDK est un kit de développement contenant le JRE et les outils de développement (débugueur et compilateur) nécessaires à l'édition d'un programme en Java. En termes simples, JRE est un sous-ensemble de JDK, c'est-à-dire qu'il est inclus dans JDK. Par conséquent, nous devons installer JDK pour écrire et exécuter des programmes.

Conditions préalables à l'installation de JDK :

JDK a des exigences minimales en matière d'espace disque et de RAM pour la plate-forme Windows 64 bits. Il faut environ 800 Mo d'espace disque pour installer JDK, car JRE est également installé avec lui. Le JDK nécessite 128 Mo d'espace mémoire pour fonctionner correctement. Il s'agit de la RAM minimale requise pour l'exécution de programmes de base et de petits programmes, mais plus la taille d'une application augmente, plus les besoins en mémoire augmentent pour que l'application fonctionne correctement.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Installation pas à pas de JDK

- Étape 1 : Télécharger JDK depuis le site d'Oracle

Allez sur le site d'Oracle et ouvrez la page de téléchargement de Java SE. Sous la dernière version de Java Platform, Standard Edition, cliquez sur le bouton de téléchargement du JDK.

Ensuite, cliquez sur le bouton Accepter le contrat de licence et choisissez votre version de Java pour Windows (32 bits ou 64 bits) pour procéder au téléchargement du fichier exécutable JDK.

Java SE Downloads



Java Platform (JDK) 12

Java Platform, Standard Edition

Java SE 12.0.2

Java SE 12.0.2 is the latest release for the Java SE Platform

[Learn more](#)

- [Installation Instructions](#)
- [Release Notes](#)
- [Oracle JDK License](#)
- [Java SE Licensing Information User Manual](#)
 - [Includes Third Party Licenses](#)
- [Certified System Configurations](#)
- [Readme](#)

Oracle JDK
DOWNLOAD

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Installation pas à pas de JDK

- **Étape 2 : Installer le fichier exe du JDK**
 - Dans cette étape, nous allons exécuter le fichier exécutable JDK (il s'agit d'un fichier portant l'extension .exe). Une fois le téléchargement terminé, cela installe le JDK ainsi que le JRE. Pour exécuter ce fichier sous Windows, nous aurons besoin des droits d'administrateur ;
 - Pour commencer l'installation, nous devons double-cliquer sur le fichier téléchargé, et la fenêtre à droite s'affichera ;
 - Pour récupérer une partie de l'espace disque de notre système, il est bon de supprimer le fichier exe une fois le téléchargement terminé.



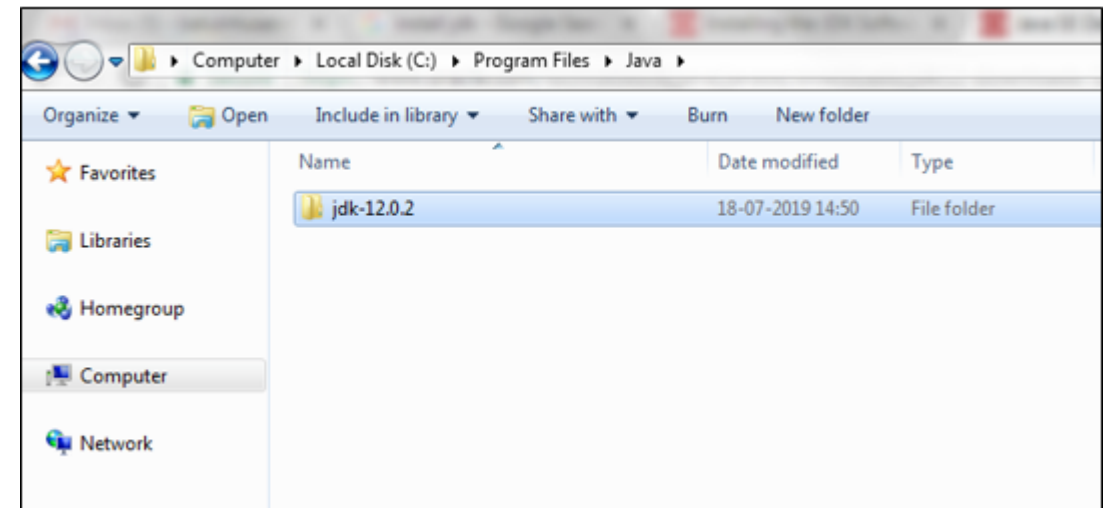
ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Installation pas à pas de JDK

- Étape 3 : Vérifiez le répertoire
- JDK est installé par défaut dans le répertoire C de notre système, avec le chemin "**C:\Program Files\Java\jdk-12.0.2**". Si nous modifions ce chemin, nous devons le noter car il sera nécessaire dans les étapes suivantes.
- Voici la structure du répertoire pour notre exemple.



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



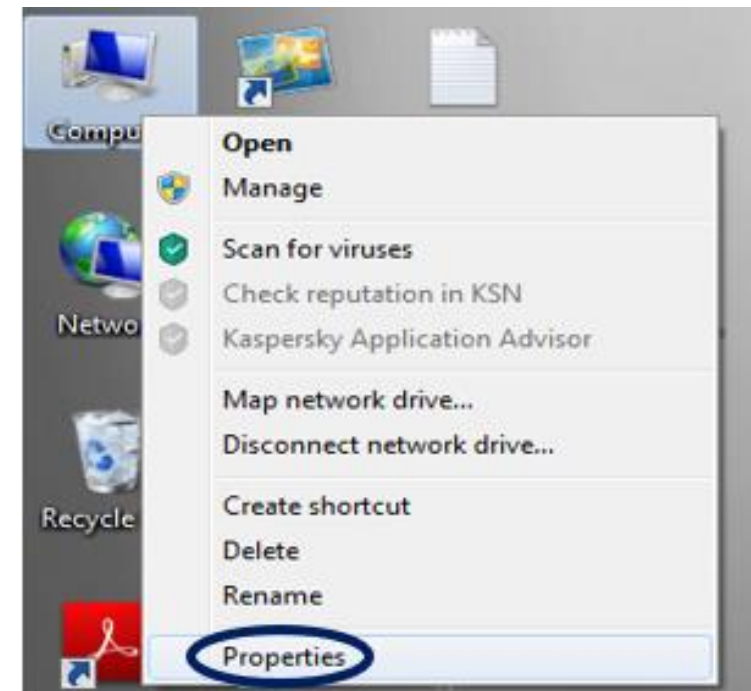
Installation pas à pas de JDK

- **Étape 4 : Mettre à jour les variables d'environnement**
- Nous devons mettre à jour les variables d'environnement de notre système avec le chemin du binaire JDK installé pour exécuter les programmes Java, car lors de l'exécution des programmes, l'invite de commande recherchera le chemin complet du binaire JDK ;
- La variable **PATH** de notre système fournit l'emplacement exact des exécutables qui seront utilisés pour exécuter les programmes Java, tels que **javac** et **java**. La variable CLASSPATH nous fournit l'emplacement des fichiers de bibliothèque ;
- Si nous ne définissons pas la variable PATH, nous spécifierons le chemin complet du bin JDK chaque fois que nous exécuterons un programme ;

Par exemple :

```
C:\> "C:\Program Files\Java\jdk-12.0.2\bin\javac" TestClass.java
```

- Pour définir ces variables, cliquez d'abord avec le bouton droit de la souris sur Mon PC et sélectionnez Propriétés.



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

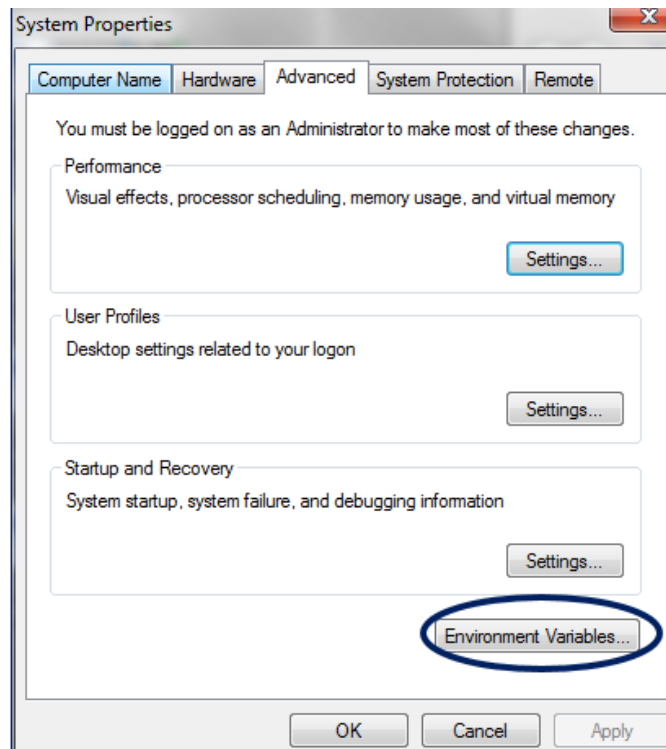
Bases du langage JAVA



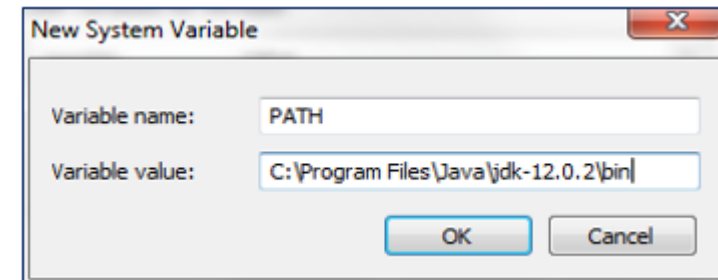
Installation pas à pas de JDK

- **Étape 4 : Mettre à jour les variables d'environnement**

Dans Propriétés, dans le panneau de gauche, sélectionnez Paramètres système avancés, puis choisissez l'option Variables d'environnement.



Cliquez sur Nouveau, tapez PATH dans le Nom de la variable, et entrez le chemin d'accès au bin du JDK installé dans le champ Valeur de la variable.



Si nous avons déjà la variable PATH, nous pouvons la modifier en l'ajoutant aux valeurs existantes.

Cliquez sur le bouton OK pour appliquer les modifications.

- **Étape 5 : Vérifier l'installation de Java**

Ouvrez l'invite de commande et entrez la commande "java -version", et si elle s'exécute avec succès, Java a été installé avec succès.

Maintenant que nous avons vu les étapes d'installation de JDK, commençons à programmer.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Exercices

Exercice 1 : Série harmonique

1. Écrire un programme calculant la somme des n premiers termes de la "série harmonique", c'est-à-dire la somme :

$$1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$$

La valeur de n sera lue en donnée.

Exemple d'exécution :

```
Donnez un entier :  
2  
La somme est : 1.5
```

2. Améliorer le programme précédant afin de lire l'entier n , lors de l'exécution du programme.

Exemple d'exécution :

```
C:\>javac Serie.java  
C:\>java Serie 2  
La somme est : 1.5
```



Remarque

- La commande javac permet de compiler la classe Serie.java ;
- La commande java permet d'exécuter le byte code, en lui passant $n = 2$.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution : Exercice 1

Question 1 :

```
import java.util.Scanner;

public class Serie {
    public static void main(String[] args) {
        int n, i = 1;
        float s = 0;
        System.out.print("Donnez la valeur de n : ");
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        while (i <= n) {
            s += (float) 1 / i;
            i++;
        }
        System.out.println("La sommes est : " + s);
    }
}
```



Remarque

- System.out désigne la sortie standard c-a-d la console ;
- System.in désigne l'entrée standard c-a-d le clavier ;
- entier/entier = entier ;
- L'instruction (float) 1/i permet de récupérer aussi la partie décimale, vous pouvez aussi utiliser 1./i.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution : Exercice 1

Question 2 :

```
public class Serie {  
  
    public static void main(String[] args) {  
        int n, i = 1;  
        float s = 0;  
        n = Integer.parseInt(args[0]);  
        while (i <= n) {  
            s += (float) 1 / i;  
            i++;  
        }  
        System.out.println("La sommes est : " + s);  
    }  
}
```



Remarque

- Les arguments transmis au programme au moment de son lancement sont stockés dans le tableau args ;
- La méthode parseInt() de la classe Integer permet de convertir une chaîne de caractère vers un entier.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Exercices

Exercice 2 : Triangle isocèle

Écrire un programme qui affiche un triangle isocèle formé d'étoiles. La hauteur du triangle (c'est-à-dire son nombre de lignes) sera fourni en donnée, comme dans l'exemple ci-dessous.

On s'arrangera pour que la dernière ligne du triangle s'affiche sur le bord gauche de l'écran.

Combien de lignes ? 5

```
      *
     ***
    *****
   *********
  ***********
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution : Exercice 2

```
import java.util.Scanner;

public class Triangle {
    public static void main(String[] args) {
        int n, i, j;
        System.out.print("Combien de lignes ? ");
        Scanner x = new Scanner(System.in);
        n = x.nextInt();
        for (i = 0; i < n; i++) {
            for (j = 0; j < n - i - 1; j++)
                System.out.print(" ");
            for (j = 0; j <= i * 2; j++)
                System.out.print("*");
            System.out.println();
        }
    }
}
```


ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



WEBFORCE
BE THE CHANGE

Exercices

Exercice 1 : Somme des éléments d'un tableau

Écrire un programme permettant de remplir un tableau de 5 éléments, ensuite, calculer et afficher la somme des éléments de ce tableau.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
import java.util.Scanner;

public class Somme {
    public static void main(String[] args) {
        final int n = 5;
        int i;
        float s = 0;
        float t[] = new float[n];
        System.out.println("Donnez les valeurs à saisir dans le tableau : ");
        Scanner x = new Scanner(System.in);
        for (i = 0; i < n; i++) {
            t[i] = x.nextFloat();
        }
        for (i = 0; i < n; i++)
            s += t[i];
        System.out.println("La sommes est : " + s);
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Exercices

Exercice 2 : Carre des nombres impaires

Écrire un programme qui crée un tableau comportant les valeurs des carrés des n premiers nombres impairs, la valeur de n étant lue au clavier et qui en affiche les valeurs sous la forme suivante :

Exemple d'exécution :

```
Combien de valeurs : 5  
1 a pour carre 1  
3 a pour carre 9  
5 a pour carre 25  
7 a pour carre 49  
9 a pour carre 81
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
import java.util.Scanner;

public class Impair {
    public static void main(String[] args) {
        int n;
        int[][] t;
        System.out.print("combien de valeurs : ");
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        t = new int[n][2];
        for (int i = 0; i < n; i++) {
            t[i][0] = 2 * i + 1;
            t[i][1] = t[i][0] * t[i][0];
        }
        for (int i = 0; i < n; i++)
            System.out.println(t[i][0] + " a pour carre " + t[i][1]);
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Exercices

Exercice 3 : Manipulation des tableaux réguliers et les tableaux non réguliers

1. Déclarer les deux tableaux suivants en Java :

2	4	5
3	33	32

Tableau 1 (*Tableau Régulier*)

3		
3	4	5
3	5	
3	1	0

Tableau 2 (*Tableau non régulier*)

2. Créer une fonction permettant d'afficher les éléments de chaque tableau ligne par ligne.
3. Créer un programme de Test (main).

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

Question 1 : Déclaration et initialisation

Tableau régulier

Les trois méthodes suivantes permettent de déclarer un tableau à deux dimensions et d'initialiser son contenu.

```
1.int [][] t = {{2, 4, 5},{3, 32, 33}};  
2.int t [][] = {{2, 4, 5},{3, 32, 33}};  
3.int [] t [] = {{2, 4, 5},{3, 32, 33}};
```

On pourra procéder également comme suit :

Déclaration :

```
int[][] t = new int[2][3];
```

Initialisation :

```
t[0][0] = 2;  
t[0][1] = 4;  
t[0][2] = 5;  
t[1][0] = 3;  
t[1][1] = 32;  
t[1][2] = 33;
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

Question 1 : Déclaration et initialisation

Tableau non régulier

- 1^{ère} Méthode : Déclaration et initialisation

```
int[][] t = {{3}, {3, 4, 5}, {3, 5}, {3, 1, 0}};
```

```
int t [][] = {{3}, {3, 4, 5}, {3, 5}, {3, 1, 0}}; int[] t  
[] = {{3}, {3, 4, 5}, {3, 5}, {3, 1, 0}};
```

- 2^{ème} Méthode : Déclaration

```
int[][] t = new int[3][]; t[0] = new int[1]; t[1] = new  
int[3]; t[2] = new int[2]; t[3] = new int[3];
```

Initialisation :

```
t[0][0] = 3;  
t[1][0] = 3;  
t[1][1] = 4;  
t[1][2] = 5;  
t[2][0] = 3;  
t[2][1] = 5;  
t[3][0] = 3;  
t[3][1] = 1;  
t[3][2] = 0;
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

Question 2 : Méthode pour parcourir les tableaux à deux dimensions ligne par ligne

```
public static void affiche(int [][] t){
    for (int[] t1 : t) {
        for (int j = 0; j < t1.length; j++) {
            System.out.print("\t" + t1[j]);
        }
        System.out.println("");
    }
}
```


ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
public class Tab {  
    public static void affiche(int [][] t){...}  
    public static void main(String[] args) {  
        int[][] t = new int[4][];  
        t[0] = new int[1];  
        t[1] = new int[3];  
        t[2] = new int[2];  
        t[3] = new int[3];  
        t[0][0] = 3;  
        t[1][0] = 3;  
        t[1][1] = 4;  
        t[1][2] = 5;  
        t[2][0] = 3;  
        t[2][1] = 5;  
        t[3][0] = 3;  
        t[3][1] = 1;  
        t[3][2] = 0;  
        affiche(t);  
    }  
}
```

Résultat d'exécution :

```
3  
3 4 5  
3 5  
3 1 0
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Exercices

Exercice 4 : Manipulation des tableaux réguliers et les tableaux non réguliers

Réaliser une classe utilitaire concernant des tableaux de tableaux de valeurs de type *double* et contenant les méthodes statiques suivantes :

- ***affiche (double t [] [])*** : affiche les valeurs de *t*, à raison d'une ligne d'écran pour une ligne du tableau ;
- ***boolean regulier (double t [] [])*** : teste si le tableau *t* est régulier, c'est-à-dire si toutes ses lignes ont la même taille ;
- ***double [] sommeLignes (double t [] [])*** : fournit un tableau de *double* correspondant aux sommes des différentes lignes de *t* ;
- ***double [] [] somme (double [] [] t1, double [] [] t2)*** : s'assure que les tableaux *t1* et *t2* sont réguliers et de mêmes dimensions et fournit dans ce cas leur somme en résultat ; dans le cas contraire, elle fournit une référence nulle.

Écrire un petit programme de test.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

Etape 1 : Créer la classe **Utilitaire**

Etape 2 : Dans la classe **Utilitaire** créer les méthodes statiques suivantes :

```
public static void affiche(double t[][]) {
    for (int i = 0; i < t.length; i++) {
        for (int j = 0; j < t[i].length; j++)
            System.out.print(t[i][j] + " ");
        System.out.println();
    }
}

public static boolean regulier(double t[][]) {
    boolean test = true;
    int n = t[0].length;
    for (int i = 1; i < t.length; i++)
        if (n != t[i].length)
            return false;
    return true;
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
public static double[] sommeLignes(double t[][]) {  
    double[] total;  
    total = new double[t.length];  
    for (int i = 0; i < t.length; i++)  
        for (int j = 0; j < t[i].length; j++)  
            total[i] += t[i][j];  
    return total;  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
public static double[][] somme(double[][] t1, double[][] t2) {
    double[][] t3;
    t3 = new double[t1.length][t1[0].length];
    if (regulier(t1) && regulier(t2)) {
        if (t1.length == t2.length) {
            if (t1[0].length == t2[0].length) {
                for (int i = 0; i < t1.length; i++)
                    for (int j = 0; j < t1[i].length; j++)
                        t3[i][j] = t1[i][j] + t2[i][j];
                return t3;
            } else {
                System.out
                    .println("Les tableaux n'ont pas le même nombres de colonnes");
                return null;
            }
        } else {
            System.out
                .println("Les tableaux n'ont pas le même nombres de lignes");
            return null;
        }
    } else {
        System.out.println("Les tableaux ne sont pas régulier");
        return null;
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Bases du langage JAVA



Solution

```
public static void main(String[] args) {  
  
    double table1[][] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };  
  
    double table2[][] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };  
  
    affiche(table1);  
  
    if (regulier(table1))  
        System.out.println("Tableau régulier");  
    else  
        System.out.println("Tableau non régulier");  
  
    for (double i : sommeLignes(table1))  
        System.out.println(i);  
  
    affiche(somme(table1, table2));  
  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Association : Professeur – Spécialité

Soit le diagramme de classe suivant :

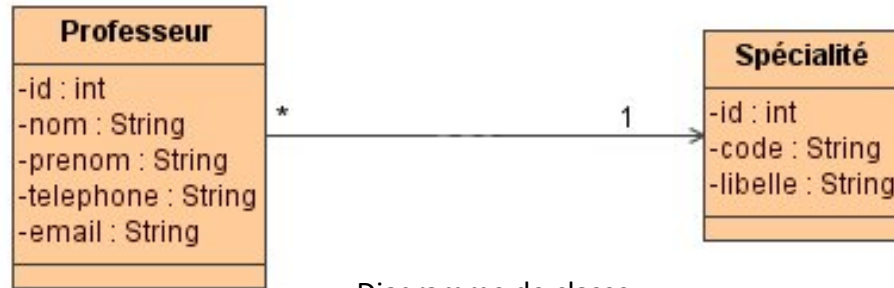


Diagramme de classe

1. Développer la classe Spécialité et la classe Professeur dans le package «**ma.projet.ecole**».

Chaque classe doit comporter:

- Un constructeur d'initialisation ;
- Les accesseurs ;
- La méthode toString.

NB : l'identifiant est auto incrément.

2. Développer une classe de test dans le package « **ma.projet.ecole.test** ».

Dans la classe de test créer :

- **5 Spécialités :**
 - JAVA/JEE ;
 - .net ;
 - Gestion de projet ;
 - CISCO ;
 - PHP.
- **4 Professeurs :** Les deux premiers enseignent JAVA/JEE les deux autres enseignent CISCO.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Association : Professeur – Spécialité

Afficher la liste des professeurs par spécialité:

Exemple :

- Spécialité JAVA/JEE
 - SAFI Amal safi@gmail.com
 - ALAMI Said alami@yahoo.fr
- Spécialité CISCO
 - ALAOUI Reda alaoui@yahoo.fr
 - KAMALI Imane i.kamali@gmail.com



Remarque

Le nom de professeur doit être en majuscule, le prénom doit commencer par une lettre en majuscule et les autres lettres doivent être en minuscule.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Solution : Association : Professeur – Spécialité

Etape 1 : Créer un projet sous **Android Studio** ;

Etape 2 : Spécifier le package de votre projet « ma.projet.ecole » lors de la création ;

Etape 3 : Choisir « **No Activity** » et cliquer sur finish ;

Etape 4 : Créer la classe Filière ;

Etape 5 : Créer la classe Professeur ;

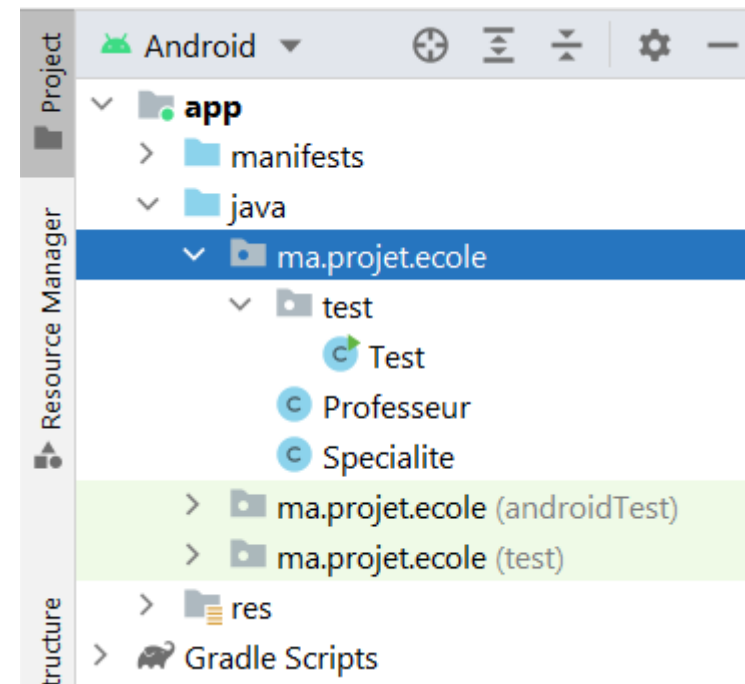
Etape 6 : Créer le package « ma.projet.ecole.test » ;

Etape 7 : Créer la classe Test ;

Etape 8 : Lancer la classe Test. Si vous rencontrez un problème.

Configurer `/${yourproject}/.idea/gradle.xml`:

```
<GradleProjectSettings>
  <option name="delegatedBuild" value="false" />
  ....
</GradleProjectSettings>
```



Structure de projet final

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Solution : Association : Professeur – Spécialité

Pour les étapes 1, 2 et 3, elles figurent dans la première partie.

Etape 4 : Création de classe Filière.

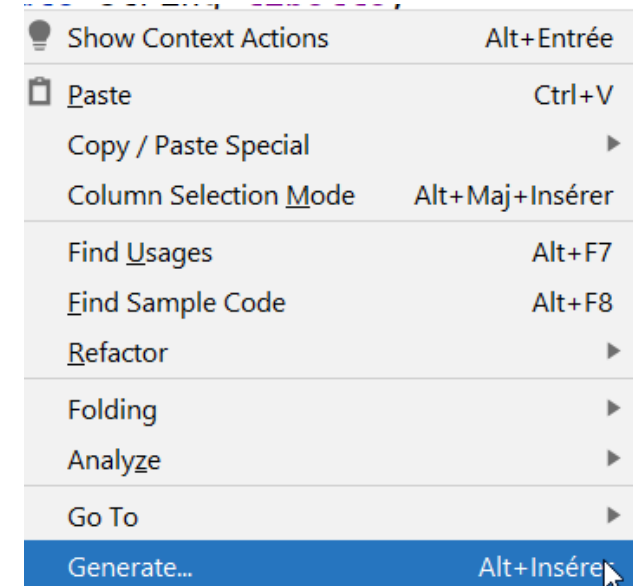
1. Créer la classe et définir ses attributs :

```
public class Specialite {  
    private int id;  
    private String code;  
    private String libelle;  
    private static int count;
```

2. Après la définition des attributs de la classe, **cliquez droit à l'intérieur de classe**, ensuite cliquez sur **Generate**, afin de générer le constructeur, les accesseurs, la méthode toString() ...

3. Modifier le constructeur afin de rendre id auto-incrément :

```
public Specialite(String code, String libelle) {  
    this.id = ++count; ←  
    this.code = code;  
    this.libelle = libelle;  
}
```



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Solution : Association : Professeur – Spécialité

Etape 5 : Créer la classe Professeur.

Ici on a une navigation dans un seul sens, donc :

```
public class Professeur {  
    private int id;  
    private String nom;  
    private String prenom;  
    private String telephone;  
    private String email;  
    private Specialite specialite;  
    private static int count;  
}
```



Générer le constructeur, les accesseurs et la méthode toString ()

Redéfinir la méthode toString () pour respecter la condition d'affichage :

```
public String toString() {  
    return "-" + this.nom.toUpperCase() + " "  
        + this.prenom.substring(0, 1).toUpperCase() + ""  
        + this.prenom.substring(1).toLowerCase() + " " + this.email;  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Solution : Association : Professeur – Spécialité

Etape 7 : Créer la classe Test.

1. Dans la classe de Test, tapez **main** ensuite **ctrl + espace**, IDE génère automatiquement le corps de la méthode main.

```
import ma.ofppt.ecole.Professeur;  
import ma.ofppt.ecole.Specialite;
```

```
public class Test {  
  
    public static void main(String[] args) {  
        //code ici  
    }  
}
```

2. Dans la méthode main, créez un tableau des spécialités :

```
Specialite[] specialites = new Specialite[5];  
// Création des spécialités  
specialites[0] = new Specialite("S1", "JAVA/JEE");  
specialites[1] = new Specialite("S2", ".net");  
specialites[2] = new Specialite("S3", "Gestion de projet");  
specialites[3] = new Specialite("S4", "CISCO");  
specialites[4] = new Specialite("S5", "PHP");
```

3. Ensuite, créez un tableau des professeurs et attribuez à chaque professeur une spécialité.

```
// Création des professeurs  
Professeur professeurs[] = new Professeur[4];  
professeurs[0] = new Professeur("Safi", "salim", "0656787567",  
    "safi@gmail.com", specialites[0]);  
professeurs[1] = new Professeur("Rami", "amal", "0654487567",  
    "rami@gmail.com", specialites[3]);  
professeurs[2] = new Professeur("Rashidi", "Mohamed", "0656777567",  
    "rashidi@gmail.com", specialites[0]);  
professeurs[3] = new Professeur("Simon", "thomas", "0654445567",  
    "simon@gmail.com", specialites[3]);
```

4. Affichez la liste des professeurs :

```
for(Professeur p :professeurs){  
    System.out.println(p);  
}
```

5. Maintenant, affichez la liste des professeur par spécialité.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes et Objets



Solution : Association : Professeur – Spécialité

```
// Professeurs par spécialité
System.out.println("Professeurs par spécialité : ");
for (Specialite s : specialites) {
    System.out.println("\t" + s);
    int etat = 0;
    for (Professeur p : professeurs) {
        if (p.getSpecialite().getId() == s.getId()) {
            System.out.println("\t\t" + p);
            etat = 1;
        }
    }
    if (etat == 0) {
        System.out.println("\t\tAucun professeur dans cette spécialité");
    }
}
```

```
Run: Test x
Professeurs par spécialité :
JAVA/JEE
    -SAFI Salim safi@gmail.com
    -RASHIDI Mohamed rashidi@gmail.com
.net
    Aucun professeur dans cette spécialité
Gestion de projet
    Aucun professeur dans cette spécialité
CISCO
    -RAMI Amal rami@gmail.com
    -SIMON Thomas simon@gmail.com
PHP
    Aucun professeur dans cette spécialité

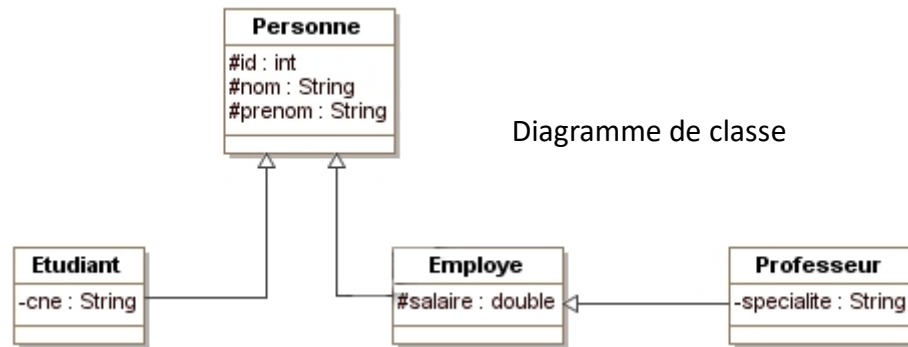
Process finished with exit code 0
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage

Héritage à plusieurs niveaux

Soit le diagramme de classe suivant :



Description

- La classe **Etudiant** hérite de la classe **Personne** ;
- La classe **Professeur** hérite de la classe **Employe** et la classe **Employe** hérite de la classe **Personne** ;
- Un **Etudiant** est une **Personne** ;
- Un **Professeur** est un **Employe** et un **Employe** est une **Personne**.

Travail à faire

1. Développer les classes dans le package "**ma.projet.classes**".
 - Chaque classe doit contenir un constructeur d'initialisation ;
 - Chaque classe doit redéfinir la méthode **toString()**.
2. Développer une classe **Application** dans le package "**ma.projet.test**", dans cette classe on demande de créer :
 - deux étudiants ;
 - deux employés ;
 - deux professeurs ;
 - afficher les informations de chaque personne.



Remarque

Visibilités dans un diagramme de classe :

- # : protected
- : private
- + : public

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Héritage à plusieurs niveaux

Exemple d'exécution :

```
La liste des employés :  
  Je suis DOUK Rachid mon salaire est: 10000.0 DH  
  Je suis NGOYE Roland mon salaire est: 10000.0 DH  
La liste des étudiants :  
  Je suis OBAKA Med mon CNE est: 65678754  
  Je suis ALSENY Thomas mon CNE est: 87543543  
La liste des professeurs :  
  Je suis OBA Kevin mon salaire est: 5700.0 DH ma spécialité est: JAVA/JEE  
  Je suis MAGASSOUBA Cheick mon salaire est: 5000.0 DH ma spécialité est: Mathématique
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Solution : Héritage à plusieurs niveaux

Etape 1 : Créer un projet sur **Android Studio**, avec le package « ma.projet » ;

Etape 2 : Créer les deux sous packages « ma.projet.classes » et « ma.projet.test » ;

Etape 3 : Créer la classe **Personne** ;

Etape 4 : Créer la classe **Etudiant** ;

Etape 5 : Créer la classe **Employe** ;

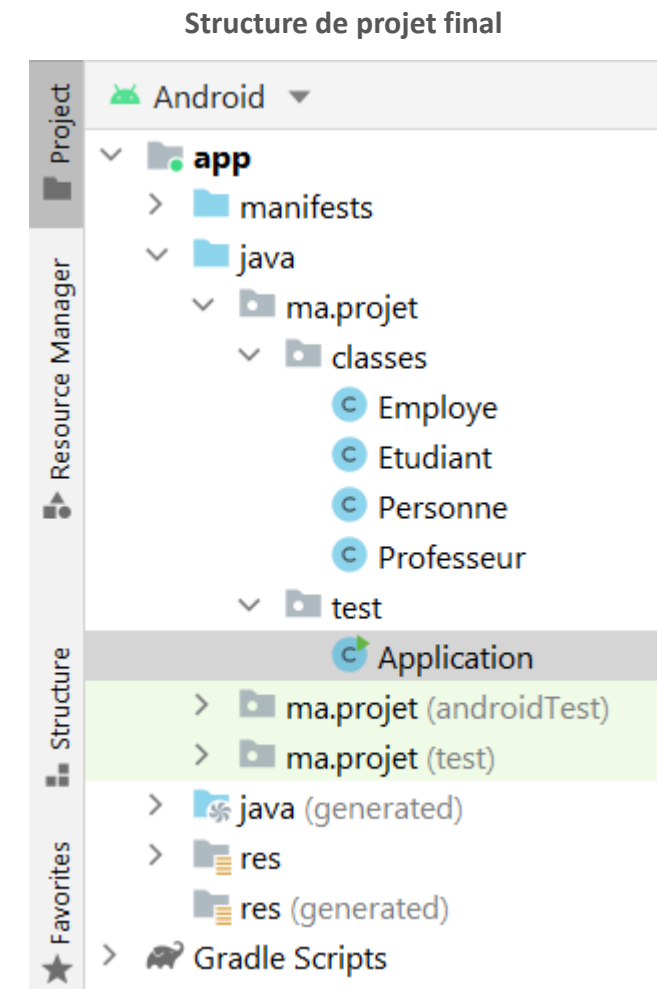
Etape 6 : Créer la classe **Professeur** ;

Etape 7 : Créer la classe **Application**. Ensuite lancer la classe **Application**.

Si vous rencontrez un problème :

Configurer `/${yourproject}/.idea/gradle.xml` :

```
<GradleProjectSettings>
  <option name="delegatedBuild" value="false" />
  ....
```



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Solution : Héritage à plusieurs niveaux

```
package ma.projet.classes;

public class Personne {
    protected int id;
    protected String nom;
    protected String prenom;

    private static int count;

    public Personne(String nom, String prenom) {
        this.id = ++count;
        this.nom = nom;
        this.prenom = prenom;
    }

    @Override
    public String toString() {
        return "Je suis " + this.nom.toUpperCase() + " "
            + this.prenom.substring(0, 1).toUpperCase() + ""
            + this.prenom.substring(1).toLowerCase();
    }
}
```

```
package ma.projet.classes;

public class Etudiant extends Personne {

    private String cne;

    public Etudiant(String nom, String prenom, String cne) {
        super(nom, prenom);
        this.cne = cne;
    }

    @Override
    public String toString() {
        return super.toString() + " mon CNE est: " + this.cne;
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Solution : Héritage à plusieurs niveaux

```
package ma.projet.classes;

public class Employe extends Personne {

    protected double salaire;

    public Employe(String nom, String prenom, double salaire) {
        super(nom, prenom);
        this.salaire = salaire;
    }

    public String toString() {
        return super.toString() + " mon salaire est: " + this.salaire + " DH";
    }

}
```

```
public class Professeur extends Employe {

    private String specialite;

    public Professeur(String nom, String prenom, double salaire,
        String specialite) {
        super(nom, prenom, salaire);
        this.specialite = specialite;
    }

    @Override
    public String toString() {
        return super.toString() + " ma spécialité est: " + this.specialite;
    }

}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Solution : Héritage à plusieurs niveaux

```
package ma.projet.test;

import ma.projet.classes.Employe;
import ma.projet.classes.Etudiant;
import ma.projet.classes.Professeur;

public class Application {
    public static void main(String[] main) {
        Employe[] employes = new Employe[2];
        employes[0] = new Employe("Douk", "Rachid", 10000);
        employes[1] = new Employe("Ngoye", "Roland", 10000);
        System.out.println("La liste des employes : ");
        for (Employe e : employes)
            System.out.println("\t" + e);
    }
}
```

```
Etudiant[] etudiants = new Etudiant[2];
etudiants[0] = new Etudiant("Obaka", "Med", "65678754");
etudiants[1] = new Etudiant("Alseny", "Thomas", "87543543");
System.out.println("La liste des étudiants : ");
for (Etudiant e : etudiants)
    System.out.println("\t" + e);

Professeur[] professeurs = new Professeur[2];
professeurs[0] = new Professeur("Oba", "Kevin", 5700, "JAVA/JEE");
professeurs[1] = new Professeur("Magassouba", "Cheick", 5000,
    "Mathématique");
System.out.println("La liste des professeurs : ");
for (Professeur p : professeurs)
    System.out.println("\t" + p);
}
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Héritage



Solution : Héritage à plusieurs niveaux

```
Run: Application x
"C:\Program Files\Android\Android Studio1\jre\bin\java.exe" ...
La liste des employes :
    Je suis DOUK Rachid mon salaire est: 10000.0 DH
    Je suis NGOYE Roland mon salaire est: 10000.0 DH
La liste des étudiants :
    Je suis OBAKA Med mon CNE est: 65678754
    Je suis ALSENY Thomas mon CNE est: 87543543
La liste des professeurs :
    Je suis OBA Kevin mon salaire est: 5700.0 DH ma spécialité est: JAVA/JEE
    Je suis MAGASSOUBA Cheick mon salaire est: 5000.0 DH ma spécialité est: Mathématique

Process finished with exit code 0
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

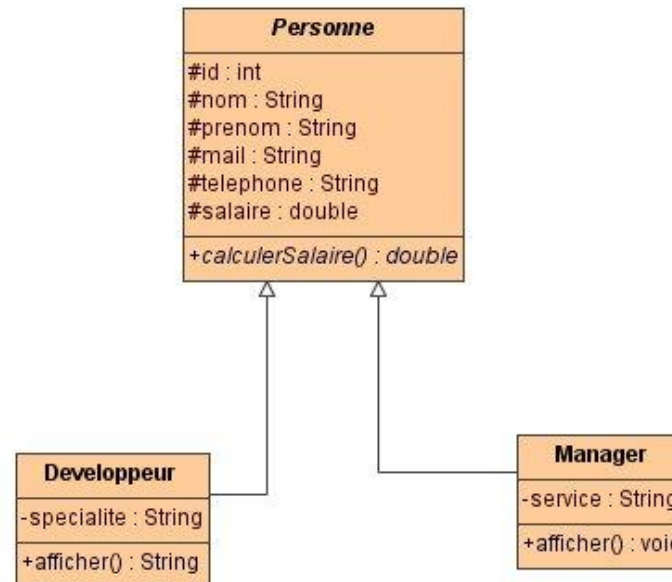
Classes abstraites



Hierarchie de type Personne

Le directeur des systèmes d'information de la société TOUBKAL-IT souhaite développer un module pour la gestion des utilisateurs de son service, il vous a fait appel pour réaliser cette tâche.

Le diagramme de classe a été établi par un analyste afin de mettre en place une base de données sous ORACLE ou MySQL :



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes abstraites



Hierarchie de type Personne

1. Créer la classe abstraite « **Personne** » dans le package « **ma.projet** » ;
2. Créer les classes « **Developpeur** » et « **Manager** » dans le package « **ma.projet.beans** » ;

Qu'est-ce que vous remarquez ?

3. Redéfinir la méthode **calculerSalaire()** :

Sachant que :

- Le développeur aura une augmentation de 20% par rapport à son salaire normal
- Le manager aura une augmentation de 35% par rapport à son salaire normal

4. Créer deux développeurs et deux managers ;
5. Afficher les informations des objets créés :

Sous la forme :

- Le salaire du manager LACHGAR Mohamed est : 30 000 dh, son service : Informatique
- Le salaire du développeur Salim karim est : 10 000 dh, sa spécialité : PHP

6. Créer un objet de type Personne. Qu'est-ce que vous remarquez ?

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

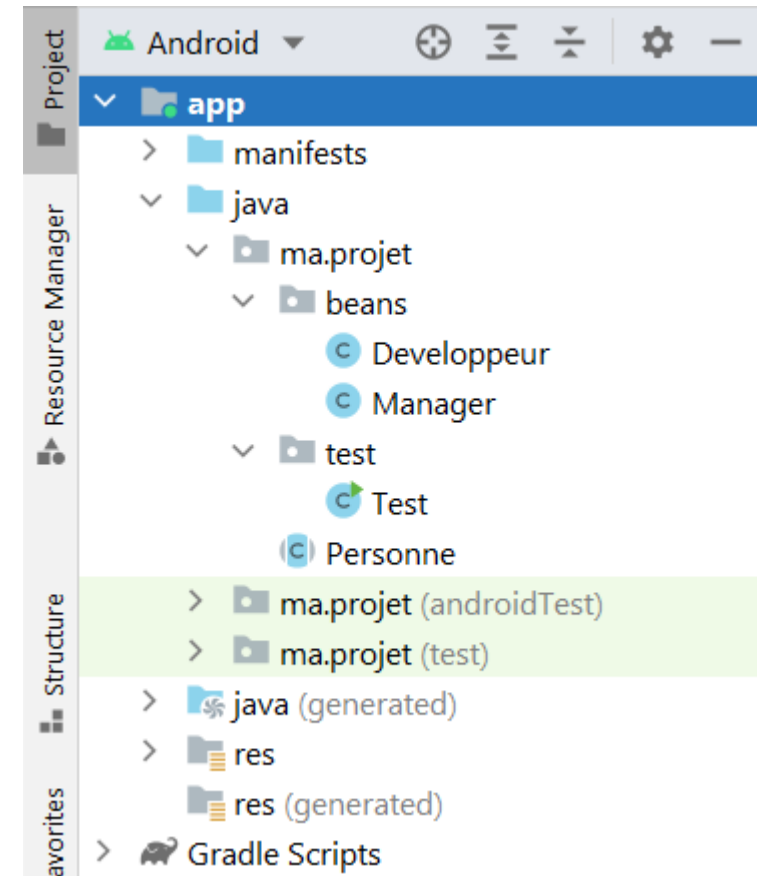
Classes abstraites



Solution : Hiérarchie de type Personne

1. **Etape 1** : Créer un projet sur Android Studio, mentionner comme package « ma.projet », puis sélectionner « No Activity », ensuite cliquer sur finish ;
2. **Etape 2** : Créer les sous package « ma.projet.beans » et « ma.projet.test » ;
3. **Etape 3** : Créer la classe abstraite Personnes ;
4. **Etape 4** : Créer les classes Manager et Employe ;
5. **Etape 5** : Redéfinir la méthode `calculerSalaire ()` et la méthode `affiche()` dans chacune des classes ;
6. **Etape 6** : Créer la classe Test.

Structure de projet final



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes abstraites



Solution : Hiérarchie de type Personne

```
public abstract class Personne {
    protected int id;
    protected String nom;
    protected String prenom;
    protected String mail;
    protected String telephone;
    protected double salaire;
    private static int cmp;

    public Personne(String nom, String prenom, String mail, String telephone,
        double salaire) {
        this.id = ++cmp;
        this.nom = nom;
        this.prenom = prenom;
        this.mail = mail;
        this.telephone = telephone;
        this.salaire = salaire;
    }

    public abstract double calculerSalaire();
}
```

```
public class Developpeur extends Personne {

    private String specialite;

    public Developpeur(String nom, String prenom, String mail,
        String telephone, double salaire, String specialite) {
        super(nom, prenom, mail, telephone, salaire);
        this.specialite = specialite;
    }

    @Override
    public double calculerSalaire() {
        return 1.2 * this.salaire;
    }

    public String affiche() {
        return "Le salaire du Developpeur " + this.nom + " " + this.prenom
            + " est : " + this.calculerSalaire() + " sa specialite : "
            + this.specialite;
    }
}
```


ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes abstraites



Solution : Hiérarchie de type Personne

```
public class Manager extends Personne {
    private String service;

    public Manager(String nom, String prenom, String mail, String telephone,
        double salaire, String service) {
        super(nom, prenom, mail, telephone, salaire);
        this.service = service;
    }

    @Override
    public double calculerSalaire() {
        return 1.35 * this.salaire;
    }

    public void affiche() {
        System.out.println("Le salaire du Manager " + this.nom + " "
            + this.prenom + " est " + this.calculerSalaire()
            + " son service est " + this.service);
    }
}
```

```
public class Test {

    public static void main(String[] args) {
        Manager managers[] = new Manager[2];
        managers[0] = new Manager("RAMI", "Amal", "rami@gmail.com", "0563424",
            2000, "Informatique");
        managers[1] = new Manager("ALAMI", "Said", "rami@gmail.com", "0563424",
            4000, "Comptabilité");
        for (Manager m : managers)
            m.affiche();

        Developpeur devs[] = new Developpeur[2];
        devs[0] = new Developpeur("SENHADJI", "Samir", "samir@gmail.com",
            "0634435", 4000, "JAVA");
        devs[1] = new Developpeur("ALAOUI", "Karim", "samir@gmail.com",
            "0634435", 6000, "C++");
        for (Developpeur dev : devs) {
            System.out.println(dev.affiche());
        }
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Classes abstraites



Solution : Hiérarchie de type Personne

```
"C:\Program Files\Android\Android Studio1\jre\bin\java.exe" ...  
Le salaire du Manager RAMI Amal est 2700.0 son service est Informatique  
Le salaire du Manager ALAMI Saïd est 5400.0 son service est Comptabilité  
Le salaire du Developpeur SENHADJI Samir est : 4800.0 sa specialite : JAVA  
Le salaire du Developpeur ALAOUI Karim est : 7200.0 sa specialite : C++  
  
Process finished with exit code 0
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Polymorphisme - Classe Personne

Exercice 1 :

Ecrire les classes nécessaires au fonctionnement du programme suivant (en ne fournissant que les méthodes nécessaires à ce fonctionnement). N'oubliez pas de commenter votre code :

```
public class TestMetiers {  
    public static void main(String[] argv) {  
        Personne[] personnes = new Personne[3];  
        personnes[0] = new Menuisier("Amine");  
        personnes[1] = new Plombier("Ahmed");  
        personnes[2] = new Menuisier("Ali");  
        for (int i = 0; i < personnes.length; i++)  
            personnes[i].affiche();  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Solution : Polymorphisme - Classe Personne

La classe Personne :

```
public abstract class Personne {  
    protected String nom;  
  
    public Personne(String nom) {  
        this.nom = nom;  
    }  
    public abstract void affiche();  
}
```

La classe Plombier :

```
public class Plombier extends Personne {  
  
    public Plombier(String nom) {  
        super(nom);  
    }  
  
    @Override  
    public void affiche() {  
        System.out.println("Je suis " + this.nom + " Le "  
            + this.getClass().getSimpleName());  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Solution : Polymorphisme - Classe Personne

La classe Menuisier :

```
public class Menuisier extends Personne {  
  
    public Menuisier(String nom) {  
        super(nom);  
    }  
  
    @Override  
    public void affiche() {  
        System.out.println("Je suis " + this.nom + " Le "  
        + this.getClass().getSimpleName());  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Polymorphisme - Classe Forme

Exercice 2 :

Ecrire les classes nécessaires au fonctionnement du programme suivant (en ne fournissant que les méthodes nécessaires à ce fonctionnement). N'oubliez pas de commenter votre code :

```
import java.text.DecimalFormat;

public class TestFormes {
    public static void main(String[] argv) {
        DecimalFormat df = new DecimalFormat("##");
        Forme[] figures = new Forme[3];
        figures[0] = new Carre(2); // Création d'un carré de 2 cm de coté
        figures[1] = new Cercle(3); // Création d'un cercle de 3 cm de rayon
        figures[2] = new Carre(5.2); // Création d'un carré de 5,2 cm de coté
        for (int i = 0; i < figures.length; i++)
            System.out.println(figures[i] + " : surface = "
                + df.format(figures[i].getSurface()) + " cm2");
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Solution : Polymorphisme - Classe Personne

La classe **Forme** :

```
public abstract class Forme {  
  
    public abstract double getSurface();  
  
}
```

La classe **Carre** :

```
public class Carre extends Forme {  
    private double cote;  
  
    public Carre(double cote) {  
        this.cote = cote;  
    }  
  
    @Override  
    public double getSurface() {  
        return Math.pow(cote, 2);  
    }  
  
    public String toString() {  
        return "Carré ( coté " + this.cote + " cm )";  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Polymorphisme



Solution : Polymorphisme - Classe Personne

La classe Cercle :

```
public class Cercle extends Forme {
    private double rayon;

    public Cercle(double rayon) {
        this.rayon = rayon;
    }

    @Override
    public double getSurface() {
        return Math.PI * Math.pow(rayon, 2);
    }

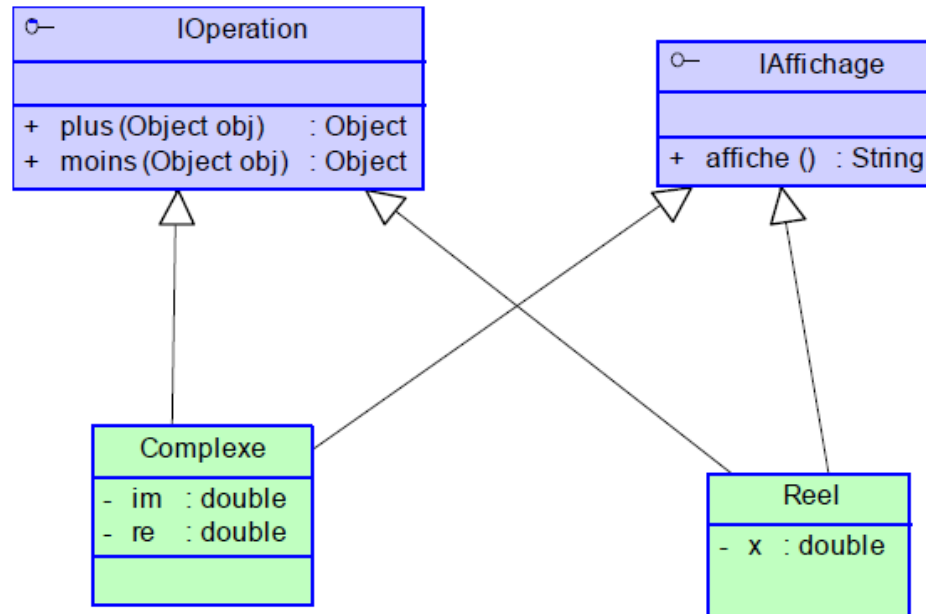
    public String toString() {
        return "Cercle ( rayon " + this.rayon + " cm )";
    }
}
```


ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces

Classes Complexe & réel

Soit le diagramme de classe suivant :



1. Créer l'interface **IOperation** et **IAffichage** dans le package «**ma.projet.inter** ».
2. Créer les classes **Complexe** et **Reel** dans le package « **ma.projet.beans** ».

Redéfinir les méthodes **moins()**, **plus()** et **affiche()** dans la classe complexe sachant que :

- Un nombre complexe est un nombre qui comporte deux parties : une partie réelle et une partie imaginaire ;
 - Un nombre complexe est généralement écrit sous la forme : $4+3i$ (partie réelle = 4 et partie imaginaire = 3) ;
 - L'addition (la soustraction) de deux nombres complexes consiste à additionner (soustraire) les parties réelles de ces deux nombres pour obtenir la partie réelle du résultat et à additionner (soustraire) les deux parties imaginaires pour obtenir la partie imaginaire du résultat.
3. Surcharger les méthodes **plus()** et **moins ()** dans la classe **Reel**.

Nouvelle signature :

- double plus (double...x) ;
- double moins (double ...x) ;

Qu'est ce que vous remarquez ?

4. Créer un programme de test.

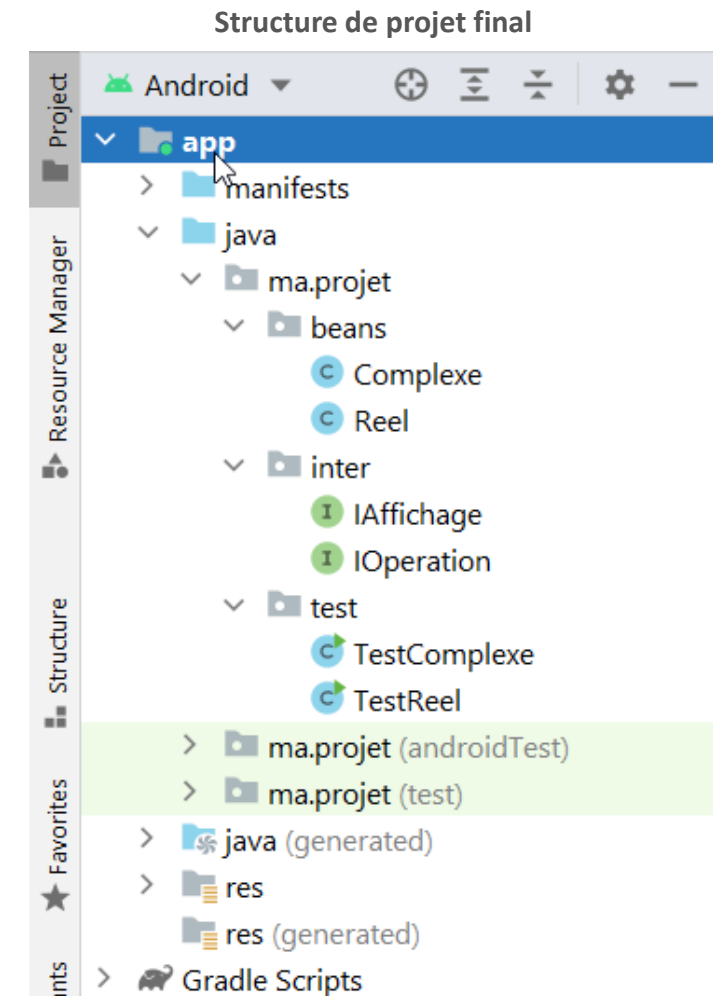
ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces



Classes Complexe & réel

1. **Etape 1** : Créer un projet sur Android Studio, mentionner comme package « ma.projet », puis sélectionner « No Activity », ensuite cliquer sur finish ;
2. **Etape 2** : Créer les sous package « ma.projet.beans », « ma.projet.test » et « ma.projet.inter » ;
3. **Etape 3** : Créer les interfaces IAffichage et IOpearation ;
4. **Etape 4** : Créer les classes Complexe et Reel ;
5. **Etape 5** : Créer les classes TestComplexe et TestReel ;



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces



Classes Complexe & réel

Interface IAffichage

```
package ma.projet.inter;
```

```
public interface IAffichage {  
    String affiche();  
}
```

Interface IOperation

```
package ma.projet.inter;
```

```
public interface IOperation {  
    Object plus (Object o);  
    Object moins (Object o);  
}
```

```
public class Reel implements IOperation, IAffichage {
```

```
    private int x;
```

```
    public Reel(int x) {  
        this.x = x;  
    }
```

```
    @Override  
    public String affiche() {  
        return this.x + "";  
    }
```

```
    @Override  
    public Object plus(Object o) {  
        Reel r =(Reel) o;  
        return new Reel(this.x + r.x);  
    }
```

```
    @Override  
    public Object moins(Object o) {  
        Reel r =(Reel) o;  
        return new Reel(this.x - r.x);  
    }
```

```
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces



Classes Complexe & réel

```
public class TestReel {  
    public static void main(String[] args) {  
        Reel r1 = new Reel(3);  
        Reel r2 = new Reel(4);  
        Reel r3, r4;  
        r3 = (Reel) r1.plus(r2);  
        r4 = (Reel) r1.moins(r2);  
        System.out.println("r1 + r2 = " + r3.affiche());  
        System.out.println("r1 - r2 = " + r4.affiche());  
    }  
}
```

```
public class TestComplexe {  
    public static void main(String[] args) {  
        Complexe c1 = new Complexe(2, 2);  
        Complexe c2 = new Complexe(-3, 4);  
        Complexe c3, c4;  
        c3 = (Complexe) c1.plus(c2);  
        c4 = (Complexe) c1.moins(c2);  
        System.out.println("c1 + c2 = " + c3.affiche());  
        System.out.println("c1 - c2 = " + c4.affiche());  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces



Classes Complexe & réel

Interface IAffichage

```
package ma.projet.inter;
```

```
public interface IAffichage {  
    String affiche();  
}
```

Interface IOperation

```
package ma.projet.inter;
```

```
public interface IOperation {  
    Object plus (Object o);  
    Object moins (Object o);  
}
```

```
public class Complexe implements IOperation, IAffichage {  
    private int re;  
    private int im;
```

```
    public Complexe(int re, int im) {  
        this.re = re;  
        this.im = im;  
    }
```

```
    @Override  
    public String affiche() {  
        if (im < 0)  
            return this.re + " " + this.im + " i";  
        else if (im > 0)  
            return this.re + " + " + this.im + " i";  
  
        return this.re + "";  
    }
```

```
    @Override  
    public Object plus(Object o) {  
        Complexe c = (Complexe) o;  
        return new Complexe(this.re + c.re, this.im + c.im);  
    }
```

```
    @Override  
    public Object moins(Object o) {  
        Complexe c = (Complexe) o;  
        return new Complexe(this.re - c.re, this.im - c.im);  
    }
```

```
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Interfaces



Test Réel :

```
▶ ↑ "C:\Program Files\Android\Android Studio1\jre\bin\java.exe"  
⚙ ↓ r1 + r2 = 7  
■ ↻ r1 - r2 = -1  
⚙ ↻  
▶ ⏸ Process finished with exit code 0
```

Test Complexe :

```
▶ ↑ "C:\Program Files\Android\Android Studio1\jre\bin\java.exe" ...  
⚙ ↓ c1 + c2 = -1 + 6 i  
■ ↻ c1 - c2 = 5 -2 i  
⚙ ↻  
▶ ⏸ Process finished with exit code 0
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Gestion des salles

Couche métier (package : ma.projet.beans)

1. Définir une classe Salle avec les attributs suivants : *id, code et libelle* ;
2. Définir les accesseurs aux différents attributs de la classe ;
3. Définir un **constructeur** permettant d'initialiser les attributs d'un objet salle par des valeurs saisies par l'utilisateur. Sachant que Id doit être auto-incrément ;
4. Définir la méthode **toString ()** permettant d'afficher les informations de la salle en cours.

Couche accès aux données (package : ma.projet.dao)

1. Créer l'interface générique **IDao** avec les méthodes :
 - boolean create (T o) : Méthode permettant d'ajouter un objet o de type T ;
 - boolean delete (T o) : Méthode permettant de supprimer un objet o de type T ;
 - boolean update (T o) : Méthode permettant de modifier un objet o de type T ;

- T findById (int id) : Méthode permettant de renvoyer un objet dont id est passé en paramètre ;
- List <T> findAll () : Méthode permettant de renvoyer la liste des objets de type T ;

2. Créer la classe **SalleService** qui implémente l'interface **IDao** dans le package « ma.projet.service ». Dans cette classe les données seront stockés dans une collection de type List.

Couche de présentation (package : ma.projet.test)

1. Dans une classe de test :
 - Créer cinq salles ;
 - Afficher la liste des salles ;
 - Supprimer une salle ;
 - Modifier les informations d'une salle ;
 - Afficher la liste des salles.

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Gestion des salles

Exemple d'exécution :

```
La liste des salles :
```

```
A Salle 1  
B Salle 2  
C Salle 3  
D Salle 4
```

```
Supprimer la salle avec id = 1
```

```
Modifier la salle avec id = 2
```

```
    Salle à modifier : B Salle 2
```

```
Donner le nouveau code :
```

```
BB
```

```
Donner le nouveau libelle :
```

```
Salle 10
```

```
La liste des salles après les mises à jour :
```

```
BB Sale 10  
C Salle 3  
D Salle 4
```

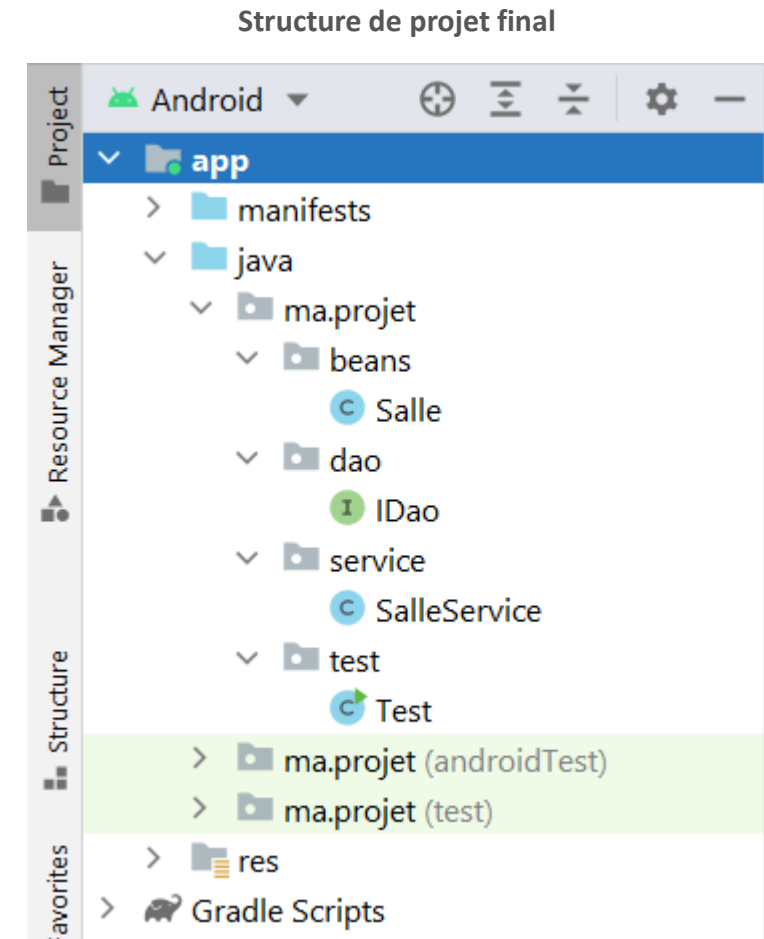

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Solution : Gestion des salles

- Etape 1** : Créer un projet sur Android Studio, mentionner comme package « ma.projet », puis sélectionner « No Activity », ensuite cliquer sur finish ;
- Etape 2** : Créer les sous package « ma.projet.beans », « ma.projet.dao », « ma.projet.test » et « ma.projet.service » ;
- Etape 3** : Créer la classe Salle ;
- Etape 4** : Créer l'interface générique IDao ;
- Etape 5** : Créer la classe SalleService qui implémente l'interface IDao ;
- Etape 5** : Créer la classe Test.



ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Solution : Gestion des salles

Classe Salle :

```
public class Salle {
    private int id;
    private String code;
    private String libelle;

    private static int comp;

    public Salle(String code, String libelle) {
        this.id = ++comp;
        this.code = code;
        this.libelle = libelle;
    }

    public Salle(int id, String code, String libelle) {
        this.id = id;
        this.code = code;
        this.libelle = libelle;
    }
    //getters and setters
    //toString()
}
```

L'interface IDao :

```
import java.util.List;

public interface IDao <T> {

    boolean create(T o);

    boolean update(T o);

    boolean delete(T o);

    List<T> findAll();

    T findById(int id);
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Solution : Gestion des salles

Classe SalleService :

```
public class SalleService implements IDao<Salle> {  
  
    private List<Salle> salles;  
  
    public SalleService() {  
        salles = new ArrayList<Salle>();  
    }  
  
    @Override  
    public boolean create(Salle o) {  
        return salles.add(o);  
    }  
  
    @Override  
    public boolean update(Salle o) {  
        for(Salle s : salles){  
            if(s.getId() == o.getId()){  
                s.setCode(o.getCode());  
                s.setLibelle(o.getLibelle());  
                return true;  
            }  
        }  
        return false;  
    }  
}
```

```
@Override  
public boolean delete(Salle o) {  
    return salles.remove(o);  
}  
  
@Override  
public List<Salle> findAll() {  
    return salles;  
}  
  
@Override  
public Salle findById(int id) {  
    for (Salle s : salles) {  
        if (s.getId() == id)  
            return s;  
    }  
    return null;  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Solution : Gestion des salles

```
public class Test {  
    public static void main(String[] args) {  
        SalleService ss = new SalleService();  
  
        ss.create(new Salle("A", "Salle 1"));  
        ss.create(new Salle("B", "Salle 2"));  
        ss.create(new Salle("C", "Salle 3"));  
        ss.create(new Salle("D", "Salle 4"));  
  
        System.out.println("La liste des salles :");  
        for (Salle s : ss.findAll())  
            System.out.println("\t" + s);  
  
        System.out.println("Supprimer la salle avec id = 1");  
        ss.delete(ss.findById(1));  
    }  
}
```

```
System.out.println("Modifier la salle avec id = 2");  
Salle salle = ss.findById(2);  
  
System.out.println("\tSalle à modifier : " + salle);  
  
Scanner sc = new Scanner(System.in);  
System.out.println("Donner le nouveau code :");  
salle.setCode(sc.nextLine());  
System.out.println("Donner le nouveau libelle :");  
salle.setLibelle(sc.nextLine());  
ss.update(salle);  
  
System.out.println("La liste des salles après les mises à jour :");  
for (Salle s : ss.findAll())  
    System.out.println("\t" + s);  
}  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Collections et la généricité



Solution : Gestion des salles

```
Run: Test x
La liste des salles :
  A Salle 1
  B Salle 2
  C Salle 3
  D Salle 4
Supprimer la salle avec id = 1
Modifier la salle avec id = 2
  Salle à modifier : B Salle 2
Donner le nouveau code :
C
Donner le nouveau libelle :
Salle 12
La liste des salles après les mises à jour :
  C Salle 12
  C Salle 3
  D Salle 4
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les exceptions



Saisie correcte

On veut écrire la fonction **saisieCorrecte** qui permet de saisir correctement un entier. Si l'utilisateur saisit une donnée dont le format n'est pas celui d'un entier, le programme lève l'exception **InputMismatchException**.

Question 1

La fonction devra traiter cette erreur en fournissant une solution alternative. Un message d'erreur sera affiché avec la proposition d'effectuer une nouvelle saisie.



Remarque

- La classe `InputMismatchException` appartient au package `java.util`.

Exemple d'exécution :

//Source : www.exelib.net

```
Donnez un entier :  
java  
Erreur de saisie  
Fin  
Donnez un entier :  
-7  
L'entier saisi est : -7  
Fin
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les exceptions



Solution : Saisie correcte

```
public class Question1 {
    public static void saisieCorrecte() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Donnez un entier : ");
        int n = sc.nextInt();
        System.out.println("L'entier saisi est : " + n);
    }
    public static void main(String[] args) {
        while (true) {
            try {
                saisieCorrecte();
                break;
            } catch (InputMismatchException e) {
                System.out.println("Erreur de saisi");
            } finally {
                System.out.println("Fin");
            }
        }
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les exceptions



Saisie correcte

Question 2

L'entier saisi doit être impérativement supérieur à 10.

On demande donc de créer une classe d'exception adaptée à cette erreur, puis de modifier le programme afin de traiter ce cas d'erreur.



Remarque

- On aurait pu utiliser l'exception prédéfinie `IllegalArgumentException` pour vérifier que l'entier saisi est supérieur à 10.

Exemple d'exécution :

```
Donnez un entier :  
java  
Erreur de saisi  
Donnez un entier :  
4  
valeur < 10  
Donnez un entier :  
34  
L'entier saisi est : 34
```


ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les exceptions



Solution 1 : Saisie correcte

```
public class Question2 {
    public static void saisieCorrecte() throws IllegalArgumentException {
        Scanner sc = new Scanner(System.in);
        System.out.println("Donnez un entier : ");
        int n = sc.nextInt();
        if (n < 10)
            throw new IllegalArgumentException("valeur < 10");
        System.out.println("L'entier saisi est : " + n);
    }
    public static void main(String[] args) {
        while (true) {
            try {
                saisieCorrecte();
                break;
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            } catch (InputMismatchException e) {
                System.out.println("Erreur de saisi");
            }
        }
    }
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les exceptions



Solution 2 : En utilisant une classe d'exception personnalisée

Classe d'exception personnalisée :

```
public class SaisiException extends Exception {  
    public SaisiException(String arg0) {  
        super(arg0);  
    }  
}
```

```
public class Question2_2 {  
    public static void saisieCorrecte() throws SaisiException {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Donnez un entier : ");  
        int n = sc.nextInt();  
        if (n < 10)  
            throw new SaisiException("valeur < 10");  
        System.out.println("L'entier saisi est : " + n);  
    }  
  
    public static void main(String[] args) {  
        while (true) {  
            try {  
                saisieCorrecte();  
            } catch (SaisiException e) {  
                System.out.println(e.getMessage());  
            } catch (InputMismatchException e) {  
                System.out.println("Erreur de saisi");  
            }  
        }  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



Exercice 1 : Lecture et écriture dans un fichier

Ecrire un programme en java permettant de lire les données stockées dans un fichier "file/**entree.txt**", et les copier dans un fichier de sortie "file/**sortie.txt**".

Salut tout le monde
Je suis un développeur mobile

entree.txt



Salut tout le monde
Je suis un développeur mobile

sortie.txt

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



WEBFORCE
BE THE CHANGE

Solution : Lecture et écriture dans un fichier

```
public static void main(String[] args) {  
    BufferedReader br = null;  
    PrintWriter pw = null;  
    try {  
        br = new BufferedReader(new FileReader("file/entree.in"));  
        pw = new PrintWriter(new FileWriter("file/sortie.out"));  
        String st = null;  
        while ((st = br.readLine()) != null)  
            pw.println(st);  
    } catch (FileNotFoundException e) {  
        System.out.println("Fichier introuvable");  
    } catch (IOException e) {  
        System.out.println("Impossible de lire les données");  
    } finally {  
        try {  
            if (br != null)  
                br.close();  
            if (pw != null)  
                pw.close();  
        } catch (IOException e) {  
            System.out.println("Impossible de fermer l'un des fichiers");  
        }  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



Exercice 2 : Somme de n lignes

Écrire un programme en Java permettant de lire les données à partir d'un fichier plat "som.in", la première ligne de ce fichier indique le nombre de lignes à lire, les lignes qui suivent contiennent deux nombres séparés par un espace. La somme de ces nombres sera stockée dans un fichier plat "som.out".

Exemple de fichier d'entrée :

```
5
12 2
4 5
4 -4
333 -44
4 5
43 4
2 6
3 4
```

Exemple de fichier de sortie :

```
14
9
0
289
9
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



WEBFORCE
BE THE CHANGE

Solution : Somme de n lignes

```
public static void main(String[] args) {  
    BufferedReader br = null;  
    PrintWriter pw = null;  
    try {  
        br = new BufferedReader(new FileReader("file/entree.in"));  
        pw = new PrintWriter(new FileWriter("file/sortie.out"));  
        String st = null;  
        while ((st = br.readLine()) != null)  
            pw.println(st);  
    } catch (FileNotFoundException e) {  
        System.out.println("Fichier introuvable");  
    } catch (IOException e) {  
        System.out.println("Impossible de lire les données");  
    } finally {  
        try {  
            if (br != null)  
                br.close();  
            if (pw != null)  
                pw.close();  
        } catch (IOException e) {  
            System.out.println("Impossible de fermer l'un des fichiers");  
        }  
    }  
}
```

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



Exercice 3 : Nombres opposés

Etant donné un tableau d'entiers non nuls, trouvez combien il y a d'entiers distincts positifs dont l'opposé est aussi dans le tableau.

Par exemple, pour le tableau de taille 15 qui suit :

-3 4 2 8 9 1 -3 -8 -4 2 8 2 -8 1 3

Il faut afficher 3. En effet, les trois entiers 3, 4, et 8 ont aussi leur opposé dans le tableau.

ENTRÉE :

- La première ligne de l'entrée contient un entier N , la taille du tableau ;
- La deuxième ligne contient N entiers séparés par des espaces : les éléments du tableau.

SORTIE :

Vous devez écrire une ligne sur la sortie, contenant un entier K : le nombre d'entiers distincts $X > 0$, tels que X et $-X$ appartiennent tous les deux au tableau.

EXEMPLE :

entrée :

15

-3 4 2 8 9 1 -3 -8 -4 2 8 2 -8 1 3

sortie :

3

ACTIVITÉ n° 1 – Créer des applications JAVA en appliquant les principes de la POO

Les entrées / sorties



WEBFORCE
BE THE CHANGE

Solution : Nombres opposés

```
public class NombresOpposes {
    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader("file/in.txt"));
        PrintWriter out = new PrintWriter(new FileWriter("file/out.txt"));
        int n = Integer.parseInt(in.readLine().toString());
        StringTokenizer st = new StringTokenizer(in.readLine(), " ");
        Set<Integer> set = new TreeSet<Integer>();
        Set<Integer> set2 = new TreeSet<Integer>();
        while (st.hasMoreElements()) {
            set.add(Integer.parseInt(st.nextElement().toString()));
        }
        int size1 = set.size();
        Iterator<Integer> it = set.iterator();
        while (it.hasNext()) {
            set2.add(Math.abs(it.next()));
        }
        out.println(size1 - set2.size());
        out.close();
        in.close();
    }
}
```


ACTIVITÉ n° 2

Réaliser des interfaces graphiques simples

Compétences visées :

- Réaliser une interface graphique simple avec du XML
- Lire des valeurs de l'interface utilisateur dans le code et les manipuler
- Utiliser Logcat dans Android Studio pour trouver des problèmes dans l'application ;
- Utiliser la communication basique entre les écrans

Recommandations clés :

- Bonne révision du résumé théorique



10 heures



1. Pour le formateur

- Guider les apprenant pour réaliser les différentes étapes de cette activité
- Cette activité est composée de deux parties distinctes

2. Pour l'apprenant

- Suivre les étapes indiquées dans l'activité

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Un ordinateur sur lequel est installée la dernière version stable d'Android Studio

4. Critères de réussite :

- Le stagiaires doit être capable de :
 - Créer des interfaces graphiques simples
 - Manipuler les données de l'interface graphique dans le code JAVA
 - Communiquer entre les interfaces graphiques



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Démarrer le projet

Consultez le calculateur de pourboires sur Google : <https://www.google.com/search?q=tip+calculator>

Bill	<input type="text" value="0.00"/>	Tip	\$0.00
Tip %	<input type="text" value="15%"/>	Total	\$0.00
Number of people	<input type="text" value="1"/>		

Dans cette activité, vous allez créer une version simple d'une calculatrice de pourboires sous forme d'application Android.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire

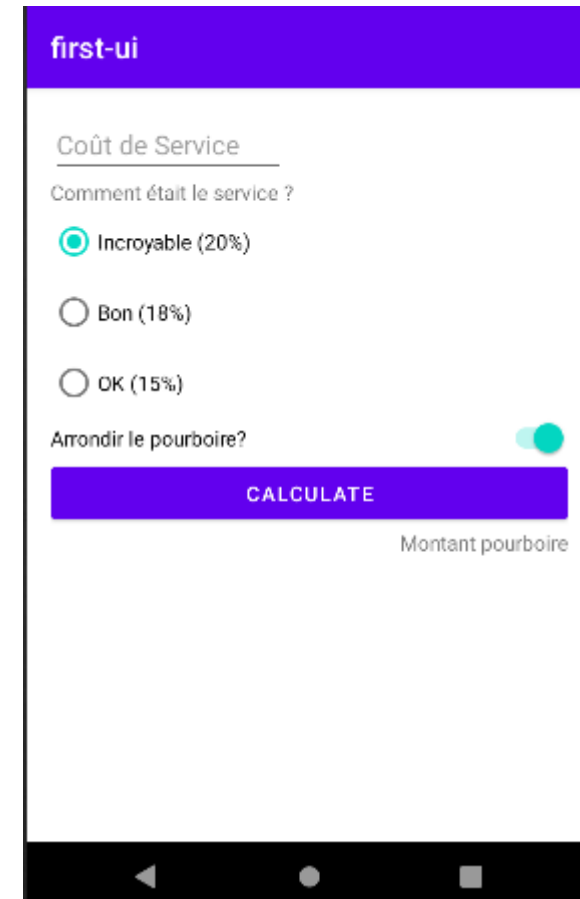


Démarrer le projet

À la fin de cette activité, votre application de calculatrice de pourboires ressemblera à ceci (à droite).

Vous utiliserez ces éléments d'interface utilisateur fournis par Android :

- **EditText** - pour saisir et modifier du texte ;
- **TextView** - pour afficher du texte comme la question sur le service et le montant du pourboire ;
- **RadioButton** - un bouton radio sélectionnable pour chaque option de pourboire ;
- **RadioGroup** - pour regrouper les options des boutons radio ;
- **Switch** – On/OFF pour choisir d'arrondir le pourboire ou non.



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Créer un projet avec une activité vide

1. Pour commencer, créez un nouveau projet Java dans Android Studio en utilisant le modèle Empty Activity.
2. Appelez l'application « First UI », avec un niveau d'API minimum de 21 (Lollipop). Le nom du package est ma.projet.ui.
3. Cliquez sur Terminer pour créer l'application.

The screenshot shows the 'New Project' dialog in Android Studio. The title bar reads 'New Project'. The main heading is 'Empty Activity'. Below it, the text says 'Creates a new empty activity'. The form contains the following fields and options:

- Name:** First UI
- Package name:** ma.projet.ui
- Save location:** C:\Users\Lachgar\AndroidStudioProjects\FirstUI
- Language:** Java
- Minimum SDK:** API 21: Android 5.0 (Lollipop)

Below the fields, there is an information icon and the text: 'Your app will run on approximately 98,6% of devices. [Help me choose](#)'. There is also an unchecked checkbox for 'Use legacy android.support libraries' with a question mark icon. Below this checkbox, it says 'Using legacy android.support libraries will prevent you from using'. At the bottom right, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Lire et comprendre le XML

- Au lieu d'utiliser l'éditeur de mise en page que vous connaissez déjà, vous allez construire la mise en page de votre application en modifiant le XML qui décrit l'interface utilisateur. Apprendre à comprendre et à modifier les dispositions de l'interface utilisateur à l'aide de XML sera important pour vous en tant que développeur Android.
- Vous allez examiner et modifier le fichier XML qui définit la disposition de l'interface utilisateur pour cette application. XML est l'acronyme de eXtensible Markup Language (langage de balisage extensible), qui est un moyen de décrire des données à l'aide d'un document textuel. Parce que le XML est extensible et très flexible, il est utilisé pour de nombreuses choses différentes, y compris pour définir la disposition de l'interface utilisateur des applications Android.

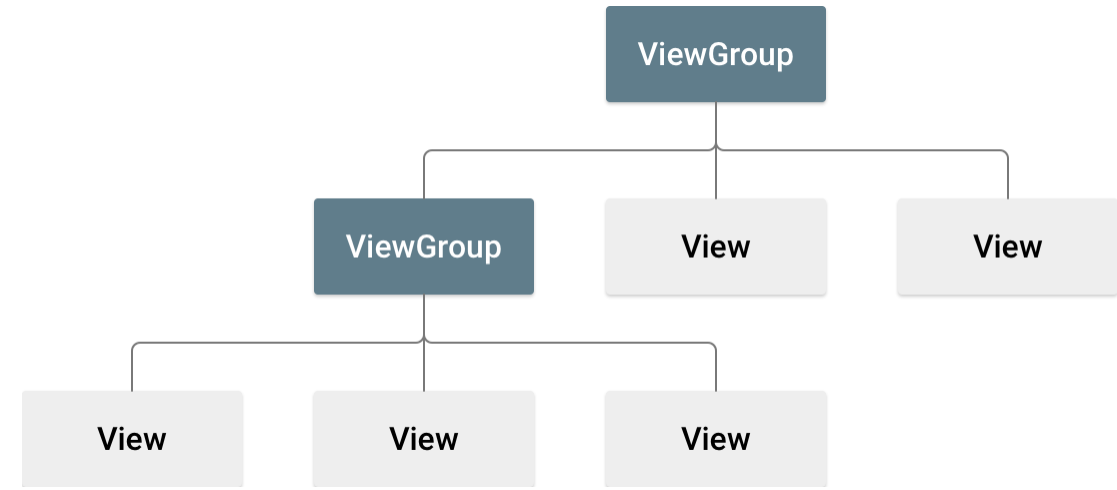
ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire

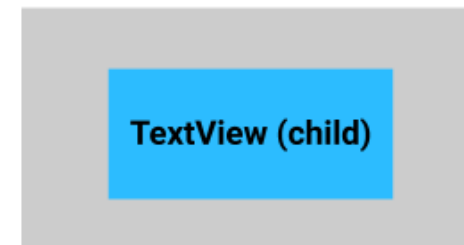


Lire et comprendre le XML

- L'interface utilisateur d'une application Android est construite comme une hiérarchie de composants (widgets) et les dispositions à l'écran de ces composants. Notez que ces mises en page sont elles-mêmes des composants de l'interface utilisateur.
- Vous décrivez la hiérarchie d'affichage des éléments de l'interface utilisateur à l'écran. Par exemple, un **ConstraintLayout** (le parent) peut contenir des boutons, des **TextViews**, des **ImageViews** ou d'autres vues (les enfants). N'oubliez pas que **ConstraintLayout** est une sous-classe de **ViewGroup**. Il vous permet de positionner ou de dimensionner les vues enfants de manière flexible.



ConstraintLayout (parent)



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

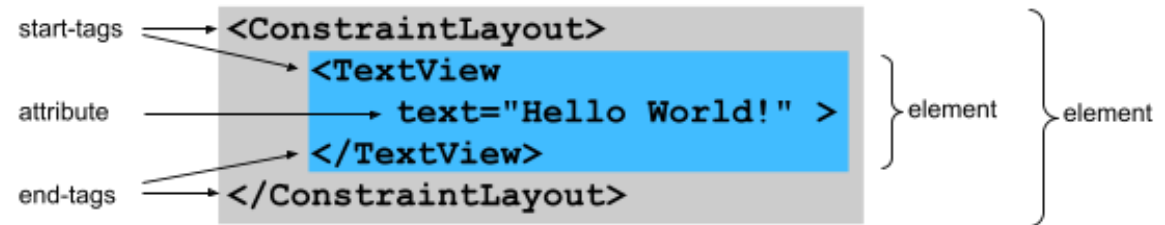
Application Android de calculateur de pourboire



Lire et comprendre le XML

Chaque élément de l'interface utilisateur est représenté par un élément XML dans le fichier XML. Chaque élément commence et se termine par une balise, et chaque balise commence par un < et se termine par un >. Tout comme vous pouvez définir des attributs sur les éléments de l'interface utilisateur à l'aide de l'éditeur de mise en page (design), les éléments XML peuvent également avoir des attributs. En simplifiant, le XML pour les éléments de l'interface utilisateur ci-dessus pourrait ressembler à quelque chose comme ceci :

```
<ConstraintLayout>
  <TextView
    text="Hello World!">
  </TextView>
</ConstraintLayout>
```



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

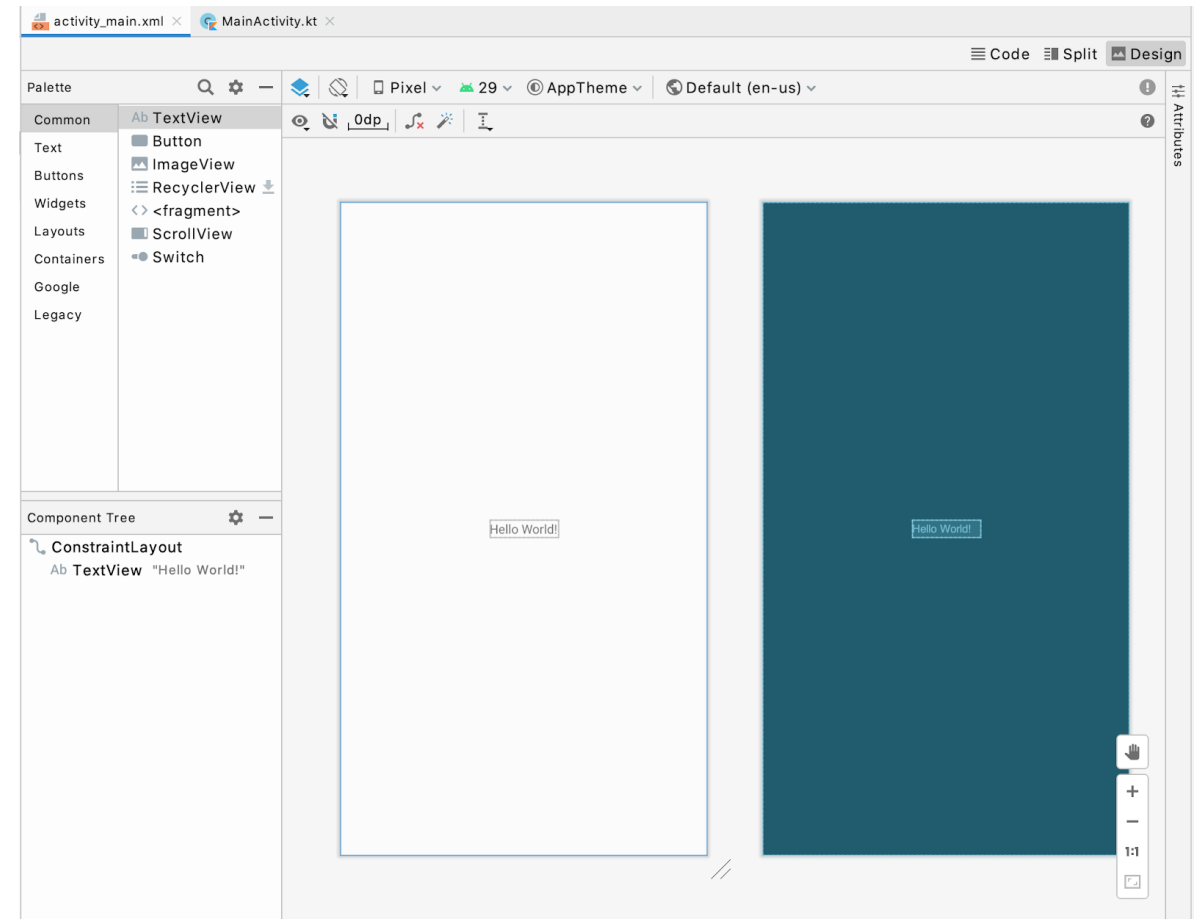
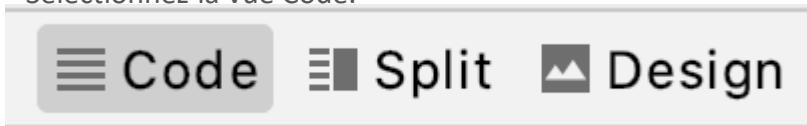
Application Android de calculateur de pourboire



Lire et comprendre le XML

Voyons un exemple réel :

1. Ouvrez activity_main.xml (res > layout > activity_main.xml) ;
2. Vous pouvez remarquer que l'application affiche un TextView avec "Hello World !" dans un ConstraintLayout, comme vous l'avez vu dans les projets précédents créés à partir de ce modèle ;
3. Trouvez les options des vues **Code**, **Split** et **Design** dans le coin supérieur droit de l'éditeur de mise en page ;
4. Sélectionnez la vue Code.



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Lire et comprendre le XML

Voici à quoi ressemble le XML dans activity_main.xml.

Il y a beaucoup plus de choses que dans l'exemple simplifié, mais Android Studio fait certaines choses pour aider à rendre le XML plus lisible, tout comme il le fait avec votre code Java.

5. Remarquez l'indentation. Android Studio le fait automatiquement pour vous montrer la hiérarchie des éléments. Le TextView est indenté car il est contenu dans le ConstraintLayout. Le ConstraintLayout est le parent, et le TextView est l'enfant. Les attributs de chaque élément sont indentés pour montrer qu'ils font partie de cet élément.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Lire et comprendre le XML

6. Remarquez le code couleur : certains éléments sont en bleu, d'autres en vert, et ainsi de suite. Les parties similaires du fichier sont dessinées dans la même couleur pour vous aider à les faire correspondre. En particulier, remarquez qu'Android Studio dessine le début et la fin des balises des éléments dans la même couleur.



Remarque

- Les couleurs utilisées dans l'activité peuvent ne pas correspondre à ce que vous voyez dans Android Studio

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Balises, éléments et attributs XML

Voici une version simplifiée de l'élément `TextView` afin que vous puissiez examiner certaines des parties importantes :

```
<TextView  
  android:text="Hello World!"  
>
```

La ligne avec `<TextView` est le début de la balise, et la ligne avec `>` est la fin de la balise. La ligne avec `android:text="Hello World !"` est un attribut de la balise. Elle représente le texte qui sera affiché par le `TextView`. Ces 3 lignes sont un raccourci couramment utilisé appelé balise `empty-element`. Elles auraient la même signification si vous les écriviez avec une balise de début et une balise de fin séparées, comme ceci :

```
<TextView  
  android:text="Hello World!"  
> </TextView>
```

Il est également courant avec une balise `empty-element` de l'écrire sur le moins de lignes possible et de combiner la fin de la balise avec la ligne qui la précède. Ainsi, vous pouvez voir une balise `empty-element` sur deux lignes (ou même une ligne si elle n'a pas d'attributs).

```
<!-- avec attributs, deux ligne -->  
<TextView  
  android:text="Hello World!" />
```

L'élément `ConstraintLayout` est écrit avec des balises de début et de fin distinctes, car il doit pouvoir contenir d'autres éléments. Voici une version simplifiée de l'élément `ConstraintLayout` avec l'élément `TextView` à l'intérieur :

```
<androidx.constraintlayout.widget.ConstraintLayout>  
  <TextView  
    android:text="Hello World!" />  
</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



En savoir plus sur le XML pour les layouts

Si vous vouliez ajouter une autre vue en tant qu'enfant du ConstraintLayout, comme un bouton sous le TextView, elle serait placée après la fin de la balise TextView.

```
<androidx.constraintlayout.widget.ConstraintLayout>
  <TextView
    android:text="Hello World!" />
  <Button
    android:text="Calculate" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

1. Regardez la balise du ConstraintLayout et remarquez qu'elle indique androidx.constraintlayout.widget.ConstraintLayout au lieu de ConstraintLayout comme pour le TextView. Cela s'explique par le fait que ConstraintLayout fait partie d'Android Jetpack, qui contient des bibliothèques de code offrant des fonctionnalités supplémentaires en plus de la plate-forme Android de base. Jetpack offre des fonctionnalités utiles dont vous pouvez tirer parti pour faciliter la création d'applications. Vous reconnaîtrez ce composant d'interface utilisateur comme faisant partie de Jetpack car il commence par "androidx" .

2. Vous avez peut-être remarqué les lignes qui commencent par xmlns ;, suivies de android, app, et tools.

```
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
```

Le xmlns signifie espace de noms XML, et chaque ligne définit un schéma, ou vocabulaire pour les attributs liés à ces mots. L'espace de noms android :, par exemple, marque les attributs qui sont définis par le système Android. Tous les attributs du Layout XML commencent par l'un de ces espaces de noms;

3. Les espaces entre les éléments XML ne changent pas la signification pour un ordinateur, mais ils peuvent aider à rendre le XML plus facile à lire.

Android Studio ajoute automatiquement des espaces et des retraits pour faciliter la lecture. Vous apprendrez plus tard comment faire pour qu'Android Studio s'assure que votre XML respecte les conventions de style de codage.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Balises, éléments et attributs XML

4. Vous pouvez ajouter des commentaires au XML, tout comme vous le feriez avec du code Java. Commencez par `<!--` et terminez par `-->`.

```
<!-- ceci est un commentaire en XML -->
```

```
<!-- ceci est un commentaire
```

```
sur plusieurs lignes
```

```
Et un autre
```

```
commentaire à plusieurs lignes -->
```

5. Notez la première ligne du fichier :

```
<?xml version="1.0" encoding="utf-8"?>
```

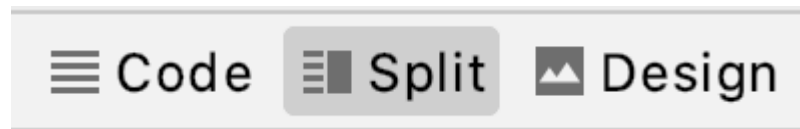
Cela indique que le fichier est un fichier XML, mais ce n'est pas le cas de tous les fichiers XML.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

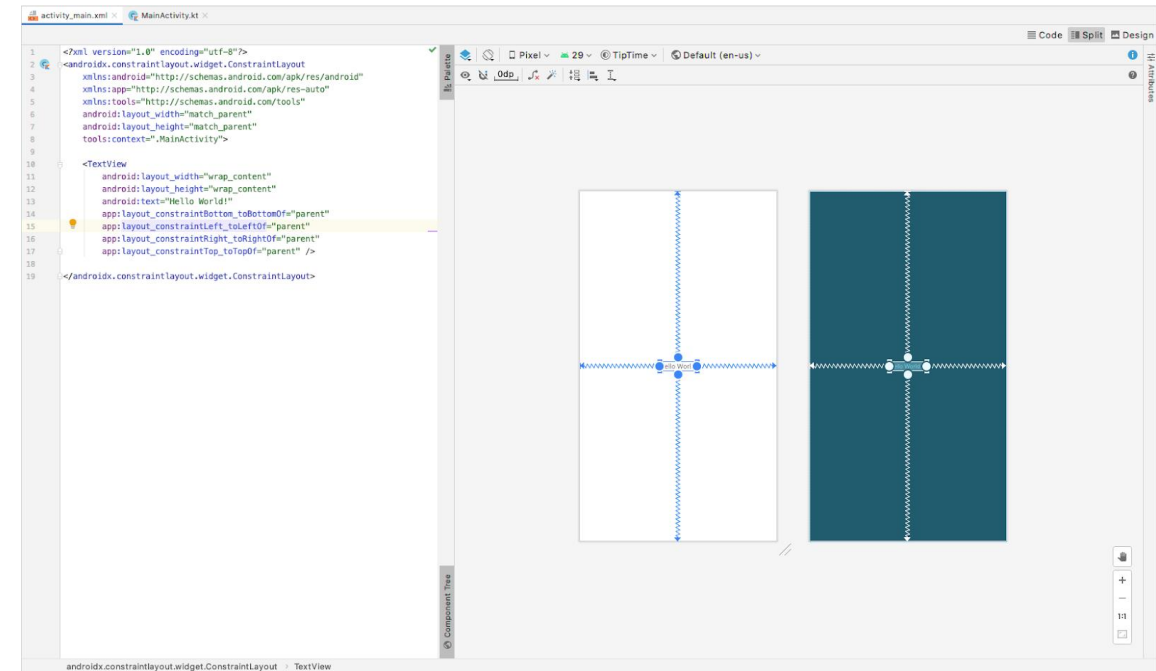
Application Android de calculateur de pourboire

Construire le Layout en XML

1. Toujours dans le fichier **activity_main.xml**, passez à la vue en écran partagé pour voir le XML à côté de l'éditeur de conception. L'éditeur de conception vous permet de prévisualiser la disposition de votre interface utilisateur ;



2. C'est à vous de voir quelle vue vous utilisez, mais pour cette activité, utilisez la vue fractionnée afin de pouvoir voir à la fois le XML que vous modifiez et les modifications apportées par ces modifications dans **l'éditeur de conception** ;
3. Essayez de cliquer sur différentes lignes, l'une sous le **ConstraintLayout**, puis l'autre sous le **TextView**, et remarquez que la vue correspondante est sélectionnée dans **l'éditeur de conception**. L'inverse fonctionne également - par exemple, si vous cliquez sur le **TextView** dans l'éditeur de conception, le XML correspondant est mis en surbrillance.



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Supprimer le TextView

1. Vous n'avez plus besoin du TextView maintenant, alors supprimez-le. Assurez-vous de supprimer tout ce qui se trouve entre le <TextView et />.

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    app:layout_constraintTop_toTopOf="parent" />
```

Tout ce qui reste dans le fichier est le ConstraintLayout :

```
<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```

2. Ajoutez 16dp de padding au ConstraintLayout afin que l'interface utilisateur ne soit pas écrasée par le bord de l'écran.

Le padding est similaire aux marges, mais il ajoute de l'espace à l'intérieur du ConstraintLayout, au lieu d'ajouter de l'espace à l'extérieur.

```
<androidx.constraintlayout.widget.ConstraintLayout  
    ...  
    android:padding="16dp"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```


ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter un champ de texte « coût de service »

Dans cette étape, vous allez ajouter l'élément d'interface utilisateur qui permettra à l'utilisateur de saisir le coût du service dans l'application. Vous utiliserez un élément EditText, qui permet à un utilisateur de saisir ou de modifier du texte dans une application.

1. Consultez la documentation relative à EditText et examinez l'exemple XML ;
2. Trouvez un espace vide entre les balises d'ouverture et de fermeture de la ConstraintLayout ;
3. Copiez et collez le XML de la documentation dans cet espace de votre disposition dans Android Studio.

Votre fichier Layout doit ressembler à ceci :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:padding="16dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/plain_text_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

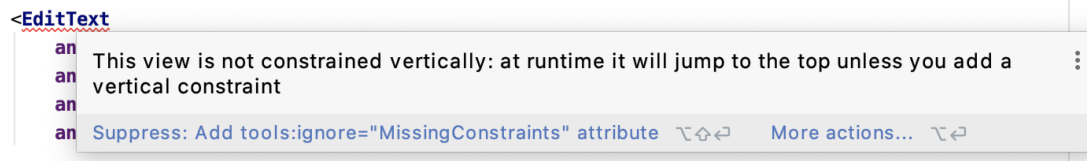
Application Android de calculateur de pourboire



Ajouter un champ de texte « coût de service »

Vous ne comprenez peut-être pas encore tout cela, mais cela sera expliqué dans les étapes suivantes.

- Remarquez que EditText est souligné en rouge ;
- Passez le pointeur dessus, et vous verrez une erreur "view is not constrained". Rappelez-vous que les enfants d'un ConstraintLayout ont besoin de contraintes pour que le layout sache comment les organiser ;



- Ajoutez ces contraintes à l'EditText pour l'ancrer dans le coin supérieur gauche du parent.

```
app:layout_constraintStart_toStartOf="parent"
```

```
app:layout_constraintTop_toTopOf="parent"
```

Si vous écrivez en anglais ou dans une autre langue qui s'écrit de gauche à droite (LTR), le bord de départ est la gauche. Mais certaines langues comme l'arabe s'écrivent de droite à gauche (RTL), et le bord de départ est donc à droite. C'est pourquoi la contrainte utilise "start", afin de pouvoir fonctionner avec les langues LTR ou RTL. De même, les contraintes utilisent "end" au lieu de "right".

Avec les nouvelles contraintes ajoutées, l'élément EditText ressemblera à ceci :

```
<EditText  
    android:id="@+id/plain_text_input"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    android:inputType="text"/>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

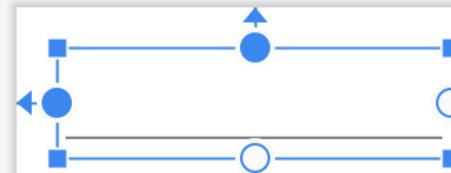
Application Android de calculateur de pourboire



Examiner les attributs de l'EditText

Vérifiez une nouvelle fois tous les attributs EditText que vous avez collés pour vous assurer qu'ils correspondent à l'utilisation que vous en ferez dans votre application.

1. Trouvez l'attribut id, qui est défini sur @+id/plain_text_input ;
2. Remplacez l'attribut id par un nom plus approprié, @+id/cout_de_service ;
3. Regardez l'attribut layout_height. Il est défini sur wrap_content, ce qui signifie que la hauteur sera égale à celle du contenu qu'il contient. Ce n'est pas grave, car il n'y aura qu'une seule ligne de texte ;
4. Regardez l'attribut layout_width. Il est défini sur match_parent, mais vous ne pouvez pas définir match_parent sur un enfant de ConstraintLayout. En outre, le champ de texte n'a pas besoin d'être aussi large. Définissez-le à une largeur fixe de 160dp, ce qui devrait être suffisant pour que l'utilisateur puisse saisir un coût de service ;



5. Remarquez l'attribut inputType - c'est une nouveauté. La valeur de l'attribut est "text", ce qui signifie que l'utilisateur peut saisir n'importe quel caractère dans le champ à l'écran (caractères alphabétiques, symboles, etc.).

```
android:inputType="text"
```

Cependant, vous souhaitez qu'ils ne saisissent que des chiffres dans l'EditText, car le champ représente une valeur monétaire.

5. Effacez le mot texte, mais laissez les guillemets ;
6. Commencez à taper le nombre à sa place. Après avoir tapé "n", Android Studio affiche une liste des compléments possibles qui incluent "n".

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Examiner les attributs de l'EditText

```
android:inputType="text" />  
roidx.constraintlayout.widget.ConstraintLayout>
```

8. Choisissez **numberDecimal**, qui les limite aux nombres avec un point décimal ;

```
android:inputType="numberDecimal"
```

Il y a encore une modification à apporter, car il est utile d'afficher une indication sur ce que l'utilisateur doit saisir dans ce champ.

9. Ajoutez un attribut d'indication à l'EditText décrivant ce que l'utilisateur doit saisir dans le champ.

```
android:hint="Coût de service"
```

Vous verrez également la mise à jour de l'éditeur de design.

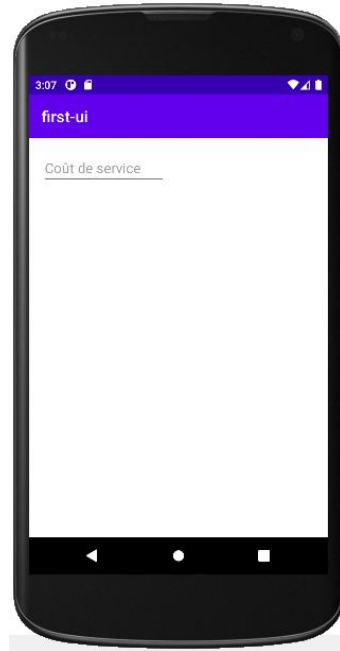
Coût de service

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire

Examiner les attributs de l'EditText

10. Exécutez votre application dans l'émulateur. Elle devrait ressembler à ceci :



Bon travail ! Il ne fait pas encore grand-chose, mais vous avez pris un bon départ et vous avez modifié un peu de XML. Le XML de votre mise en page devrait ressembler à quelque chose comme ceci.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/cost_of_service"
        android:layout_width="160dp"
        android:layout_height="wrap_content"
        android:hint="Coût de service"
        android:inputType="numberDecimal"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter la question de service

Dans cette étape, vous allez ajouter un TextView pour la question "Comment était le service ?". Essayez de taper ceci sans copier.

1. Après la fermeture de la balise EditText, />, ajoutez une nouvelle ligne et commencez à taper <TextView ;
2. Sélectionnez TextView dans les suggestions, et Android Studio ajoutera automatiquement les attributs layout_width et layout_height pour le TextView ;
3. Choisissez wrap_content pour les deux, puisque vous avez seulement besoin que le TextView soit aussi grand que le contenu textuel qu'il contient ;

<TextView

android:layout_width=""

android:layout_height: wrap_content

4. androidx.constraintlayout.w match_parent

```
<TextView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Comment était le service ?"
```

5. Fermez la balise avec /> ;
6. Remarquez dans l'éditeur de conception que le TextView chevauche l'EditText.

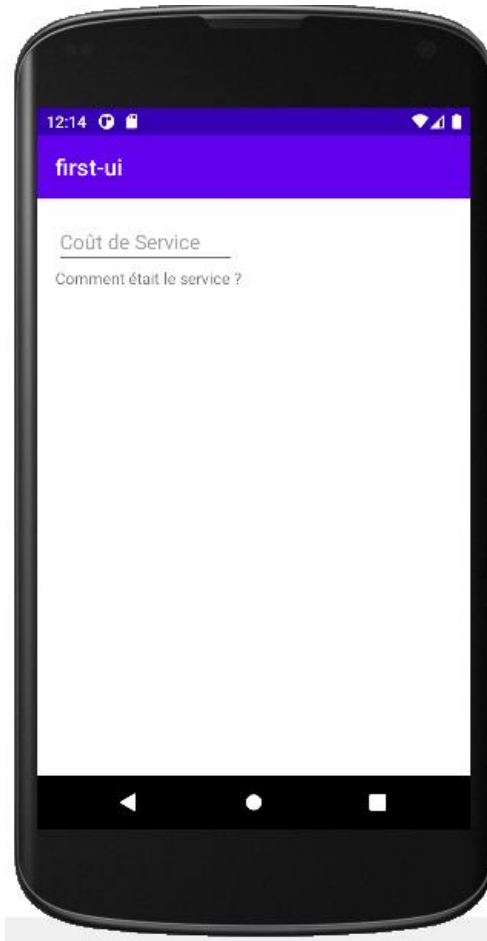
Cela ne semble pas correct, c'est pourquoi vous allez ajouter des contraintes sur le TextView. Réfléchissez aux contraintes dont vous avez besoin. Où le TextView doit-il être positionné horizontalement et verticalement ? Vous pouvez consulter la capture d'écran de l'application pour vous aider.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter la question de service



verticalement, vous voulez que le TextView se trouve sous le champ de texte du coût du service. Horizontalement, vous voulez que le TextView soit aligné sur le bord de départ du parent.

7. Ajoutez une contrainte horizontale au TextView pour contraindre son bord de départ au bord de départ du parent :

```
app:layout_constraintStart_toStartOf="parent"
```

8. Ajoutez une contrainte verticale au TextView pour contraindre le bord supérieur du TextView au bord inférieur de la vue du coût de service :

```
app:layout_constraintTop_toBottomOf="@id/cout_de_service"
```

Notez qu'il n'y a pas de plus dans @id.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter la question de service

9. Ajoutez un ID de ressource sur le TextView. Vous aurez besoin de vous référer à cette vue plus tard lorsque vous ajouterez d'autres vues et les contraindrez les unes aux autres.

```
android:id="@+id/question_service"
```

À ce stade, votre XML devrait ressembler à ceci.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/cout_de_service"
        android:hint="Coût de Service"
        android:layout_height="wrap_content"
        android:layout_width="160dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:inputType="numberDecimal"/>

    <TextView
        android:id="@+id/question_service"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comment était le service ?"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/cout_de_service"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```


ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter des options de pourboire

Ensuite, vous ajouterez des boutons radio pour les différentes options de conseil que l'utilisateur peut choisir.

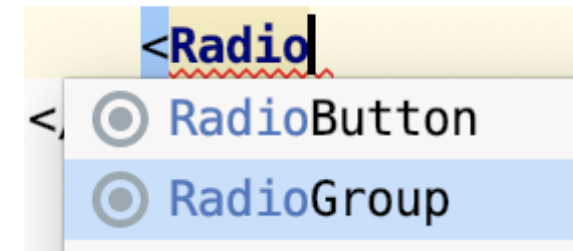
Il devrait y avoir trois options :

- Incroyable (20%) ;
- Bon (18%) ;
- Ok (15%).

1. Faites une recherche Google sur les boutons radio et Android. Le premier résultat est un guide du site des développeurs Android sur la façon d'utiliser les boutons radio - parfait ! ;
2. Parcourez le guide des boutons radio ;

Il y a un certain XML qui semble répondre à vos besoins. Lisez-le et voyez comment RadioGroup est la vue parent et les RadioButtons sont des vues (views) enfants à l'intérieur de celle-ci.

3. Retournez dans votre layout dans Android Studio pour ajouter le RadioGroup et le RadioButton à votre application :
4. Après l'élément TextView, mais toujours dans le ConstraintLayout, commencez à taper <RadioGroup. Android Studio vous fournira des suggestions utiles pour vous aider à compléter votre XML ;



5. Définissez les paramètres layout_width et layout_height du RadioGroup sur wrap_content ;
6. Ajoutez un ID de ressource défini sur @+id/options ;
7. Fermez la balise de début par >.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter des options de pourboire

- Android Studio ajoute `</RadioGroup>`. Comme le `ConstraintLayout`, l'élément `RadioGroup` aura d'autres éléments à l'intérieur, donc vous pourriez vouloir le déplacer sur sa propre ligne ;



- Contraintez le `RadioGroup` sous la question de service (verticalement) et au début du parent (horizontalement) ;
- Définissez l'attribut `android:orientation` sur `vertical`. Si vous vouliez que les boutons radio soient alignés, vous mettriez l'orientation à l'horizontale ;

Le XML pour le `RadioGroup` devrait ressembler à ceci :

```
<RadioGroup
    android:id="@+id/options"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/question_service">
```

```
</RadioGroup>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter des boutons radio

1. Après le dernier attribut du RadioGroup, mais avant la balise de fin `</RadioGroup>`, ajoutez un RadioButton ;
2. Définissez les paramètres `layout_width` et `layout_height` sur `wrap_content` ;
3. Attribuez l'ID de ressource `@+id/option_vingt_percent` au bouton radio ;
4. Définissez le texte sur Incroyable (20 %) ;
5. Fermez la balise avec `</>`.

```
<RadioGroup
    android:id="@+id/options"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/question_service">
```

```
<RadioButton
    android:id="@+id/option_vingt_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Incroyable (20%)" />
```

```
</RadioGroup>
```

Coût de Service
Comment était le service ?
 Incroyable (20%)

Maintenant que vous avez ajouté un RadioButton, pouvez-vous modifier le XML pour ajouter 2 autres boutons radio pour les options Bon (18%) et Ok (15%) ?

Voici à quoi ressemble le XML pour le RadioGroup et les RadioButtons :

```
<RadioButton
    android:id="@+id/option_vingt_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Incroyable (20%)" />
```

```
<RadioButton
    android:id="@+id/option_dix_height_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bon (18%)" />
```

```
<RadioButton
    android:id="@+id/option_quinze_percent"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="OK (15%)" />
```

Coût de Service
Comment était le service ?
 Incroyable (20%)
 Bon (18%)
 OK (15%)

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter un choix par défaut

Actuellement, aucune des options n'est sélectionnée. Il serait bon de pouvoir sélectionner par défaut l'une des options des boutons radio.

Il existe un attribut sur le **RadioGroup** qui permet de spécifier quel bouton doit être coché initialement. Il s'agit de **checkedButton**, que vous définissez comme l'ID de ressource du bouton radio que vous souhaitez sélectionner.

1. Sur le groupe radio, définissez l'attribut **android:checkedButton** sur **@id/option_vingt_percent**.

Remarquez dans l'éditeur de design que la mise en page a été mise à jour. L'option de pourboire de 20 % est sélectionnée par défaut, Maintenant, cela commence à ressembler à une calculatrice de pourboires !

Coût de Service

Comment était le service ?

Incroyable (20%)

Bon (18%)

OK (15%)

Ajoutez un switch pour arrondir le pourboire

Ensuite, vous utiliserez un widget Switch pour permettre à l'utilisateur de sélectionner oui ou non pour arrondir le pourboire.

Vous souhaitez que le Switch soit aussi large que le parent, et vous pourriez donc penser que la largeur devrait être définie sur `match_parent`. Comme indiqué précédemment, vous ne pouvez pas définir `match_parent` sur les éléments d'interface utilisateur dans un `ConstraintLayout`. Au lieu de cela, vous devez contraindre les bords de début et de fin de la vue, et définir la largeur à `Odp`. La définition de la largeur à `Odp` indique au système de ne pas calculer la largeur, mais d'essayer de correspondre aux contraintes de la vue.



Remarque

- Vous ne pouvez pas utiliser `match_parent` pour une vue (view) quelconque dans un `ConstraintLayout`. Utilisez plutôt `Odp`, ce qui signifie que les contraintes correspondent.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter un choix par défaut

1. Ajoutez un élément Switch après le XML de RadioGroup ;
2. Comme indiqué avant, définissez la valeur de `layout_width` sur `Odp` ;
3. Définissez le paramètre `layout_height` sur `wrap_content`. Ainsi, la vue de l'élément Switch sera aussi haute que le contenu qu'il contient ;
4. Définissez l'attribut `id` sur `@+id/round_up_switch` ;
5. Définissez l'attribut `text` sur `Round up tip ?`. Ce texte sera utilisé comme étiquette pour le commutateur ;
6. Contraintez le bord de départ du commutateur au bord de départ de `options` et la fin à la fin du parent ;
7. Contraintez le haut du commutateur au bas de l'option `options` ;
8. Fermez la balise avec `/>`.

Ce serait bien si l'interrupteur était activé par défaut, et il y a un attribut pour cela, **android:checked**, où les valeurs possibles sont `true` (on) ou `false` (off).

Ajoutez un switch pour arrondir le pourboire

9. Définissez l'attribut **android:checked** sur **true**.

En mettant tout cela ensemble, le XML pour l'élément Switch ressemble à ceci :

```
<Switch
  android:id="@+id/round_up_switch"
  android:layout_width="Odp"
  android:layout_height="wrap_content"
  android:checked="true"
  android:text="Arrondir le pourboire?"
  app:layout_constraintEnd_toEndOf="parent"
  app:layout_constraintStart_toStartOf="@id/options"
  app:layout_constraintTop_toBottomOf="@id/options" />
```

Coût de Service

Comment était le service ?

Incroyable (20%)

Bon (18%)

OK (15%)

Arrondir le pourboire?

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Ajouter le bouton Calculer

Ensuite, vous ajouterez un bouton pour que l'utilisateur puisse demander que le pourboire soit calculé. Vous voulez que le bouton soit aussi large que le parent, donc les contraintes horizontales et la largeur sont les mêmes que pour le commutateur.

1. Ajoutez un bouton après le Switch ;
2. Définissez la largeur à `Odp`, comme vous l'avez fait pour le Switch ;
3. Définissez la hauteur sur `wrap_content` ;
4. Donnez-lui un ID de ressource de `@+id/calculate_button`, avec le texte "Calculate" ;
5. Contraintez le bord supérieur du bouton au bord inférieur de l'astuce Round up ? Changez ;
6. Contraintez le bord de départ au bord de départ du parent et le bord de fin au bord de fin du parent ;
7. Fermez la balise avec `/>`.

Voici à quoi ressemble le XML du bouton Calculer

Coût de Service

Comment était le service ?

Incroyable (20%)

Bon (18%)

OK (15%)

Arrondir le pourboire?

CALCULATE

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



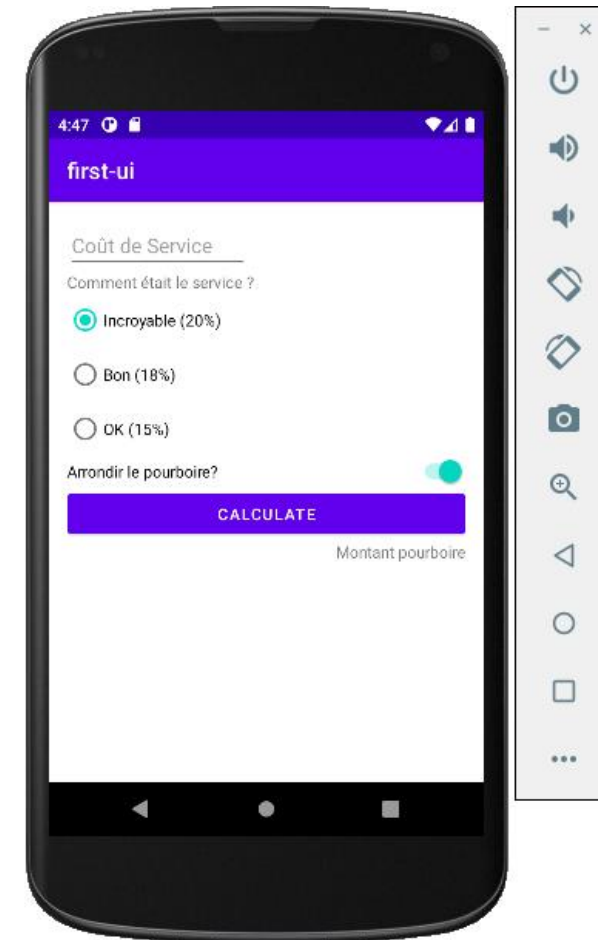
Ajouter le résultat

Vous avez presque terminé la mise en page ! Dans cette étape, vous allez ajouter un TextView pour le résultat du pourboire, en le plaçant sous le bouton Calculer, et en l'alignant à la fin plutôt qu'au début comme les autres éléments de l'interface utilisateur.

1. Ajoutez un TextView avec un ID de ressource nommé `tip_result` et le texte Montant pourboire ;
2. Contraintez le bord final de l'affichage de texte au bord final du parent ;
3. Faites coïncider le bord supérieur avec le bord inférieur du bouton Calculer.

<TextView

```
android:id="@+id/result"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintTop_toBottomOf="@id/calculate_button"  
android:text="Montant pourboire" />
```



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Calculer le montant de pourboire

1. Dans la classe MainActivity.java déclarer les variables suivantes :

```
private RadioGroup groupe;  
private EditText cout;  
private Button calculate;  
private TextView result;  
private Switch aSwitch;  
private double montant;
```

2. Dans la méthode *onCreate* initialiser les variables :

```
cout = findViewById(R.id.cout_de_service);  
groupe = findViewById(R.id.options);  
calculate = findViewById(R.id.calculate_button);  
aSwitch = findViewById(R.id.round_up_switch);  
result = findViewById(R.id.result);
```

À ce stade, votre Activité devrait ressembler à ceci.

```
public class MainActivity extends AppCompatActivity {  
  
    private RadioGroup groupe;  
    private EditText cout;  
    private Button calculate;  
    private TextView result;  
    private Switch aSwitch;  
    private double montant;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        cout = findViewById(R.id.cout_de_service);  
        groupe = findViewById(R.id.options);  
        calculate = findViewById(R.id.calculate_button);  
        aSwitch = findViewById(R.id.round_up_switch);  
        result = findViewById(R.id.result);  
    }  
}
```


ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Application Android de calculateur de pourboire



Calculer le montant de pourboire

3. Associer un évènement de clic au bouton :

```
calculate.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
    }  
});
```

4. Récupérer le pourcentage selon le radio bouton sélectionné :

```
int option = groupe.getCheckedRadioButtonId();  
double percent = 0;  
switch (option) {  
    case R.id.option_dix_height_percent:  
        percent = 0.18;  
        break;  
    case R.id.option_quinze_percent:  
        percent = 0.15;  
        break;  
    case R.id.option_vingt_percent:  
        percent = 0.2;  
        break;  
}
```

5. Vérifier si le switch est sélectionné et arrondir le montant (2 nombres après la virgule) :

```
montant = Double.parseDouble(cout.getText().toString()) * percent;  
  
if (aSwitch.isChecked()) {  
    montant = Math.round(montant * 100.0) / 100.0;  
}
```

6. Afficher les résultat :

```
result.setText("");  
result.append("Montant pourboire : " + montant);
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

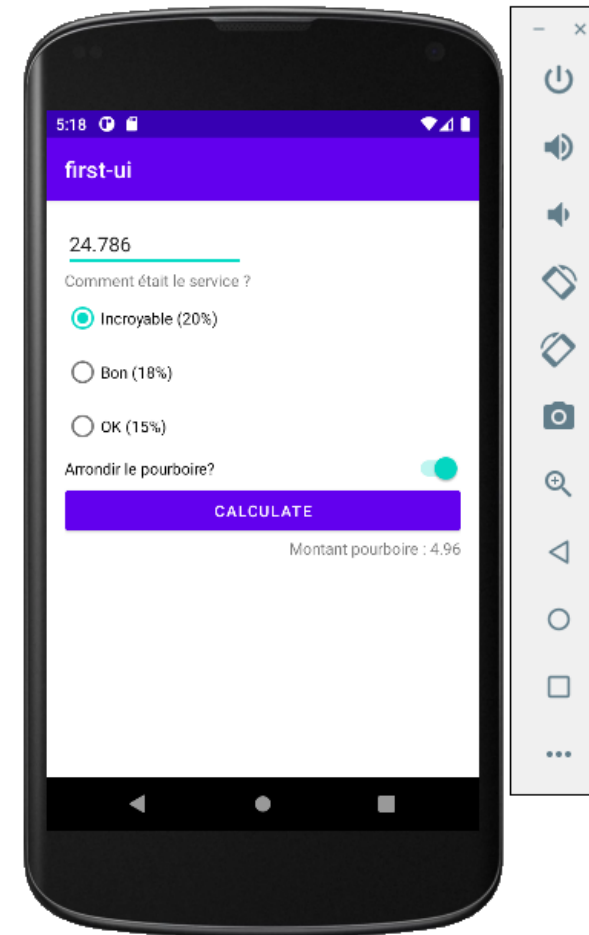
Application Android de calculateur de pourboire



Code de la méthode onCreate ()

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    cout = findViewById(R.id.cout_de_service);  
    groupe = findViewById(R.id.options);  
    calculate = findViewById(R.id.calculate_button);  
    aSwitch = findViewById(R.id.round_up_switch);  
    result = findViewById(R.id.result);  
    calculate.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            int option = groupe.getCheckedRadioButtonId();  
            double percent = 0;  
            switch (option) {  
                case R.id.option_dix_height_percent:  
                    percent = 0.18;  
                    break;  
                case R.id.option_quinze_percent:  
                    percent = 0.15;  
                    break;  
                case R.id.option_vingt_percent:  
                    percent = 0.2;  
                    break;  
            }  
            montant = Double.parseDouble(cout.getText().toString()) *  
percent;  
            if (aSwitch.isChecked()) {  
                montant = Math.round(montant * 100.0) / 100.0;  
            }  
            result.setText("");  
            result.append("Montant pourboire : " + montant);  
        }  
    });  
}
```

Résultat



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

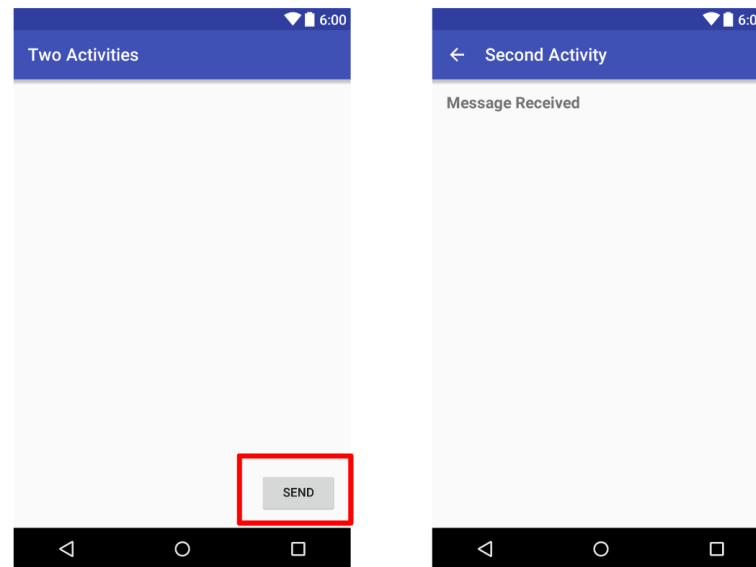
Activités et intents



Aperçu de l'application

Dans ce activité, vous créez et construisez une application appelée Deux activités (Two Activities). Vous construisez l'application en trois étapes.

Dans la première étape, vous créez une application dont l'activité principale contient un bouton, "Send". Lorsque l'utilisateur clique sur ce bouton, votre activité principale utilise un Intent pour démarrer la deuxième activité.



Main activity → Second activity

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 1 : Créer le projet TwoActivities

Dans cette étape, vous configurez le projet initial avec une activité (Activity) principale, définissez le layout et définissez une méthode pour l'événement onClick du bouton.

1. Démarrez Android Studio et créez un nouveau projet Android Studio ;

Nommez votre application "**Two Activities**" et choisissez les mêmes paramètres de Phone and Tablet que ceux que vous avez utilisés dans les travaux pratiques précédents. Le dossier du projet est automatiquement nommé **TwoActivities** et le nom de l'application qui apparaît dans la barre des applications sera "Two Activities".

2. Choisissez "**Empty Activity**" pour le modèle d'activité (Activity). Cliquez sur "Next« ;
3. Acceptez le nom d'activité (Activity) par défaut (MainActivity). Assurez-vous que les options Generate Layout file et Backwards Compatibility (AppCompat) sont cochées ;
4. Cliquez sur "Finish".

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 1 : Définir la mise en page (layout) de l'activité principale

1. Ouvrez **res> layout> activity_main.xml** dans le volet **Project> Android**. L'éditeur de disposition layout) apparaît ;
2. Cliquez sur l'onglet **Design** s'il n'est pas déjà sélectionné et supprimez le TextView (celui qui dit "Hello World") dans le volet **Component Tree** ;
3. Avec la connexion automatique (Autoconnect) activée (paramètre par défaut), faites glisser un bouton du volet **Palette** vers le coin inférieur droit de la présentation (layout). La connexion automatique (Autoconnect) crée des contraintes pour le bouton (Button) ;
4. Dans le volet **Attributes**, définissez l'**ID** sur **button_main** les propriétés **layout_width** et **layout_height** sur **wrap_content** et entrez **Send** pour le champ Text. La mise en page devrait maintenant ressembler à ceci:



TwoActivities



SEND

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 1 : Définir la mise en page (layout) de l'activité principale

5. Modifiez TextView par un Button et attribuer lui l'ID « button_main », comme suit :

```
<Button
    android:id="@+id/button_main"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginRight="16dp"
    android:text="SEND"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toRightOf="parent" />
```

TwoActivities

SEND



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 1 : Définir l'action du bouton

6. Dans la classe MainActivity.java, déclarer un objet Button, ensuite initialisez-le :

```
private static final String LOG_TAG = MainActivity.class.getSimpleName();  
private Button send;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    send = findViewById(R.id.button_main);  
}
```

7. Associez un évènement de clic au bouton, et affichez un message de journalisation :

```
send.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Log.d(LOG_TAG, "Button Clicked !");  
    }  
});
```



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 2 : Créer et lancer la 2^{ème} activité

Chaque nouvelle activité que vous ajoutez à votre projet a sa propre structure et ses propres fichiers Java, distincts de ceux de l'activité principale. Ils ont également leurs propres éléments <activity> dans le fichier AndroidManifest.xml. Comme pour l'activité principale, les nouvelles implémentations d'activité que vous créez dans Android Studio s'étendent également à partir de la classe AppCompatActivity.

1. Cliquez sur le dossier d'application de votre projet et choisissez **File > New > Activity > Empty Activity** ;
2. Nommez la nouvelle activité **SecondActivity**. Assurez-vous que les options **Generate Layout File** et **Backwards Compatibility (AppCompat)** sont cochées. Le nom de la présentation (layout) est nommé sous la forme activity_second. Ne cochez pas l'option **Launcher Activity** ;
3. Cliquez sur Finish. Android Studio ajoute à la fois une nouvelle layout d'activité (activity_second.xml) et un nouveau fichier Java (SecondActivity.java) à votre projet pour la nouvelle activité (Activity). Il met également à jour le fichier AndroidManifest.xml pour inclure la nouvelle activité.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 2 : Définir la mise en page (layout) pour la deuxième activité

1. Ouvrez `activity_second.xml` et cliquez sur l'onglet **Design** s'il n'est pas déjà sélectionné ;
2. Faites glisser un **TextView** du volet **Palette** vers le coin supérieur gauche de la disposition (layout) et ajoutez des contraintes aux côtés supérieur et gauche de la disposition (layout). Définissez ses attributs dans le volet **Attributes** comme suit :

Attribut	Valeur
id	text_header
Top margin	16
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	Message Received
textAppearance	AppCompat.Medium
textStyle	B (bold)

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



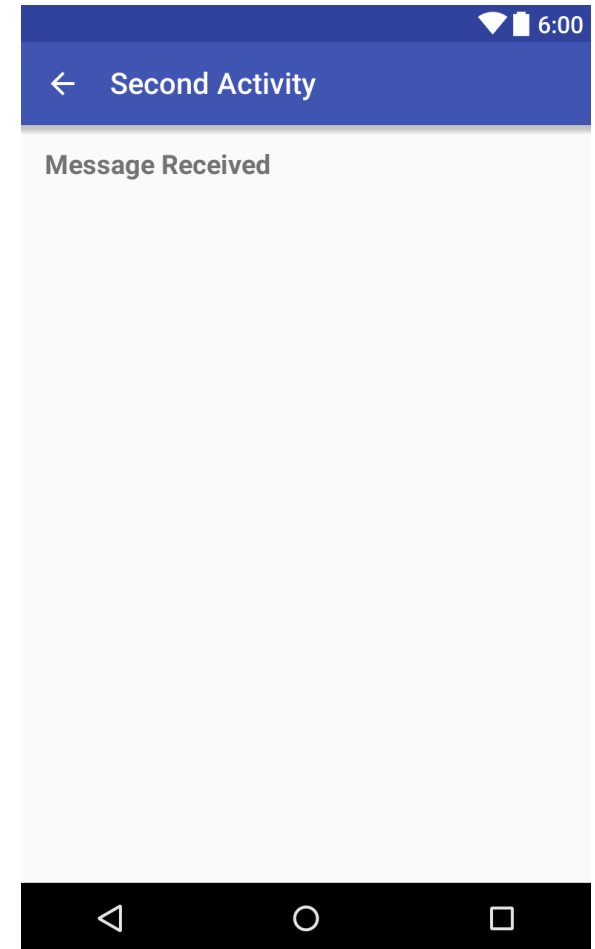
Etape 2 : Définir la mise en page (layout) pour la deuxième activité

La valeur de `textAppearance` est un attribut spécial du thème Android qui définit les styles de police de base.

La mise en page (layout) devrait maintenant ressembler à ceci :

- Ajoutez l'attribut `android:layout_marginLeft="8dp"` à `TextView` pour compléter l'attribut `layout_marginStart` pour les anciennes versions d'Android.

Le code XML pour `activity_second.xml` devrait être le suivant :



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 2 : Définir la mise en page (layout) pour la deuxième activité

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">
    <TextView
        android:id="@+id/text_header"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="Message Received"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 2 : Ajouter une intention (Intent) à l'activité principale

Dans cette étape, vous ajoutez une intention (Intent) explicite à l'activité (Activity) principale. Cette intention (Intent) est utilisée pour activer la deuxième activité lorsque l'utilisateur clique sur le bouton Send.

1. Ouvrez MainActivity ;
2. Créez une nouvelle intention (Intent) dans la méthode onClick() ;

Le constructeur d'intention (Intent) prend deux arguments pour une intention (Intent) explicite: un contexte (Context) d'application et le composant spécifique qui recevra cette intention (Intent). Ici, vous devez utiliser MainActivity.this en tant que contexte (Context) et SecondActivity.class en tant que classe spécifique:

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
```

3. Appelez la méthode startActivity() avec le nouvel Intent comme argument.

```
startActivity(intent);
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 2 : Ajouter une intention (Intent) à l'activité principale

4. Exécutez l'application.

Lorsque vous cliquez sur le bouton Send, MainActivity envoie l'intention (Intent) et le système Android lance SecondActivity, qui apparaît à l'écran. Pour revenir à MainActivity, cliquez sur le bouton Haut (Up la flèche gauche dans la barre d'application) ou sur le bouton Précédent (Back) en bas de l'écran.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Envoyer les données de l'activité principale à la deuxième activité

Dans la dernière étape, vous avez ajouté une intention explicite à MainActivity qui a lancé SecondActivity. Vous pouvez également utiliser une intention pour envoyer des données d'une activité à une autre lors de son lancement.

Votre objet d'intention peut transmettre des données à l'activité cible de deux manières: dans le champ de données ou dans les extras d'intention (intent extras). Les données d'intention sont un URI indiquant les données spécifiques sur lesquelles il faut agir. Si les informations que vous souhaitez transmettre à une activité par le biais d'une intention ne sont pas un URI ou si vous souhaitez envoyer plus d'une information, vous pouvez insérer ces informations supplémentaires dans les extras.

Les extras d'intention (Intent) sont des paires clé / valeur dans un Bundle. Un Bundle est un ensemble de données stockées sous forme de paires clé / valeur. Pour transmettre des informations d'une activité à une autre, vous devez insérer des clés et des valeurs dans le Bundle supplémentaire d'intention (Intent extras) à partir de l'activité d'envoi, puis les récupérer dans l'activité de réception.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Envoyer les données de l'activité principale à la deuxième activité

Dans cette étape, vous modifiez l'intention (Intent) explicite dans MainActivity afin d'inclure des données supplémentaires (dans ce cas, une chaîne de caractère entrée par l'utilisateur) dans un Bundle supplémentaire de l'intention (Intent). Vous modifiez ensuite SecondActivity pour extraire ces données du Bundle supplémentaire de l'intention et les afficher à l'écran.

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Ajouter un EditText à la mise en page (layout) de MainActivity

1. Ouvrez `activity_main.xml` ;
2. Faites glisser un élément **Texte brut** (EditText) du volet **Palette** vers le bas de la disposition (layout) et ajoutez des contraintes au côté gauche de la disposition (layout), au bas de la disposition (layout) et au côté gauche du bouton **Send**. Définissez ses attributs dans le volet **Attributs** comme suit :

Attribut	Valeur
id	editText_main
Right margin	8
Left margin	8
Bottom margin	16
layout_width	match_constraint
layout_height	wrap_content
inputType	textLongMessage
hint	Enter Your Message Here
text	(Supprimer tout texte dans ce champ)

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents

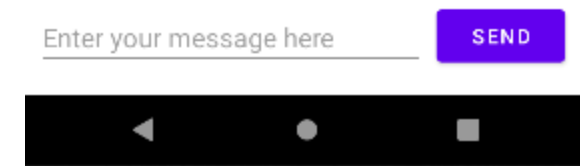


Etape 3 : Envoyer les données de l'activité principale à la deuxième activité

Le code XML de l'élément EditText est le suivant :

```
<EditText
    android:id="@+id/editText_main"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:ems="10"
    android:hint="Enter your message here"
    android:inputType="textLongMessage"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/button_main"
    app:layout_constraintStart_toStartOf="parent" />
```

La nouvelle présentation dans activity_main.xml ressemble à ceci:



ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Ajouter une chaîne aux extras d'intention (Intent)

1. Ouvrez **MainActivity** ;
2. Ajoutez une variable privée en haut de la classe pour contenir l'EditText :
`private EditText txtMessage ;`
3. Dans la méthode onCreate(), utilisez findViewById() pour obtenir une référence à EditText et l'affecter à cette variable privée :
`txtMessage = findViewById(R.id.editText_main) ;`
4. Dans la méthode onClick(), juste sous le nouvel Intent, obtenez le texte de EditText sous la forme d'une chaîne de caractère :
`String msg = txtMessage.getText().toString() ;`
5. Ajoutez cette chaîne à l'intention (Intent) en tant qu'extra avec la constante «message » en tant que clé et la chaîne en tant que valeur :

```
intent.putExtra("message", msg) ;
```

La méthode onCreate() dans MainActivity devrait maintenant ressembler à ceci :

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    txtMessage = findViewById(R.id.editText_main);  
    send = findViewById(R.id.button_main);  
    send.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            String msg = txtMessage.getText().toString();  
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);  
            intent.putExtra("message", msg);  
            startActivity(intent);  
        }  
    });  
}
```

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Ajouter un TextView à SecondActivity pour le message

1. Ouvrez `activity_second.xml` ;
2. Faites glisser un autre **TextView** vers la présentation (layout) sous `text_header`, puis ajoutez des contraintes à gauche et à la fin de `text_header` ;
3. Définissez les nouveaux attributs TextView dans le volet **Attributes** comme suit :

Attribut	Valeur
id	text_message
Top margin	8
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	(Supprimer tout texte dans ce champ)
textAppearance	AppCompat.Medium

ACTIVITÉ n° 2 – Réaliser des interfaces graphiques simples

Activités et intents



Etape 3 : Ajouter un TextView à SecondActivity pour le message

La nouvelle présentation (layout) a la même apparence que dans la tâche précédente, car la nouvelle TextView ne contient pas (encore) de texte et n'apparaît donc pas à l'écran.

Le code XML de TextView devrait ressembler à ceci :

```
<TextView
    android:id="@+id/text_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/text_header" />
```

Etape 3 : Modifiez SecondActivity pour obtenir les extras et afficher le message

1. Ouvrez **SecondActivity** pour ajouter du code à la méthode onCreate().
2. Obtenez l'intention (Intent) qui a activé cette activité (Activity) :

```
Intent intent = getIntent();
```

3. Obtenez la chaîne contenant le message des extras d'intention (Intent) en utilisant la clé « message » en tant que clé :

```
String message = intent.getStringExtra("message");
```

4. Utilisez findViewById() pour obtenir une référence à TextView pour le message à partir de la présentation (layout) :

```
TextView textView = findViewById(R.id.text_message);
```

5. Définissez le texte de TextView sur la chaîne extraite de l'intention (Intent):

```
textView.setText(message);
```

6. Exécutez l'application. Lorsque vous tapez un message dans MainActivity et cliquez sur Send, SecondActivity démarre et affiche le message.

La méthode onCreate() de SecondActivity devrait ressembler à ceci:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_second);  
    Intent intent = getIntent();  
    String message = intent.getStringExtra("message");  
    TextView textView = findViewById(R.id.text_message);  
    textView.setText(message);  
}
```

ACTIVITÉ n° 3

Utiliser Git pour manipuler le code des deux applications créées

Compétences visées :

- Manipuler des repository
- Utiliser des commandes Git
- Utiliser Git dans Android Studio

Recommandations clés :

- Révision générale du résumé théorique



8 heures



1. Pour le formateur :

- Rappeler les commandes Git
- Constituer des groupes de travail

2. Pour l'apprenant :

- Installer Git
- Créer un compte dans GitHub
- Collaborer sur les projets à distance

3. Conditions de réalisation :

- Installer Git
- Utiliser Git Bash ou Git CMD
- Utiliser Android Studio

4. Critères de réussite :

- Le stagiaire doit être capable :
 - de manipuler les commandes Git
 - de collaborer sur un projet à distance



ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

Commandes GiT



Git

Rappel :

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. En 2016, il s'agit du logiciel de gestion de versions le plus populaire qui est utilisé par plus de douze millions de personnes.

Lors cette activité, nous allons nous initier à l'utilisation du Git et puis à la maintenance d'un répertoire de travail de manière structurée et ordonnée.

ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

Commandes GiT



Premiers pas avec Git

- Git est un logiciel de contrôle de versions, il permet de sauvegarder l'historique du contenu d'un répertoire de travail. Pour ce faire l'utilisateur doit régulièrement enregistrer (en créant une révision ou commit) les modifications apportées au répertoire, il pourra ensuite accéder à l'historique de toutes les modifications et inspecter l'état du dossier à chaque révision.
- Git a la particularité de permettre de créer une copie d'un répertoire de travail, working copy, et de synchroniser entre eux plusieurs copies du même répertoire, permettant la décentralisation du travail.
- De plus, Git permet d'utiliser une ou plusieurs branches de développement et de fusionner entre elles ces branches.

ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

Commandes GiT



Création d'un nouveau dépôt

Nous allons d'abord nous intéresser à l'aspect gestionnaire de versions de Git : comment enregistrer l'historique des modifications apportées à un projet. Pour obtenir un dépôt Git sur lequel travailler, deux options sont possibles :

1. **Création d'un dépôt vide** (typiquement utilisé pour commencer un nouveau projet de développement) ;
2. **Copie** (clone dans le langage de Git) d'un dépôt existant pour travailler sur cette copie de travail (typiquement utilisé pour collaborer avec les développeurs d'un projet en cours).

Examinons la première option. Git a plusieurs interfaces utilisateur. La plus complète étant l'interface en ligne de commande (CLI), nous nous servons de celle-ci.

Pour créer un nouveau dépôt, on utilise la commande **git init first-ui**. Cette commande initialise un dépôt Git dans le répertoire first-ui (celui-ci est créé s'il n'existe pas). Ce répertoire contient alors à la fois une version de travail (dans first-ui) et un dépôt Git (dans first-ui/.git).

ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

Commandes GiT



Création d'un nouveau dépôt

Initialiser un nouveau dépôt Git dans le répertoire **first-ui** de l'application mobile de **calculateur de pourboire** :

1. Ouvrir une ligne de commande (Win+R et cmd) ;
 2. Naviguer et se mettre à l'intérieur du répertoire du projet :
 - *cd.. pour remonter*
 - *cd nomDuDossier pour descendre*
 - *ls ou dir pour afficher la liste du contenu*
- ```
> cd first-ui
```

3. Initialiser le nouveau dépôt Git :

```
> git init
```

un dépôt Git (dans first-ui/.git) sera créé. Afficher les fichiers cachés pour visualiser le dossier .git.

Cela ajoute les trois arbres nécessaire au suivi de version:

- le répertoire de travail (la version actuelle): working directory
- l'index : la liste des choses qui sont préparées (la commande pour ajouter au « stage » est « add » )
- le head qui correspond a ce qui a été « commit », c'est à dire vraiment envoyé comme version

## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT

### Add et Commit

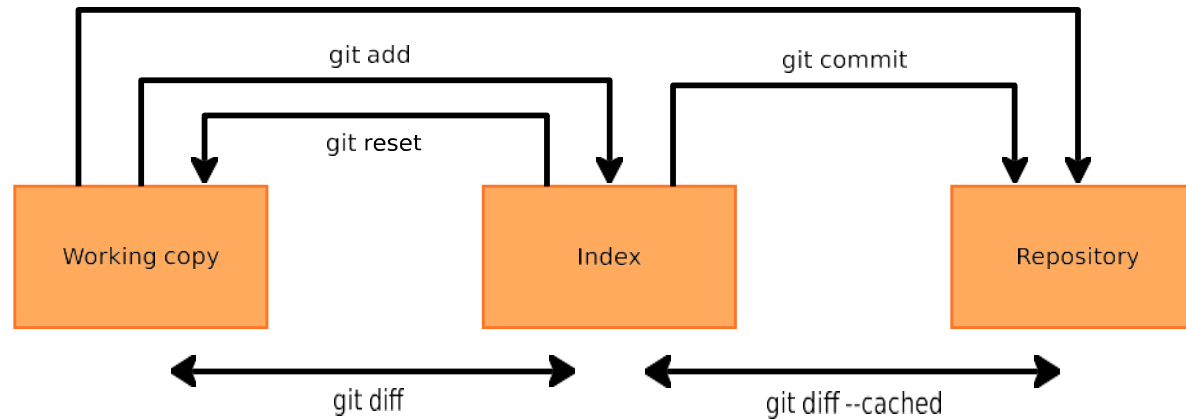


Figure 1 : git add commit *workflow*.

Pour être intégrée dans l'historique des révisions du dépôt (pour être "commitée"), chaque modification doit suivre le *workflow* montré dans Figure 1:

- la modification est d'abord effectuée sur la copie de travail ;
- elle est ensuite mémorisée dans une aire temporaire nommée **index**, avec la commande **git add** ;
- enfin, ce qui a été placé dans l'index peut être "commité" avec la commande **git commit**.

**git diff**, selon les paramètres d'appel peut être utilisé pour observer les différences entre les états dans la Figure 1; le format d'affichage est le même de la command `diff -u`.

## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT



#### Add et Commit

4. Vérifiez avec **git status** l'état dans lequel se trouve votre dépôt. Vos modifications devraient être présentes seulement dans la copie de travail ;
5. Préparez le projet pour le commit avec « **git add .** », Utilisez **git status** à nouveau pour vérifier que les modifications ont bien été placées dans l'index.  
Puis, utilisez **git diff --cached** pour observer les différences entre l'index et la dernière version présente dans l'historique de révision (qui est vide) ;
6. Commitez votre modification avec **git commit -m "<votre\_message\_de\_commit>"**. Le message entre guillemets doubles décrira la nature de votre modification (généralement  $\leq 65$  caractères) ;
7. Exécutez à nouveau **git status**, pour vérifier que vos modifications ont bien été comitées ;
8. Essayez à présent la commande **git log** pour afficher la liste des changements effectués dans ce dépôt ; combien y en a-t-il ? Quel est le numéro (un hash cryptographique en format SHA1) du dernier commit effectué ?

## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT



#### Add et Commit

9. Créez quelques autres activités dans le projet. . .et/ou modifiez l'activité MainActivity déjà créée, en commitant chaque modification séparément. Chaque commit doit contenir une et une seule création ou modification de fichier. Effectuez au moins 5 modifications différentes (et donc 5 commits différents).

À chaque étape essayez les commandes suivantes :

- **git diff** avant **git add** pour observer ce que vous allez ajouter à l'index.
- **git diff --cached** après **git add** pour observer ce que vous allez committer.

**Note** : la commande `git commit <file>` a le même effet que `git add <file>` suivie de `git commit`.

10. Regardez à nouveau l'historique des modifications avec **git log** et vérifiez avec **git status** que vous avez tout commité. Git offre plusieurs interfaces, graphiques ou non, pour afficher l'historique. Essayez les commandes suivantes (`gitg` et `gitk` ne sont pas forcément installés) :

- **git log**
- **git log --graph --pretty=short**
- **gitg**
- **gitk**

## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT



### Voyage dans le temps

11. Vous voulez changer d'avis entre les différents états de la Figure [1](#)? Faites une modification d'une ou plusieurs activités, ajoutez-la à l'index avec **git add** (vérifiez cet ajout avec **git status**), mais ne la committez pas. Exécutez **git reset** sur le nom de fichier (ou les noms de fichiers) que vous avez préparés pour le commit ; vérifiez avec **git status** le résultat.
12. Votre modification a été « retirée » de l'index. Vous pouvez maintenant la jeter à la poubelle avec la commande **git checkout** sur le ou les noms des fichiers modifiés, qui récupère dans l'historique leurs versions correspondant au tout dernier commit. Essayez cette commande, et vérifiez avec **git status** qu'il n'y a maintenant plus aucune modification à commiter.



#### Remarque

- **git checkout** est une commande très puissante. Elle vous permet de voyager entre différentes branches (voir plus loin) et aussi de revenir temporairement à une version précédente de votre copie de travail.

## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT



### Voyage dans le temps

13. Regardez l'historique de votre dépôt avec **git log** ; choisissez dans la liste un commit (autre que le dernier). Exécutez **git checkout COMMITID** où COMMITID est le numéro de commit que vous avez choisi. Vérifiez que l'état de vos fichiers est maintenant revenu en arrière, au moment du commit choisi. Que dit maintenant **git status** ?

**git log** n'affiche plus les commits postérieurs à l'état actuel, sauf si vous ajoutez l'option --all.

Attention, avec **git checkout** les fichiers de votre copie de travail sont modifiés directement par Git pour les remettre dans l'état que vous avez demandé. Si les fichiers modifiés sont ouverts par d'autres programmes (e.g. un éditeur de texte comme Emacs), il faudra les réouvrir pour observer les modifications.

14. Vous pouvez retourner à la version plus récente de votre dépôt avec **git checkout master**. Vérifiez que cela est bien le cas. Que dit maintenant **git status** ?



# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

## Commandes GiT



### Envoi des données sur le serveur de GitHub

15. Créez le projet sur **GitHub.com**, en ne mettant pas de Readme. Il vaut mieux l'ajouter après, une fois que les fichiers ont été uploader, pour éviter tout conflit ;
16. Créer le repository et copier le lien situé généralement en haut de la page dans une boîte bien visible: le lien se termine par « .git » :

**`https://github.com/lachgar/first-ui.git`**

17. Dans la ligne de commande, désigner le répertoire distant comme cible du projet :

**`git remote add origin https://github.com/lachgar/first-ui.git`**

18. Envoyer les fichiers vers le repository distant :


**`git push -u origin master`**

19. Vérifier si le projet figure maintenant dans le repository distant.

### Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*      Repository name \*

 lachgar ▾ / first-ui ✓

Great repository names are short and memorable. Need inspiration? How about [verbose-fortnight?](#)

Description (optional)

-  **Public**  
Anyone on the internet can see this repository. You choose who can commit.
-  **Private**  
You choose who can see and commit to this repository.

# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

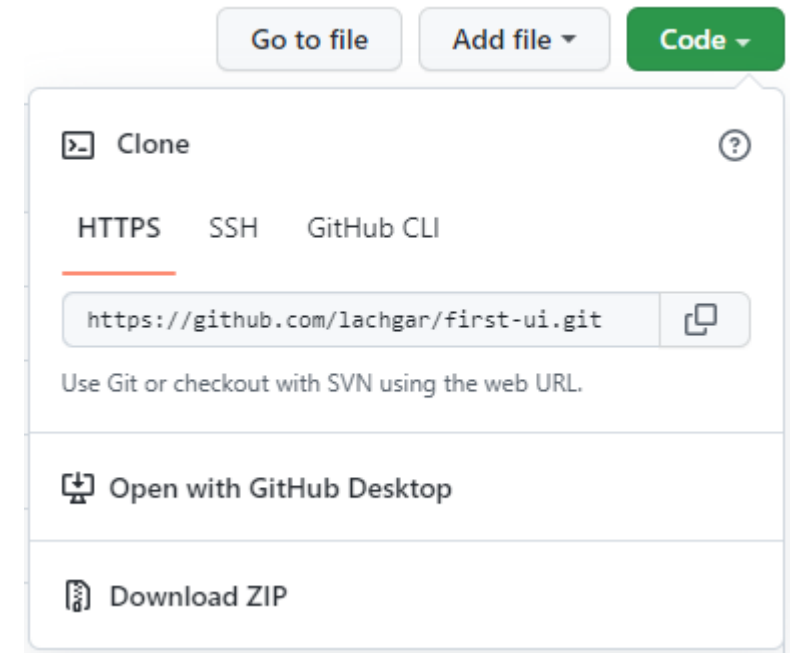
## Commandes GiT



### Cloner un repository

Maintenant demander à vos collègues de récupérer le projet depuis GitHub.

20. Sur GitHub.com, accédez à la page principale du repository ;
21. Au-dessus de la liste des fichiers, cliquez sur ↓Code ;
22. Copiez l'URL du repository.
  - Pour cloner le référentiel en utilisant HTTPS, sous "HTTPS", cliquez sur
  - Pour cloner le référentiel à l'aide d'une clé SSH, y compris un certificat émis par l'autorité de certification SSH de votre organisation, cliquez sur SSH, puis sur
  - Pour cloner un référentiel à l'aide de GitHub CLI, cliquez sur GitHub CLI, puis sur



## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Commandes GiT



### Cloner un repository

23. Ouvrez Git Bash ;
24. Changez le répertoire de travail actuel par l'emplacement où vous voulez cloner le projet ;
25. Tapez git clone, puis collez l'URL que vous avez copiée précédemment ;

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

26. Appuyez sur Entrée pour créer votre clone local ;
27. Importer le projet dans Android Studio.

# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

## Utiliser Git dans Android Studio



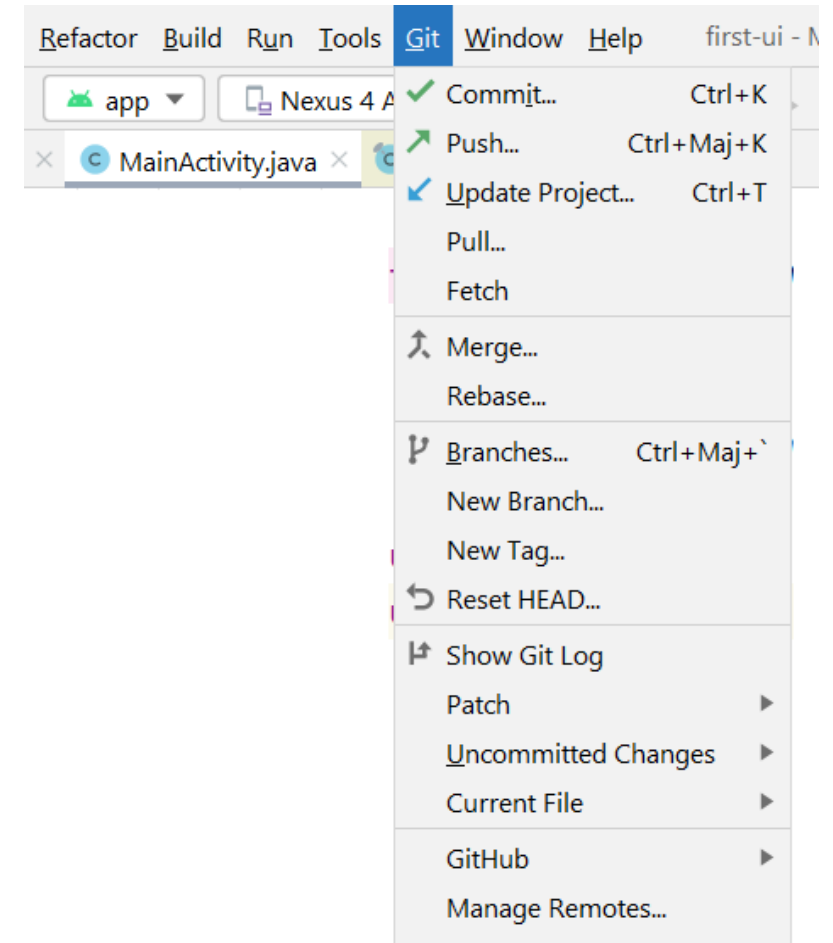
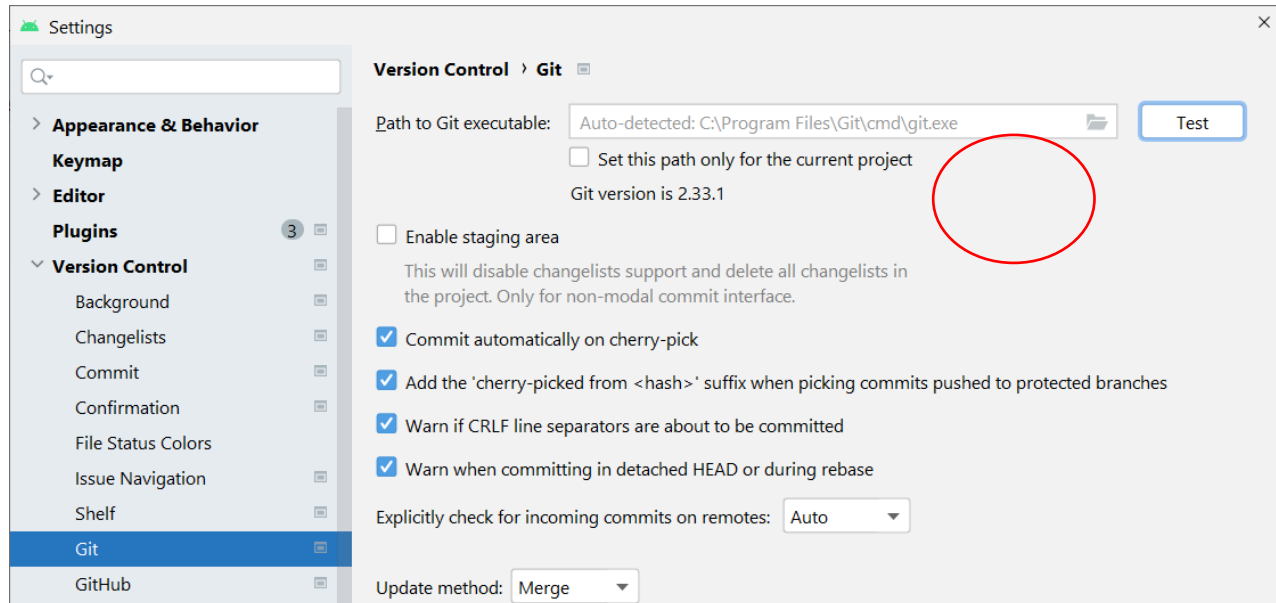
### Utiliser Git dans Android Studio

1. Vérifier si Git est configuré

Dans Android Studio, accédez à **Android Studio > Préférences >**

**Contrôle de version > Git**. Cliquez sur Tester pour vous assurer que Git

est configuré correctement dans Android Studio.



# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

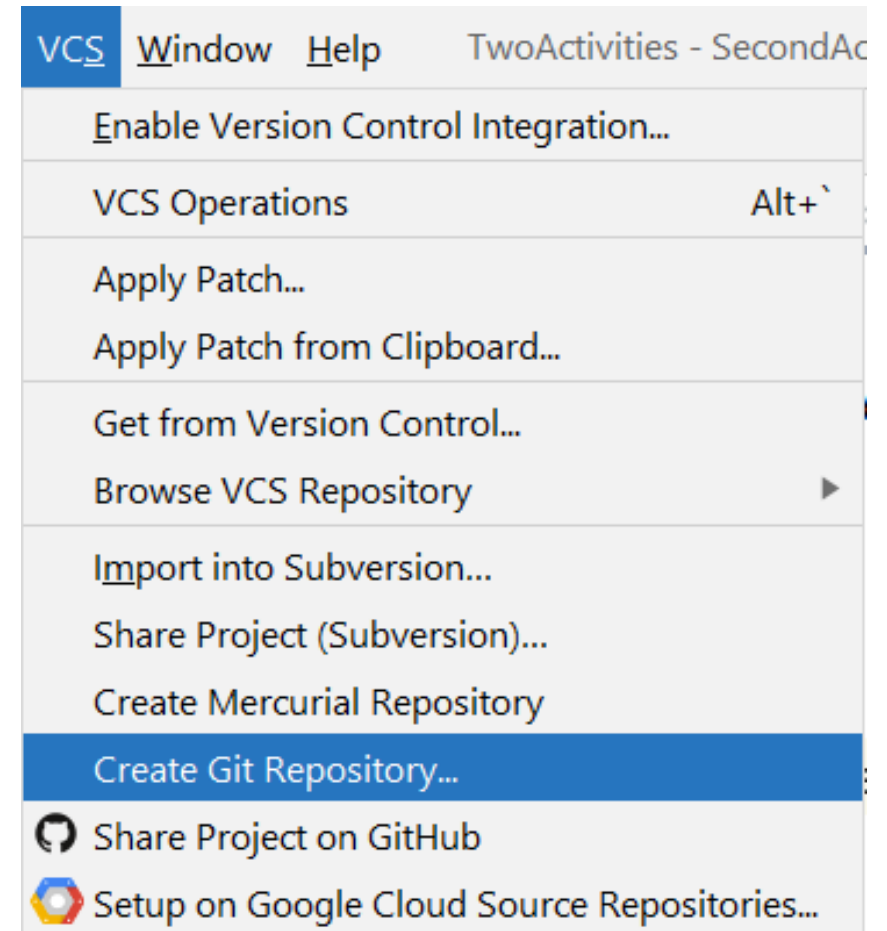
## Utiliser Git dans Android Studio



### Utiliser Git dans Android Studio

Nous allons intégrer Git dans le 2<sup>ème</sup> projet mobile à l'aide d'Android Studio (Activités et intents).

2. Une fois que votre projet Android Studio a été ouvert, cliquez sur le menu VCS, survolez le menu Importer dans le contrôle de version, et sélectionnez Créer un dépôt Git...



## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

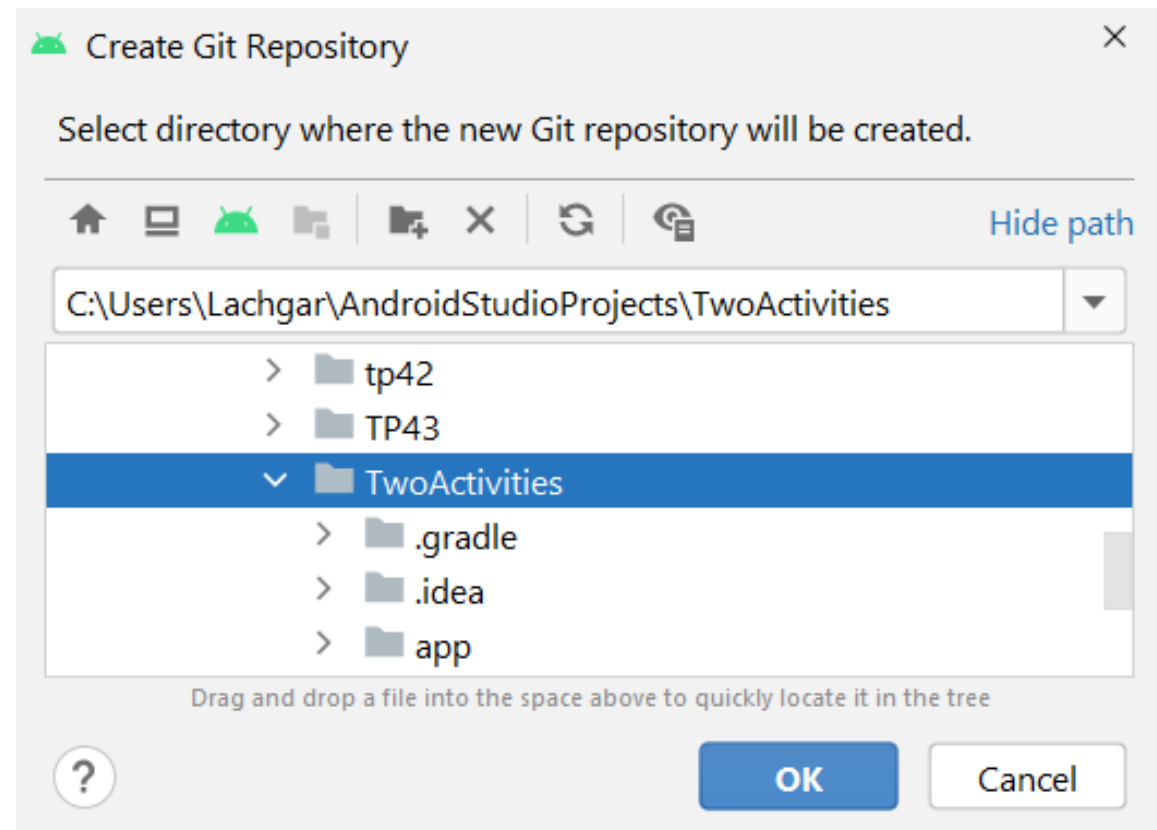
### Utiliser Git dans Android Studio



### Utiliser Git dans Android Studio

- Sélectionnez ensuite le dossier parent supérieur de votre projet Android Studio.

Cliquez sur le bouton OK pour initialiser le projet avec Git., en arrière plan Android Studio exécute la commande Git : git init.



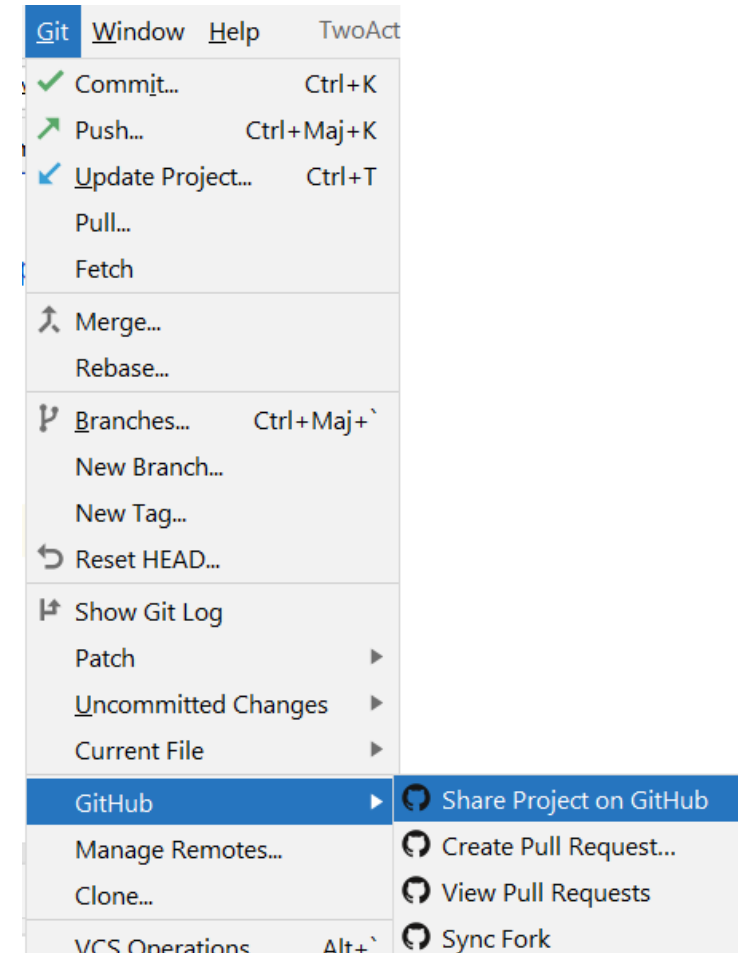
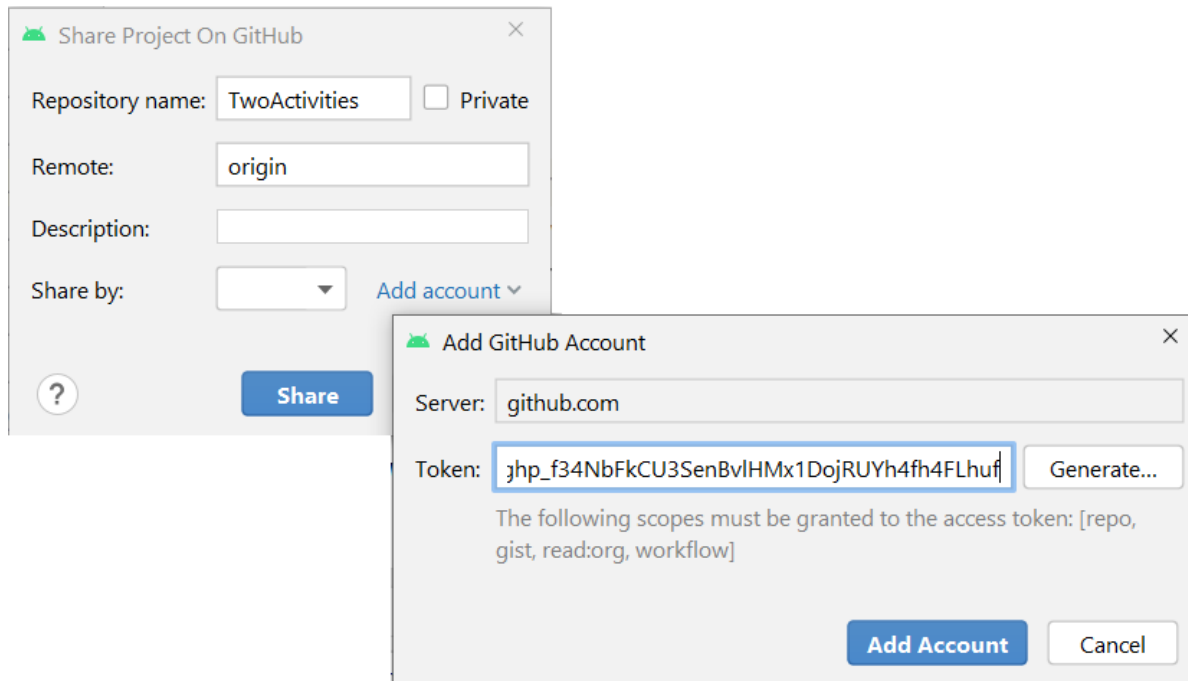
# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

## Utiliser Git dans Android Studio



### Utiliser Git dans Android Studio

4. Partager le projet dans GitHub : **Git > GitHub > Share Project on GitHub**. Un pop up s'affiche, cliquez sur « Add account » et générer le token pour se connecter à GitHub.



# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

## Utiliser Git dans Android Studio



### Utiliser Git dans Android Studio

Une fois que la connexion est réussie ! Un pop up s'affiche (Voir à droite).

- Sélectionner la racine de votre projet et cliquez sur « Add » ensuite « Share ». Vérifiez si le repository figure bien dans GitHub;

lachgar / TwoActivities Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags

Go to file Add file Code

| File/Folder            | Commit Message        | Time          |
|------------------------|-----------------------|---------------|
| lachgar Initial commit | 92eF1de 6 minutes ago | 1 commit      |
| .idea                  | Initial commit        | 6 minutes ago |
| app                    | Initial commit        | 6 minutes ago |
| gradle/wrapper         | Initial commit        | 6 minutes ago |
| .gitignore             | Initial commit        | 6 minutes ago |
| build.gradle           | Initial commit        | 6 minutes ago |
| gradle.properties      | Initial commit        | 6 minutes ago |
| gradlew                | Initial commit        | 6 minutes ago |

Add Files For Initial Commit

TwoActivities 13 files C:\Users\Lachgar\AndroidStudioProjects\TwoActivities

- .idea 5 files
  - .gitignore
  - compiler.xml
  - gradle.xml
  - misc.xml
  - vcs.xml
- gradle\wrapper 2 files
  - gradle-wrapper.jar
  - gradle-wrapper.properties
- .gitignore
- build.gradle

Commit Message

Initial commit

Add Cancel



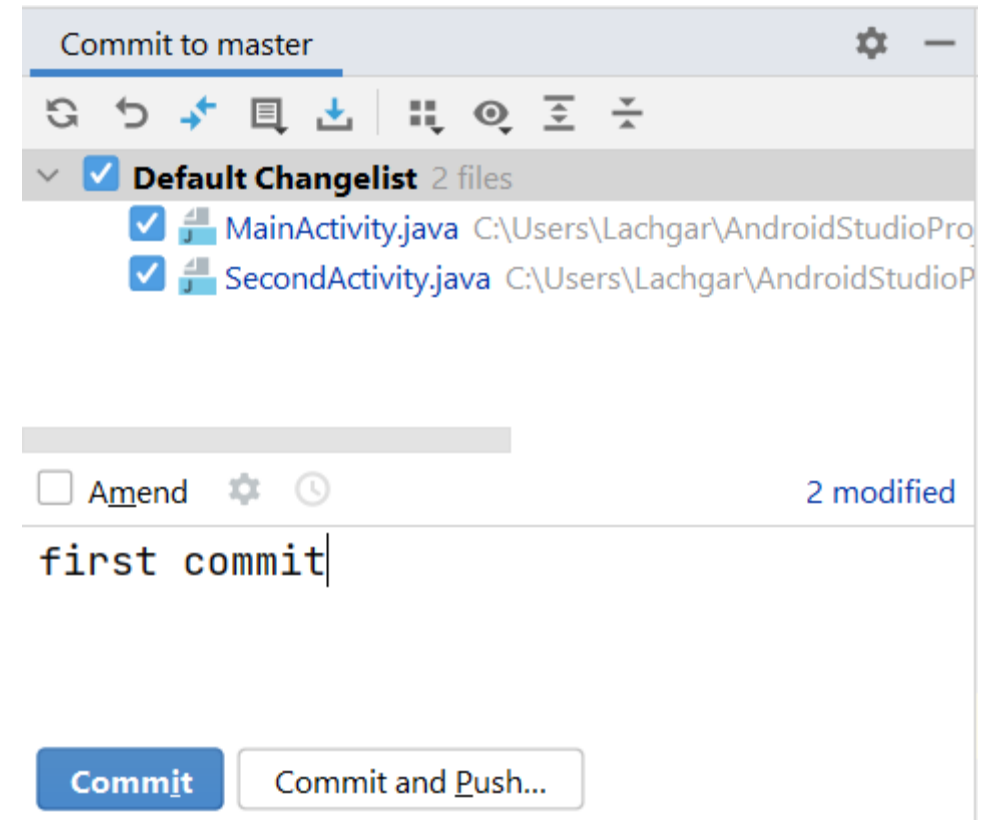
## ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

### Utiliser Git dans Android Studio



### Utiliser Git dans Android Studio

- Effectuez des modifications sur le projet dans Android Studio ensuite commiter.
- Après le commit, vous faites un push. Allez vers le menu **Git > push** pour envoyer les modifications vers le repository distant.
- Pour charger les dernières modifications réalisées dans le projet, cliquez sur le menu **Git > pull**.



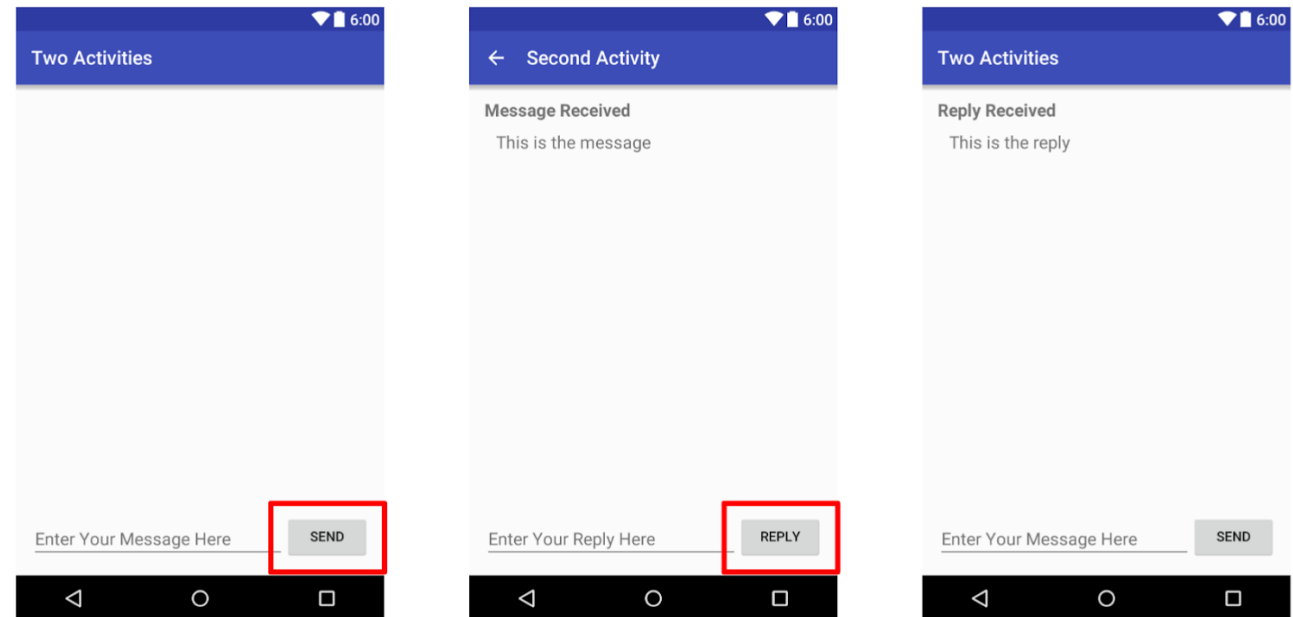
# ACTIVITÉ n° 3 – Utiliser Git pour manipuler le code des deux applications créées

## Utiliser Git dans Android Studio

### Collaborer sur un projet avec Git

Dans cette tâche, on demande d'ajouter au projet « twoActivities » la possibilité de renvoyer un message depuis l'activité « SecondActivity » vers « MainActivity ». Cette tâche sera réalisée par deux développeurs en même temps.

1. L'un va travailler sur le code JAVA ;
2. L'autre sur le code XML ;
3. Une fois terminer, les deux développeurs envoient le code vers le repository distant ;
4. Après chaque développeur récupère la nouvelle version de l'application depuis GitHub en local.



Main activity → Second activity → Back to Main activity