TRAVAUX PRATIQUES – FILIÈRE DÉVELOPPEMENT DIGITAL
Option – Applications mobiles
M210 – Programmer en Kotlin

55 heures

SOMMAIRE

MODALITÉS PÉDAGOGIQUES

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

PARTIE 1

Découvrir les fondamentaux de Kotlin

Dans ce module, vous allez :

- Préparer l'environnement de développement Kotlin
- Créer une application Hello world en Kotlin
- Maîtriser le lancement d'une application sur smartphone

6 heures

ACTIVITÉ n° 1

Installation de l’environnement et plugin Kotlin

Compétences visées :

- Utilisation de l'IDE Android Studio
- Installation du plugin Kotlin

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail

3 heures

CONSIGNES

Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

Pour l’apprenant :

- Ajouter les extensions recommandées pour Kotlin

Conditions de réalisation :

- Support de résumé théorique accompagnant

Critères de réussite :

- Le stagiaire doit être capable de :
  - Ajouter les plugins recommandées pour Kotlin
  - Créer et exécuter l'application Hello World sur les périphériques physiques

ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

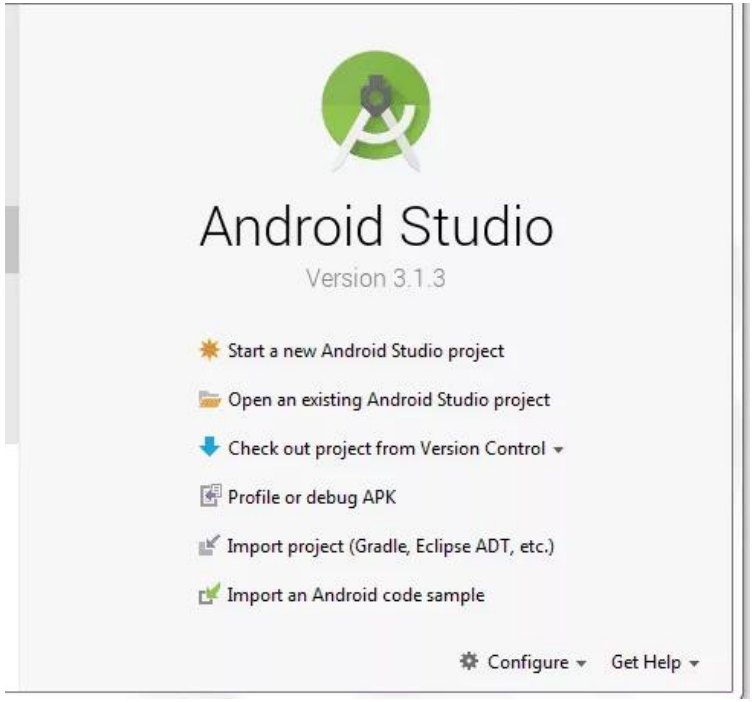
PARTIE 1

Etape 1 : Ajouter le plugin Kotlin à partir du menu de démarrage d'Android Studio

Démarrez Android Studio.

Si vous avez déjà un projet ouvert dans Android Studio, sautez cette étape.

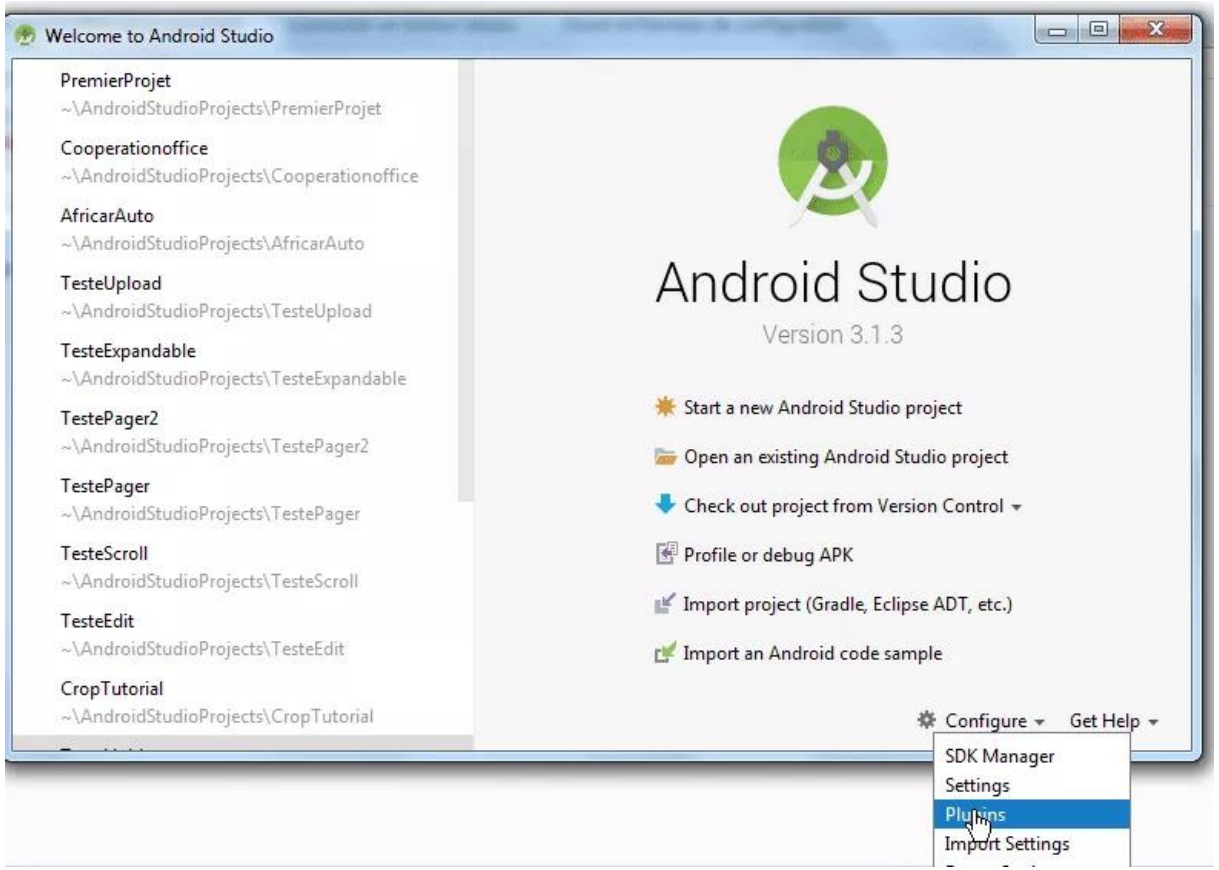
Si vous n'avez pas encore créé de projet ou si vous n'avez pas de projet ouvert dans Android Studio, vous devez voir afficher cette fenêtre.



ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

Etape 1 : Ajouter le plugin Kotlin à partir du menu de démarrage d'Android Studio

Cliquer sur **Configure** puis sur **Plugin.**

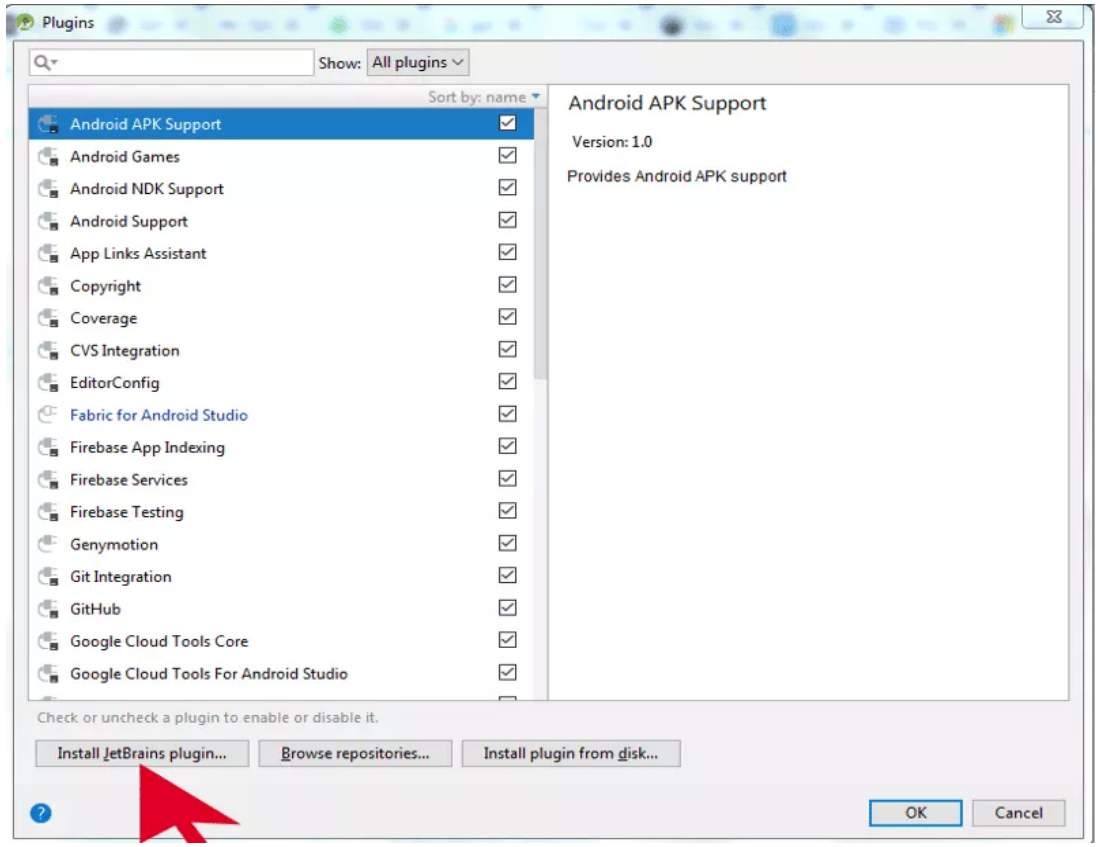


PARTIE 1

ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

Etape 1 : Ajouter le plugin Kotlin à partir du menu de démarrage d'Android Studio

Puis cliquez sur le bouton **Install JetBrains plugin.**



PARTIE 1

ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

Etape 1 : Ajouter le plugin Kotlin à partir du menu de démarrage d'Android Studio

Dans la nouvelle fenêtre qui s'affiche, tapez dans la barre de recherche **Kotlin**, puis cliquez sur le bouton **Install** pour installer le plugin. Vous devez redémarrer Android Studio pour appliquer le nouveau plugin.

PARTIE 1

Etape 2 : Ajouter le plugin de Kotlin depuis un projet existant déjà ouvert dans Android Studio

Pour ajouter Kotlin dans votre projet, vous pouvez soit ajouter le support de Kotlin manuellement soit utiliser le plugin de Kotlin pour l'ajouter automatiquement.

Ajouter Kotlin manuellement.

Pour commencer, ajouter dans le fichier Gradle de votre projet les lignes suivantes correspondant à Kotlin.

```
1.  buildscript {
2.      ext.kotlin_version = '1.2.71' //Le version de kotlin dans votre
    projet
3.  repositories {
4.          google()
5.          jcenter()
6.      }
7.      dependencies {
8.          classpath 'com.android.tools.build:gradle:3.1.3'
9.  classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    //Le plugin Kotlin
10.  }
11.  }
```

Etape 2 : Ajouter le plugin de Kotlin depuis un projet existant déjà ouvert dans Android Studio

Ajouter dans le fichier gradle (App) les lignes suivantes correspondant à Koltin.

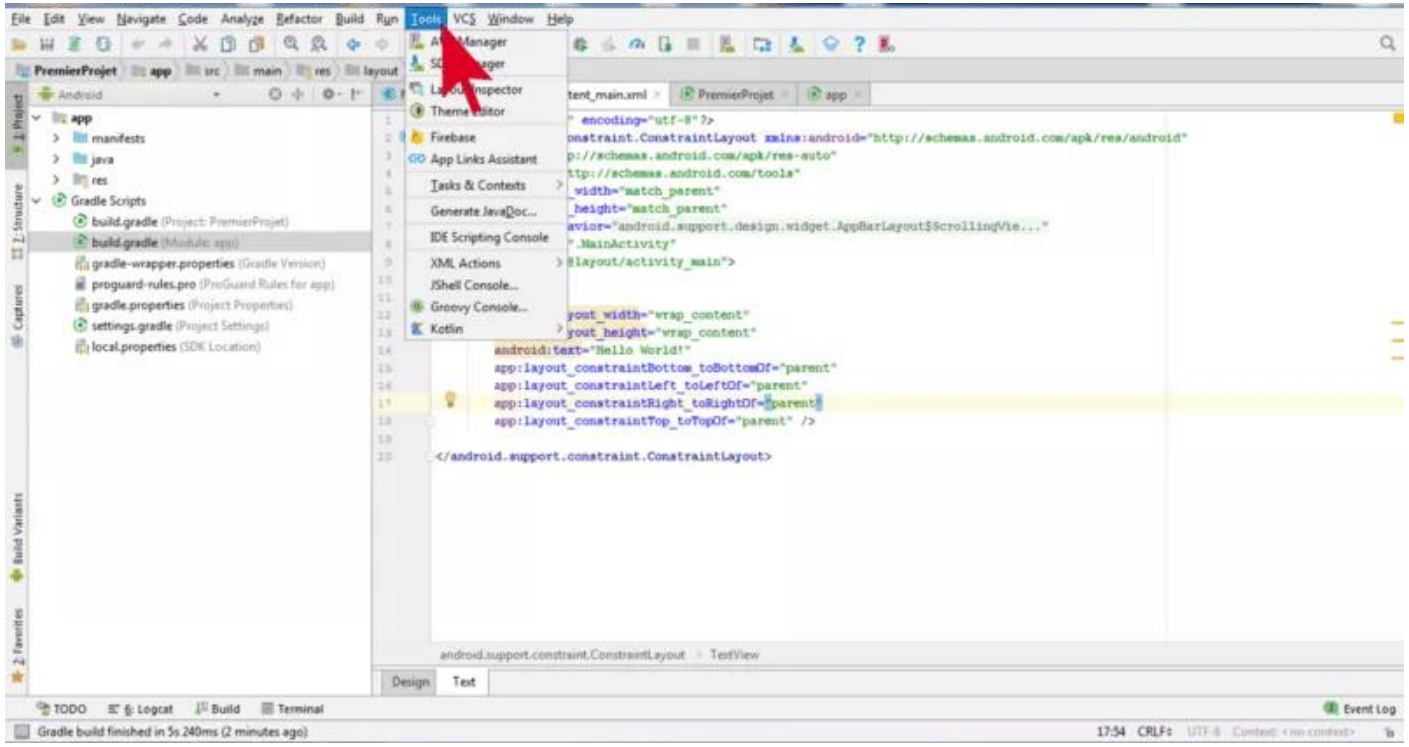
```
1.  apply plugin: 'com.android.application'
2.  apply plugin: 'kotlin-android'
3.
4.
5.  dependencies {
6.
7.  compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
8.  }
```

Puis cliquez sur **Sync now.**

ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

Etape 3: Ajouter Kotlin à votre projet à partir du plugin Kotlin

Dans le menu en haut sous Android Studio, cliquez sur **Tools.**

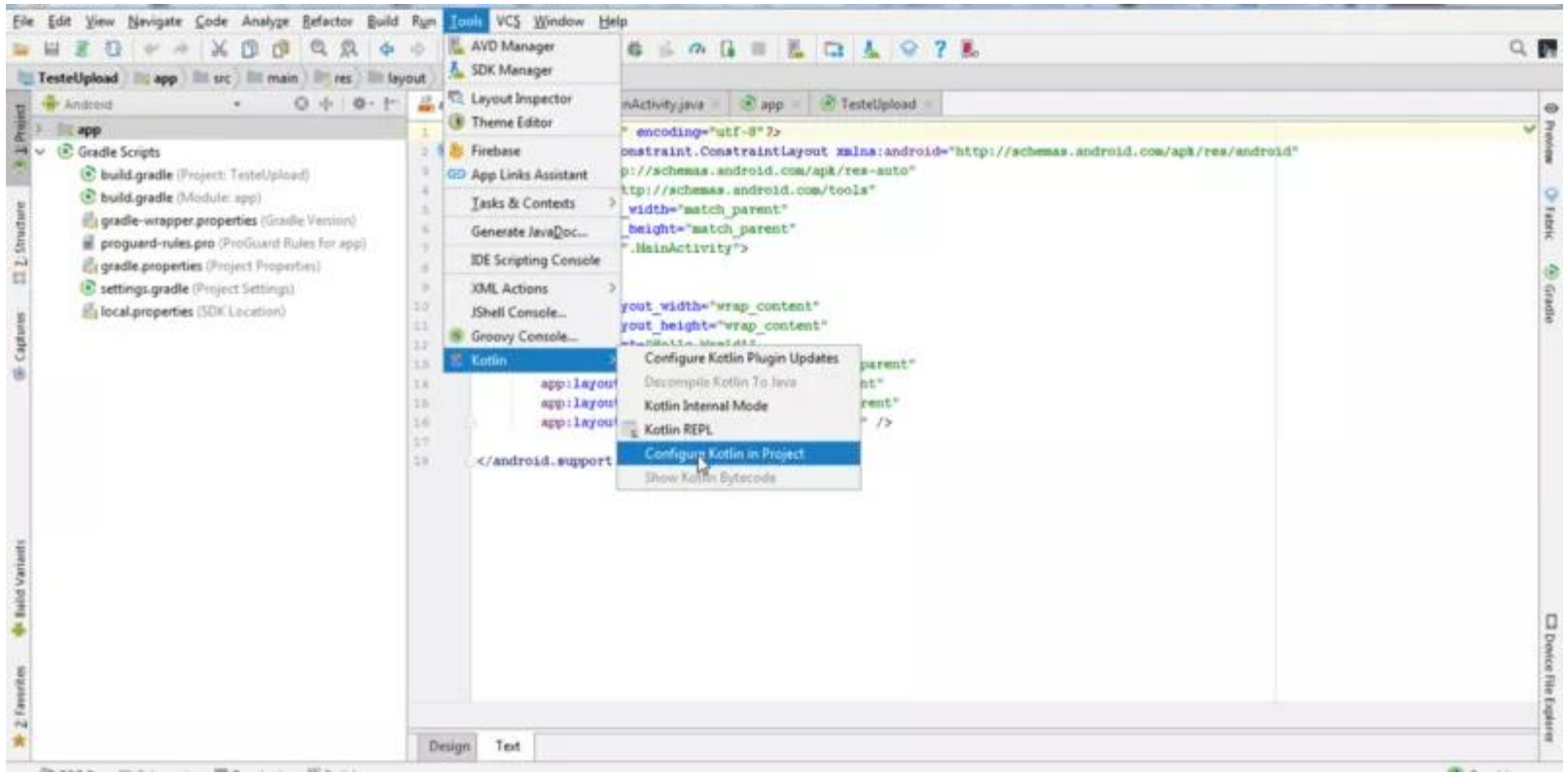


PARTIE 1

ACTIVITÉ n°1
Installation de l'environnement et plugin Kotlin

Etape 3: Ajouter Kotlin à votre projet à partir du plugin Kotlin

Puis sur **Kotlin->Configure Kotlin in Project.**



PARTIE 1

Etape 3: Ajouter Kotlin à votre projet à partir du plugin Kotlin

Une boite de dialogue s'affiche.

Choisissez **Android With gradle.** Toutes les configurations nécessaires seront ajoutées à votre projet comme nous l'avons vu dans la méthode d'ajout de Kotlin manuellement.

Cliquez ensuite sur **Sync now.**

PARTIE 1

ACTIVITÉ n° 2

Créer une application Hello World en Kotlin et la lancer sur smartphone

Compétences visées :

- Créer une application HELLO WORLD
- Lancer l'application Hello World, sur un émulateur et sur un périphérique physique

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail

3 heures

CONSIGNES

Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

Pour l’apprenant :

- Créer une application HELLO WORLD
- Lancer l’application Hello World, sur un émulateur et sur un périphérique physique

Conditions de réalisation :

- Support de résumé théorique accompagnant

Critères de réussite :

- Le stagiaire doit être capable de :
  - Créer et exécuter l'application Hello World sur les périphériques virtuels et physiques

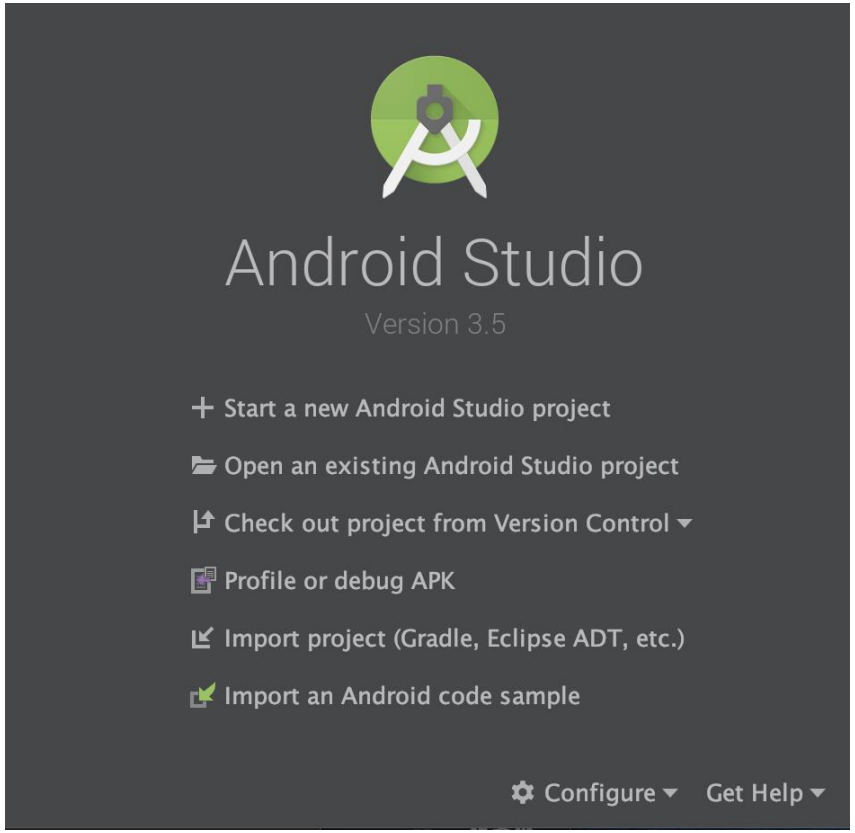
ACTIVITÉ n°2

Créer une application Hello World en Kotlin et la lancer sur smartphone

Ouverture d'Android Studio

Lorsque vous ouvrez Android Studio pour la première fois, vous serez invités à sélectionner ce que vous voulez faire :

Ici, nous allons simplement sélectionner **Démarrer un nouveau projet Android Studio**.



Remarques

- Cette activité suppose que vous avez installé Android Studio avec succès et que vous êtes prêt à créer votre toute première application, la classique Hello World.

PARTIE 1

ACTIVITÉ n°2

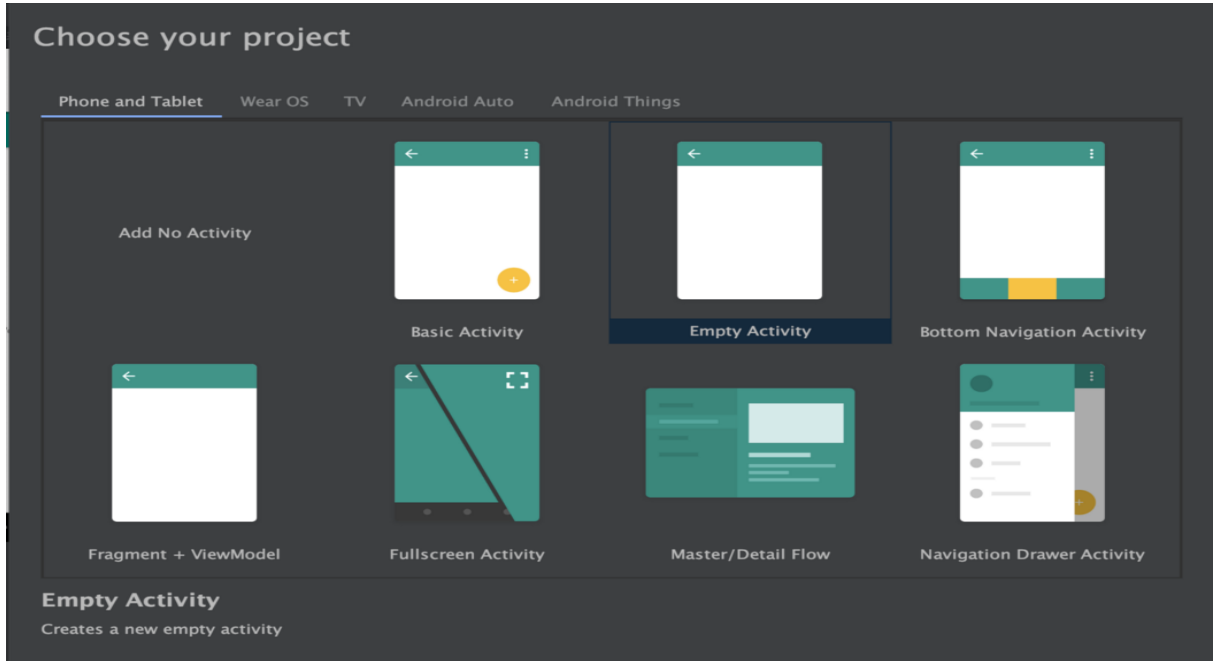
Créer une application Hello World en Kotlin et la lancer sur smartphone

Sélection du template

Ensuite, il vous demandera sur quel type de projet vous travaillez.

Android Studio vous propose 5 catégories de projets au choix :

- Téléphone & Tablette
- Wear OS : Montres
- la télé
- Android Auto : Automobile
- Choses Android : IoT



Parmi eux, vous avez le choix entre différents modèles.

Ici, nous commençons avec une activité vide dans le téléphone et la tablette.

PARTIE 1

ACTIVITÉ n°2

Créer une application Hello World en Kotlin et la lancer sur smartphone

Configuration du projet

L'étape suivante consiste à configurer votre projet :

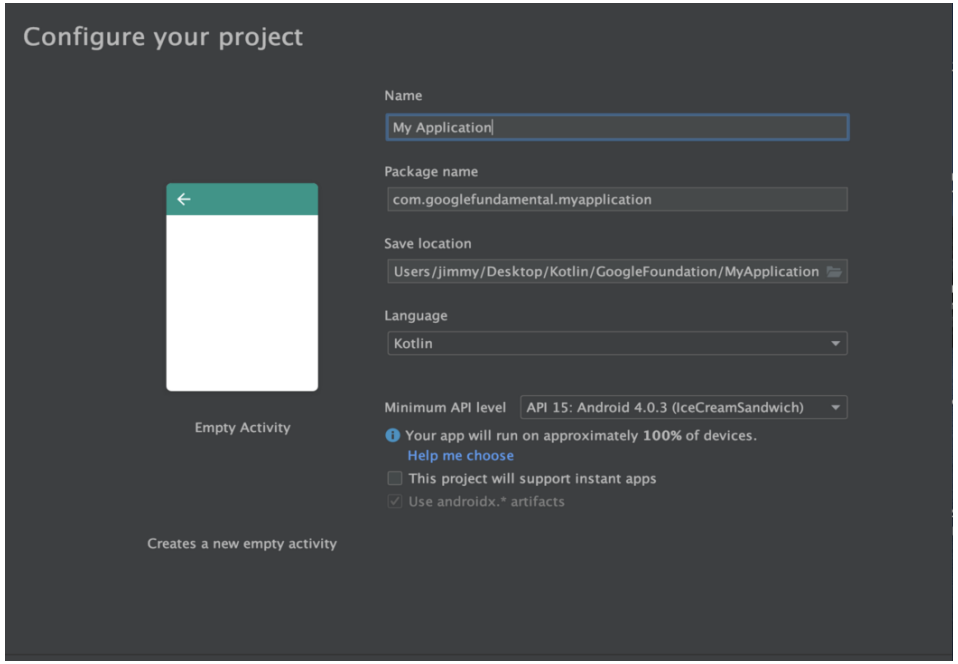
Ici, il vous sera demandé de remplir votre :

- **Nom de projet :** le nom d'affichage par défaut
- **Nom de package :** habituellement, nous lui donnons un nom de domaine dans un format inversé.

  Par exemple : www.google.com → com.google.www
- **Emplacement d'enregistrement :** l'emplacement où votre projet est enregistré.
- **Langage :** il peut s'agir de Kotlin ou de Java
- **Niveau d'API minimum :** le niveau d'API est déterminé par la version d'Android dans laquelle vous codez.

  Étant donné qu'Android est rétrocompatible, avec un niveau d'API inférieur, votre application couvrira plus d'appareils.

  Inversement, avec un niveau d'API plus élevé, la couverture de votre appareil diminuera. La couverture de l'appareil est indiquée juste sous cette sélection.
- **Applications instantanées :** si vous avez décidé de créer une application permettant à l'utilisateur d'accéder à vos données sans l'installation de votre application, alors c'est le bon choix pour vous.



PARTIE 1

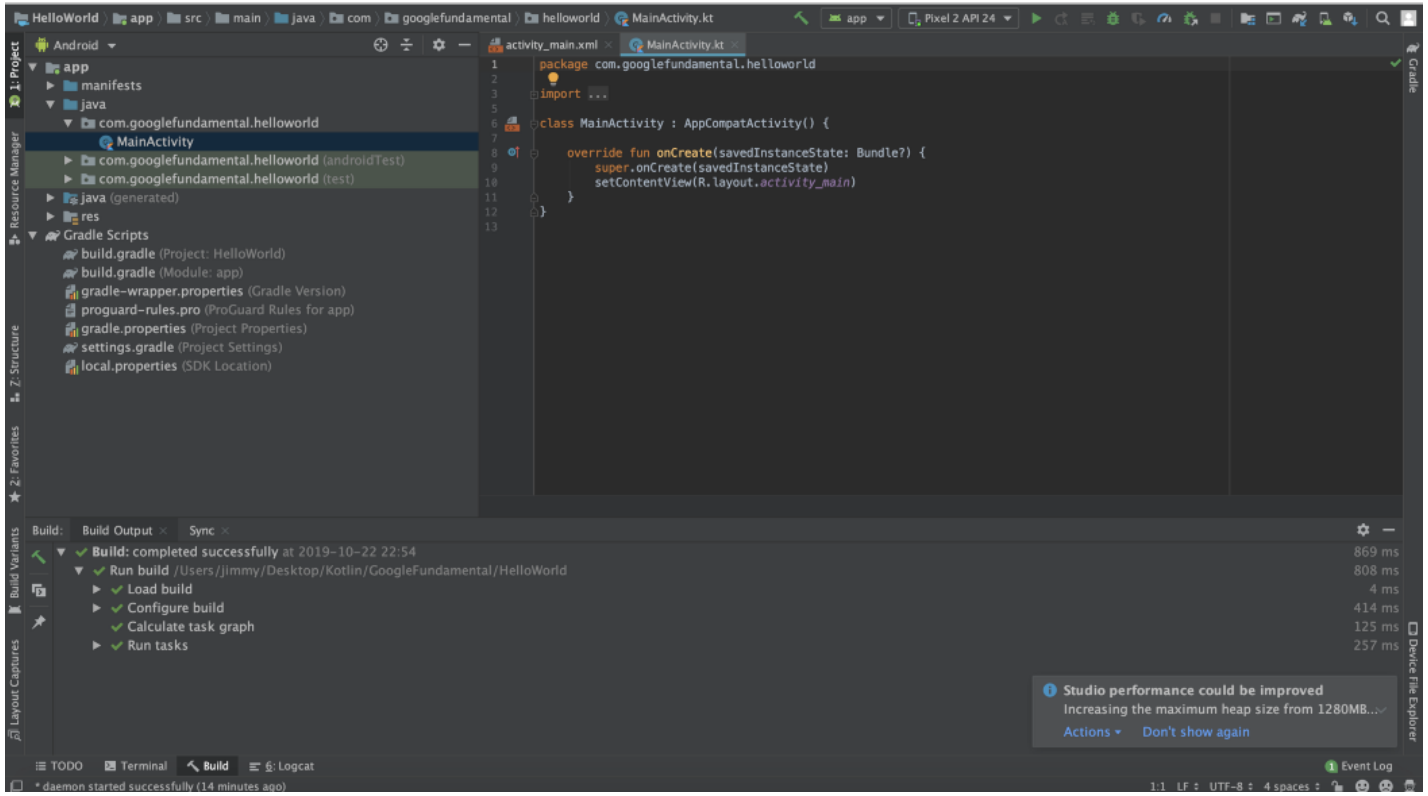
ACTIVITÉ n°2

Créer une application Hello World en Kotlin et la lancer sur smartphone

PARTIE 1

Exécutez l'application

Maintenant que vous avez terminé, vous devriez voir ceci :



ACTIVITÉ n°2

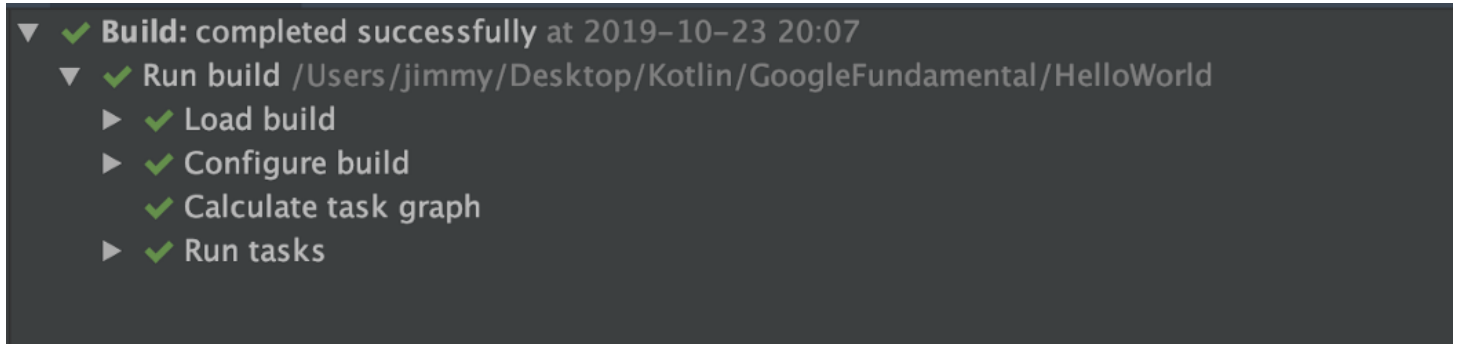
Créer une application Hello World en Kotlin et la lancer sur smartphone

Exécutez l'application

Vous avez un AVD déjà installé donc vous pouvez voir en haut :

Cela montre que si vous exécutez votre application en ce moment, votre application fonctionnera sur l'AVD selectionné.

Sélectionner le device branché avec votre ordinateur et cliquer sur RUN.



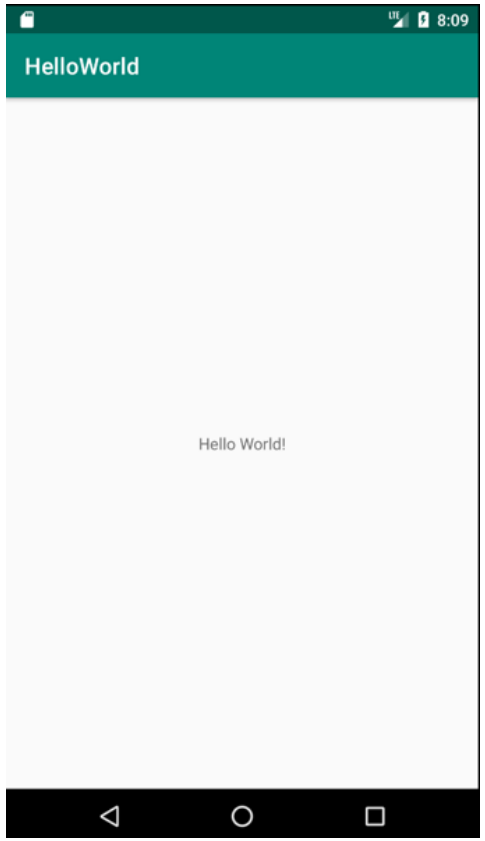
PARTIE 1

ACTIVITÉ n°2

Créer une application Hello World en Kotlin et la lancer sur smartphone

Exécutez l'application

C'est donc tout ce que vous devez faire pour exécuter un Hello World dans Android Studio.



PARTIE 1

PARTIE 2

S’initier à la programmation Kotlin

Dans ce module, vous allez :

- Utiliser les sealed class
- Maîtriser les data class

16 heures

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Compétences visées :

- Programmer en orienté objet avec Kotlin
- Maitriser des data class
- Utiliser des sealed class

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 :

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

2. Pour l'apprenant :

Dans cette activité, vous apprendrez comment :

- Programmer en orienté objet avec Kotlin
- Utiliser les data class
- Utiliser les sealed class

3. Conditions de réalisation :

- Un ordinateur sous Windows ou Linux ou un Mac sous MacOS
- Android Studio et environnement Android installé et configuré

4. Critères de réussite :

- Le stagiaire doit être capable de :
  - Programmer en orienté objet avec Kotlin
  - Utiliser les data class
  - Utiliser les sealed class

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Exercice 1

Écrire un programme en Kotlin permettant de modéliser la gestion d'une université :

- Créer une classe **University** avec un constructeur qui prend en paramètre le nom de l'université de type *string* et l'année de fondation de type *int*, cette classe contient 2 méthodes, une qui s'appelle **hide()** qui permet de recruter un professeur, et une autre qui s'appelle **enroll()** qui permet d'inscrire un étudiant à l'université.
- Créer une classe **Professor** avec un constructeur qui prend en paramètre le nom du professeur de type *string*, l'âge, et le salaire annuel de type *int*, un professeur enseigne plusieurs cours avec une méthode **teach()** qui prend en paramètre un objet de type **Course**, et une autre méthode **doOralExam()** qui examine oralement un étudiant sur un cours passé en paramètre et retourne un objet **Result** qui représente le résultat de l'examen.
- Créer une classe **Student** avec un constructeur qui prend en paramètre le nom de l'étudiant de type *string*, l'âge, et le numéro de matricule de type *int*, un étudiant s'inscrit à un cours avec la méthode **enroll()**, une méthode **takeExam()** pour passer un examen sur un cours, une méthode **learn()** qui incrémente la probabilité de succès de 2%, une méthode **party()** qui décrémente la probabilité de succès de 2%, et enfin une méthode **grade()** qui affecte le résultat d'un examen à un cours.
- Créer une classe **Result** qui contient deux états, un état de succès et d'échec.
- Dans une fonction main :
  - Créer un objet University
  - Créer un objet Professor
  - Créer un objet Student
  - Créer un objet Course

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Exercice 1

- Créer un objet Course
- L'Université recrute un Professeur
- L'Université inscrit un Etudiant
- Le Professeur enseigne un Cours
- L'Etudiant s'inscrit à un Cours
- L'Etudiant fait la Party 3 fois
- L'Etudiant Etudie une seule fois
- L'Etudiant passe un Examen
- Le Professeur passe un Examen Oral à un Etudiant

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 1

La classe **University :**

```
class University(val name: String, val foundingYear: Int) {

  val professors = mutableListOf<Professor>()
  val students = mutableListOf<Student>()

  fun hire(prof: Professor) {
    professors.add(prof)
    println("Hired Professor ${prof.name} at $name...")
  }

  fun enroll(newStudent: Student) {
    students.add(newStudent)
    println("Enrolled student ${newStudent.name} at $name...")
  }
}
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 1

La classe **Professor :**

```
class Professor(val name: String, val age: Int, val yearlySalary: Int) {

  val coursesTaught = mutableListOf<Course>()

  fun teach(course: Course) {
    coursesTaught.add(course)
    println("Teaching ${course.title}...")
  }

  fun doOralExam(student: Student, course: Course) {
    println("Testing student ${student.name} in ${course.title}")
    student.grade(course, Result.Sucess(70))  // Everyone gets an sucess
  }
}
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 1

La classe **Student :**

```
class Student(val name: String, val age: Int, val matriculationNo: Int) {

  val grades = mutableMapOf<Course, Result>()
  val currentEnrollments = mutableListOf<Course>()
  private var successLikelihoodInPercent: Int = 50

  fun enroll(course: Course) {
    currentEnrollments.add(course)
    println("$name enrolled in ${course.title}...")
  }
  fun takeExam(course: Course) {
    println("$name takes exam in ${course.title}...")
  }
  fun learn() {
    println("$name is learning...")
    successLikelihoodInPercent = min(successLikelihoodInPercent + 2, 100)
  }
  fun party() {
    println("$name is partying! \uD83C\uDF89")
    successLikelihoodInPercent = max(successLikelihoodInPercent - 2, 0)
  }
  fun grade(course: Course, grade: Result) {
    grades[course] = grade}
}
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 1

La classe **Course :**

```
data class Course(val title: String, val description: String)
```

La classe **Résultat :**

```
sealed class Result {
data class Sucess(val note: long) : Result()
data class Failure(val note: long) : Result()
object NoResultYet : Result()
 }
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 1

La fonction **Main :**

```
fun main(args: Array<String>) {
  val rwth = University(“OFPPT", 1870)
  val professor = Professor(“jean dupon", 44, 50_000)
  val student = Student("Sarah Keller", 22, 123456)
  val course = Course(
     "Programming I",
     "Learn object-oriented, functional, and logical programming"
  )

  rwth.hire(professor)
  rwth.enroll(student)
  professor.teach(course)
  student.enroll(course)
  student.party()
  student.party()
  student.party()
  student.learn()
  student.takeExam(course)
  professor.doOralExam(student, course)
}
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Exercice 2

Les sealed class représentent des hiérarchies de classes restreintes qui offrent plus de contrôle sur l'héritage. Toutes les sous-classes directes d'une sealed class sont connues au moment de la compilation. Aucune autre sous-classe ne peut apparaître en dehors d'un module dans lequel la sealed class est définie.

Le but de cet exercice est de récupérer la réponse d'une API et base de donnée sous forme d'une sealed class pour gérer les cas suivants :

- Ecrire dans la console d'Android Studio si l'API réseau retourne un code Failure
- Ecrire dans la console d'Android Studio les données récupérées si l'API retourne un code Success
- Ecrire dans la console d'Android Studio si la requête réseau est en cours

Remarques

- Cet exercice considère que la partie appel réseau est déjà implémentée.

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 2

La fonction **NetworkResponse<T :**

```
sealed class NetworkResponse<T>(
val data : T? = null,
val error : String? = null
){
class Success<T> (data : T) : NetworkResponse<T>(data = data)
class Failure<T> (error : String) : NetworkResponse<T>(error = error)
class Loading<T> () : NetworkResponse<T>()
}
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 2

La fonction **requestApiData :**

```
private fun requestApiData() {

viewModel.response.observe(viewLifecycleOwner) {response ->

        when (response) {
                is NetworkResponse.Success -> {

                // We can update the UI with the data  or any other task which requires the API response
                Log.d("Resultats",response.data)

                }

                is NetworkResponse.Failure -> {

               //we can handle the API failure case here like shwoing a Toast, dialog etc.
               Log.d("Error","failure")

                }

                is NetworkResponse.Loading -> {

                Log.d("Loading","loading")
                // this is the loading state, we can show shimmer here.

                } } }
```

PARTIE 2

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Exercice 3

Réécrivez le code Java suivant en Kotlin :

```
public class Person {
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

ACTIVITÉ n° 1

Créer une petite application POO en Kotlin et en utilisant les data class, sealed class

Solution 3

```
data class Person(val name: String, val age: Int)
```

PARTIE 2

ACTIVITÉ n° 1

Extensions : ajouter des fonctions personnalisées

Compétences visées :

- Utilisation de la fonction d'extension Kotlin

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 :

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

2. Pour l'apprenant :

- Dans cette activité, vous apprendrez comment :
  - Utiliser les extensions en Kotlin

3. Conditions de réalisation :

- Un ordinateur sous Windows ou Linux ou un Mac sous MacOS
- Android Studio et environnement Android installé et configuré

4. Critères de réussite :

- Le stagiaire doit être capable de :
  - ➢ Utiliser les extensions en Kotlin

ACTIVITÉ n° 1

Extensions : ajouter des fonctions personnalisées

Exercice 1

Kotlin offre la possibilité d'étendre une classe ou une interface avec une fonctionnalité avancée sans avoir besoin d'hériter de la classe ou d'utiliser des modèles de conception, comme Decorator.

Ceci est effectué avec des déclarations spéciales connues des extensions. En bref, une fonction d'extension est une fonction membre d'une classe, qui est définie en dehors de la classe.

Le but de cet exercice est de **créer une méthode de la classe String qui renvoie une nouvelle chaîne avec le premier et le dernier caractère éliminés**. Cette méthode n'est pas déjà accessible dans la classe String. Par conséquent, vous pouvez avoir la possibilité d'utiliser la fonction d'extension dans la programmation Kotlin pour effectuer cette tâche facilement.

PARTIE 2

ACTIVITÉ n° 1

Extensions : ajouter des fonctions personnalisées

Solution 1

```
fun String.removeFirstLastChar(): String =  this.substring(1, this.length - 1)
fun main(args: Array<String>) {
    val sampleString= "Extension Function"
    val output = sampleString.removeFirstLastChar()
    println("The final string is: $output")
}
```

Après avoir exécuté le code, vous verrez ceci comme suit :

```
The final string is: xtension Functio
```

PARTIE 2

ACTIVITÉ n° 1

Extensions : ajouter des fonctions personnalisées

Exercice 2

1) En utilisant les extensions Kotlin, créer une fonction d'échange à MutableList<Int>.
2) En utilisant les extensions Kotlin, créer une fonction qui ajoute le calcul du périmètre à une classe Circle.
3) Supposons que nous voulions appeler une méthode (disons isExcellent()) de la classe Student qui n'est pas définie dans la classe. Dans une telle situation, nous devons créer une fonction d'extension (isExcellent()) en dehors de la classe Student en tant que Student.isExcellent() et l'appelons depuis la fonction main(), sachant que pour être excellent la valeur du mark devra être supérieure à 90, la classe Student :

```
class Student{
    fun isPassed(mark: Int): Boolean{
        return mark>40
    }
}
```

PARTIE 2

Solution 2

1) En utilisant les extensions Kotlin, créer une fonction d'échange à MutableList<Int>.

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1]
    this[index1] = this[index2]
    this[index2] = tmp
}
fun main() {
    val list = mutableListOf(2, 4, 6, 8, 10)
    list.swap(1, 3)
    println(list)   // It will print [2, 8, 6, 4, 10]
}
```

Après avoir exécuté le code, vous verrez ceci comme suit :

```
[2, 8, 6, 4, 10]
```

Solution 2

2) En utilisant les extensions Kotlin, créer une fonction qui ajoute le calcul du périmètre à une classe Circle.

```
class Circle (val radius: Double){
    fun area(): Double{
        return Math.PI * radius * radius;
    }
}
fun Circle.perimeter(): Double{
        return 2*Math.PI*radius;
    }
fun main(){
    val sampleCircle = Circle(2.5);
    println("The area is: ${sampleCircle.area()}")
    println("The Perimeter is: ${sampleCircle.perimeter()}")
}
```

Solution 2

3) Supposons que nous voulions appeler une méthode (disons isExcellent()) de la classe Student qui n'est pas définie dans la classe. Dans une telle situation, nous devons créer une fonction d'extension (isExcellent()) en dehors de la classe Student en tant que Student.isExcellent() et l'appelons depuis la fonction main(), sachant que pour être excellent la valeur du mark devra être supérieure à 90.

```
class Student{
    fun isPassed(mark: Int): Boolean{
        return mark>40
    }
 }
fun Student.isExcellent(mark: Int): Boolean{
    return mark > 90  }
fun main(args: Array<String>){
val student = Student()
val passingStatus = student.isPassed(55)
println("student passing status is $passingStatus")
val excellentStatus = student.isExcellent(95)
println("student excellent status is $excellentStatus")
}
```

PARTIE 3

Maitriser les fonctions et lambdas

Dans ce module, vous allez :

- Comprendre les expressions et opérateurs
- Utiliser les fonctions inline et high-order functions
- Manipuler les expressions lambdas
- Manipuler les expressions anonymes

10 heures

Activité 1

Travaux pratiques sur la réalisation d'une application Kotlin

Compétences visées :

- Comprendre les expressions et opérateurs
- Utiliser les fonctions inline et high-order functions
- Manipuler les expressions lambdas
- Manipuler les expressions anonymes

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail
- Utiliser le résumé théorique pour réaliser le projet de synthèse

10 heures

CONSIGNES

1. Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

2. Pour l'apprenant :

- Comprendre les expressions et opérateurs
- Utiliser les fonctions inline et high-order functions
- Manipuler les expressions lambdas
- Manipuler les expressions anonymes

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Environnement de développement installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Déclarer des fonctions
  - Manipuler les expressions lambdas et fonctions anonymes
  - Utiliser les high-order Functions et fonctions inline

Activité 1 :
Travaux pratiques sur la réalisation d'une application Kotlin

Exercice 1 :

Kotlin prend en charge la programmation fonctionnelle. Voir le résumé théorique pour en savoir plus sur les lambdas dans Kotlin.

Passez un lambda à la fonction any pour vérifier si la collection contient un nombre pair. La fonction any obtient un prédicat comme argument et renvoie true si au moins un élément satisfait le prédicat :

```
fun containsEven(collection: Collection<Int>): Boolean =
        collection.any { TODO() }
```

PARTIE 3

Activité 1 :
Travaux pratiques sur la réalisation d'une application Kotlin

Solution 1 :

Kotlin prend en charge la programmation fonctionnelle. Voir le résumé théorique pour en savoir plus sur les lambdas dans Kotlin.

Passez un lambda à la fonction any pour vérifier si la collection contient un nombre pair. La fonction any obtient un prédicat comme argument et renvoie true si au moins un élément satisfait le prédicat :

```
fun containsEven(collection: Collection<Int>): Boolean =
        collection.any {it % 2 == 0}
```

PARTIE 3

Exercice 2 :

Implémentez une fonction qui calcule le montant total dépensé par le client : la somme des prix de tous les produits commandés par un client donné. Notez que chaque produit doit être compté autant de fois qu'il a été commandé.

Utilisez sum sur une collection de nombres ou sumOf pour convertir d'abord les éléments en nombres, puis les additionner.

Solution 2 :

Implémentez une fonction qui calcule le montant total dépensé par le client : la somme des prix de tous les produits commandés par un client donné. Notez que chaque produit doit être compté autant de fois qu'il a été commandé.

Utilisez sum sur une collection de nombres ou sumOf pour convertir d'abord les éléments en nombres, puis les additionner.

```
// Return the sum of prices for all the products ordered by a given customer
fun moneySpentBy(customer: Customer): Double =
    customer.orders.flatMap { it.products }.sumOf { it.price }
```

Activité 1 :
Travaux pratiques sur la réalisation d'une application Kotlin

Exercice 3 :

Ajoutez un getter personnalisé pour rendre le val lazy. Il doit être initialisé en appelant **initializer()** lors du premier accès.

Vous pouvez ajouter des propriétés supplémentaires selon vos besoins. N'utilisez pas de propriétés déléguées !

```
class LazyProperty(val initializer: () -> Int) {
    /* TODO */
    val lazy: Int
        get() {
            TODO()
        }
}
```

PARTIE 3

Solution 3 :

Ajoutez un getter personnalisé pour rendre le val lazy. Il doit être initialisé en appelant **initializer()** lors du premier accès.

Vous pouvez ajouter des propriétés supplémentaires selon vos besoins. N'utilisez pas de propriétés déléguées !

```
class LazyProperty(val initializer: () -> Int) {
    var value: Int? = null
    val lazy: Int
        get() {
            if (value == null) {
                value = initializer()
            }
            return value!!
        }
}
```

Exercice 4 :

Les objets avec la méthode **invoke()** peuvent être invoqués en tant que fonction.

Vous pouvez ajouter une extension d'appel pour n'importe quelle classe, mais il vaut mieux ne pas en abuser :

```
operator fun Int.invoke() { println(this) }

1() //huh?..
```

Implémentez la fonction **Invokable.invoke()** pour compter le nombre de fois où elle est invoquée.

Solution 4 :

```
class Invokable {
    var numberOfInvocations: Int = 0
        private set

    operator fun invoke(): Invokable {
        numberOfInvocations++
        return this
    }
}

fun invokeTwice(invokable: Invokable) = invokable()()
```

Activité 1 :
Travaux pratiques sur la réalisation d'une application Kotlin

Exercice 5 :

Implémentez l'arithmétique des dates et prenez en charge l'ajout d'années, de semaines et de jours à une date. Vous pouvez écrire le code comme ceci : date + ANNÉE * 2 + SEMAINE * 3 + JOUR * 15.

- Tout d'abord, ajoutez la fonction d'extension **plus()** à **MyDate**, en prenant **TimeInterval** comme argument. Utilisez la fonction utilitaire **MyDate.addTimeIntervals()** déclarée dans **DateUtil.kt**
- Ensuite, essayez de prendre en charge l'ajout de plusieurs intervalles de temps à une date. Vous aurez peut-être besoin d'un cours supplémentaire

PARTIE 3

Solution 5 :

```
data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int)
// Supported intervals that might be added to dates:
enum class TimeInterval { DAY, WEEK, YEAR }
operator fun MyDate.plus(timeInterval: TimeInterval) = addTimeIntervals(timeInterval, 1)


class RepeatedTimeInterval(val timeInterval: TimeInterval, val number: Int)


operator fun TimeInterval.times(number: Int) =  RepeatedTimeInterval(this, number)


operator fun MyDate.plus(timeIntervals: RepeatedTimeInterval) =addTimeIntervals(timeIntervals.timeInterval, timeIntervals.number)
fun task1(today: MyDate): MyDate {
    return today + YEAR + WEEK
}
fun task2(today: MyDate): MyDate {
    return today + YEAR * 2 + WEEK * 3 + DAY * 5
}
```

Activité 1 :
Travaux pratiques sur la réalisation d'une application Kotlin

Exercice 6 :

Ajoutez la fonction compareTo à la classe MyDate pour la rendre comparable. Après cela, le code ci-dessous date1 < date2 devrait commencer à se compiler.

Notez que lorsque vous remplacez un membre dans Kotlin, le modificateur de remplacement est obligatoire.

```
data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int) : Comparable<MyDate> {
    /* TODO */
}

fun test(date1: MyDate, date2: MyDate) {
    // this code should compile:
    println(date1 < date2)
}
```

PARTIE 3

Solution 6 :

```
data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int) : Comparable<MyDate> {
    override fun compareTo(other: MyDate) = when {
        year != other.year -> year - other.year
        month != other.month -> month - other.month
        else -> dayOfMonth - other.dayOfMonth
    }
}

fun test(date1: MyDate, date2: MyDate) {
    // this code should compile:
    println(date1 < date2)
}
```

PARTIE 4

Maitriser les aspects avancés de Kotlin

Dans ce module, vous allez :

- Introduire les coroutines
- Utiliser les types checks et Casts

16 heures

Activité 1

Créer une petite application multithread en utilisant les coroutines

Compétences visées :

- Utiliser les coroutines

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail
- Utiliser le résumé théorique pour réaliser le projet de synthèse

12 heures

CONSIGNES

1. Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

2. Pour l’apprenant :

- Créer une application multithread en utilisant les coroutines

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Environnement de développement installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - ➢ Utiliser les coroutines

Activité 1 :

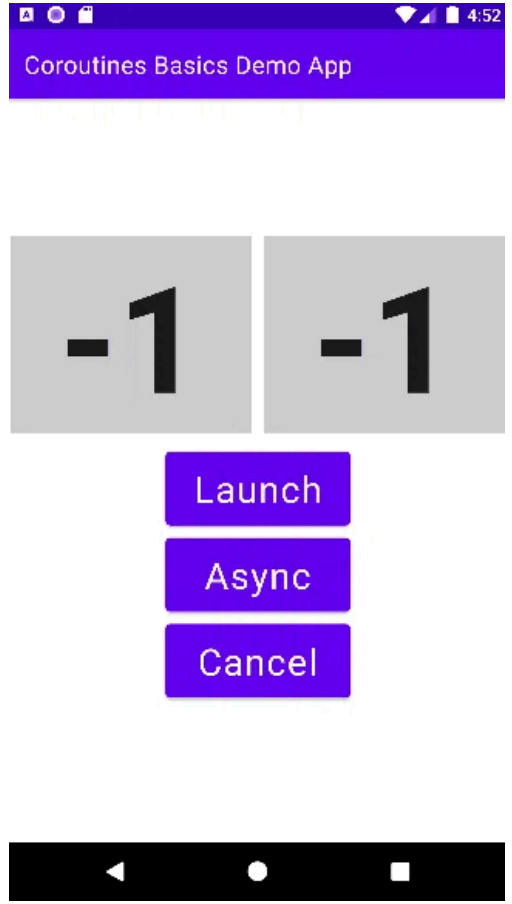
Créer une petite application multithread en utilisant les coroutines

Exercice 1 :

L'objectif de cet exercice est de créer une application qui montre comment créer des tâches de coroutines et les exécuter simultanément.

Il y a 3 boutons et 2 textes d'affichage (gauche et droite) dans cette application.

- Les boutons Launch et Async mettent à jour simultanément l'interface utilisateur du texte gauche et du texte droit
- Le texte de gauche et le texte de droite commencent à -1 comme valeur invalide
- Lorsque le bouton launch ou Async est cliqué, des coroutines sont créées pour mettre à jour l'interface utilisateur de texte gauche de 0 → 9 et l'interface utilisateur de texte droit de 10 → 19
- Si le bouton Annuler est cliqué, les deux textes ne sont plus mis à jour et la valeur est définie sur -1
- Si aucun bouton Annuler n'est cliqué, les deux textes continuent d'être mis à jour jusqu'à la valeur finale 9 et 19



Remarques

- Vous êtes libre de choisir l'architecture et l'interface de l'application qui vous convient le but de cet exercice est vraiment comprendre le multithreading avec les coroutines.

PARTIE 4

Activité 1 :

Créer une petite application multithread en utilisant les coroutines

Solution 1 :

Voir le projet Android : **Demo_CoroutinesBasics**

- La classe MainActivity voir : **app/src/main/java/com/example/coroutinesbasics/MainActivity.kt**
- La classe Utils voir : **app/src/main/java/com/example/coroutinesbasics/utils/Utils.kt**
- La classe MainScreen voir : **app/src/main/java/com/example/coroutinesbasics/main/MainScreen.kt**
- La classe MainViewModel voir : **app/src/main/java/com/example/coroutinesbasics/main/MainViewModel.kt**
- Les classes Colors, Shape, Theme et Type voir le package : **app/src/main/java/com/example/coroutinesbasics/ui/theme/**

Activité 2

Créer un programme pour manipuler les différents types Kotlin

Compétences visées :

- Utiliser les types Checks et Casts

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail
- Utiliser le résumé théorique pour réaliser le projet de synthèse

2 heures

CONSIGNES

1. Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

2. Pour l'apprenant :

- Utiliser les types Checks et Casts

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Environnement de développement installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - ➢ Utiliser les types Checks et Casts

Activité 2 :
Créer un programme pour manipuler les différents types Kotlin

Exercice 1 :

Réécrivez le code Java suivant à l'aide de smart cast et de l'expression when :

```
public int eval(Expr expr) {
    if (expr instanceof Num) {
        return ((Num) expr).getValue();
    }
    if (expr instanceof Sum) {
        Sum sum = (Sum) expr;
        return eval(sum.getLeft()) + eval(sum.getRight());
    }
    throw new IllegalArgumentException("Unknown expression");
}
```

Activité 2 :

Créer un programme pour manipuler les différents types Kotlin

Solution 1 :

```
fun eval(expr: Expr): Int =
        when (expr) {
            is Num -> expr.value
            is Sum -> eval(expr.left) + eval(expr.right)
            else -> throw IllegalArgumentException("Unknown expression")
        }

interface Expr
class Num(val value: Int) : Expr
class Sum(val left: Expr, val right: Expr) : Expr
```

PARTIE 4

Activité 2 :
Créer un programme pour manipuler les différents types Kotlin

Exercice 2 :

Réutilisez votre solution de l’exercice 1, mais remplacez l'interface par l'interface sealed. Ensuite, vous n'avez plus besoin de la branche else dans when.

Activité 2 :

Créer un programme pour manipuler les différents types Kotlin

Solution 2 :

```
fun eval(expr: Expr): Int =
        when (expr) {
            is Num -> expr.value
            is Sum -> eval(expr.left) + eval(expr.right)
        }

sealed interface Expr
class Num(val value: Int) : Expr
class Sum(val left: Expr, val right: Expr) : Expr
```

PARTIE 5

Utiliser les outils Android et Kotlin

Dans ce module, vous allez :

- Utiliser Dokka et KDoc
- Manipuler Gradle
- Créer un plugin Kotlin

7 heures

Activité 1

Ajouter des bibliothèques externes dans Gradle

Compétences visées :

- Utiliser Dokka et KDoc
- Manipuler Gradle

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail
- Utiliser le résumé théorique pour réaliser le projet de synthèse

3 heures

CONSIGNES

1. Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

2. Pour l'apprenant :

- Utiliser Dokka et KDoc
- Manipuler Gradle

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Environnement de développement installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - ➢ Utiliser Dokka et KDoc
  - ➢ Manipuler Gradle

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Exercice 1 :

Ajouter les bibliothèques Dokka, Kotlin, Espresso et Constraint layout comme dépendance à un fichier app/gradle.

PARTIE 5

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 1 :

```
apply plugin: 'com.android.application'
apply plugin: 'Kotlin-android'
apply plugin: 'Kotlin-android-extensions'
apply plugin: 'org.jetbrains.dokka'
android {
compileSdkVersion 27
defaultConfig {
applicationId "br.az.dokkatest"
minSdkVersion 15
targetSdkVersion 27
versionCode 1
versionName "1.0"
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
```

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 1 - suite:

```
buildTypes {
release {
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
}
}
task dokka(overwrite: true, type: org.jetbrains.dokka.gradle.DokkaTask) {
outputFormat = 'html'
outputDirectory = "$buildDir/dokka"
packageOptions {
prefix = "android"
suppress = true
}
```

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 1 - suite :

```
packageOptions {
prefix = "br.az.dokkatest"
suppress = false
}
}
}
dependencies {
implementation fileTree(dir: 'libs', include: ['*.jar'])
implementation"org.jetbrains.Kotlin:Kotlin-stdlib-jre7:$Kotlin_version"
implementation 'com.android.support:appcompat-v7:27.1.1'
implementation 'com.android.support.constraint:constraint-layout:1.1.2'
testImplementation 'junit:junit:4.12'
androidTestImplementation 'com.android.support.test:runner:1.0.2'
androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
}
```

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Exercice 2 :

Après l'ajout des dépendances dans l'exercice précédent, créer et commenter les classes suivantes :

MainActivity.kt :

```
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
                super.onCreate(savedInstanceState)
                setContentView(R.layout.activity_main)
                val objectOfObject = Object("Test")
                objectOfObject.parametersWithReturn("Hello World")
                objectOfObject.versionAndSince()
                try {
                objectOfObject.exceptionThrow()
                } catch (e: Exception) {
                Toast.makeText(applicationContext, "Exception method called", Toast.LENGTH_LONG).show()
                }
        }
}
```

PARTIE 5

Exercice 2 :

Object.kt :

```
class Object(internal var mainData: String) {
fun parametersWithReturn(input: String): String {
        val output = "output"
        return output
    }
fun exceptionThrow(): Exception {
        throw Exception()
    }
fun versionAndSince(): String {
        return "String"
    }
}
```

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 2 :

MainActivity.kt :

```
/**
 * @author TEST TEST
 * Email: TEST_TEST@gmail.com
 * this Class is main activity of this project
 */
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val objectOfObject = Object("Test")
        objectOfObject.parametersWithReturn("Hello World")
        objectOfObject.versionAndSince()
        try {
            objectOfObject.exceptionThrow()
        } catch (e: Exception) {
            Toast.makeText(applicationContext, "Exception method called", Toast.LENGTH_LONG).show()
        } }}
```

PARTIE 5

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 2 :

Object.kt :

```
/**
 * @author TEST TEST
 * Email: TEST_TEST@gmail.com
 * Copyright. All rights reserved
 * @param mainData send as a parameter
 * @constructor Create an object
 */
class Object(internal var mainData: String) {
    /**
     * This method shows parameters input and output
     * @param input gets data from calling
     * @return String as "output"
     */
    fun parametersWithReturn(input: String): String {
        val output = "output"
        return output
    }
```

Activité 1 :
Ajouter des bibliothèques externes dans Gradle

Solution 2 :

Object.kt – suite :

```
    /**
     * This method throws Exception
     * @throws Exception
     */
    fun exceptionThrow(): Exception {
        throw Exception()
    }

    /**
     * @since 5.5.8.5
     * @return String as a parameter
     */
    fun versionAndSince(): String {
        return "String"
    }
}
```

Activité 2

Créer un plugin Kotlin réutilisable et personnalisé

Compétences visées :

- Manipuler les plugins et modules Kotlin

Recommandations clés :

- Suivre les instructions du TP et organiser le dossier de travail
- Utiliser le résumé théorique pour réaliser le projet de synthèse

4 heures

CONSIGNES

1. Pour le formateur :

- Demander aux apprenants de suivre les étapes décrites dans le résumé théorique du cours et d'appliquer les procédures
- Demander aux apprenants de réaliser le travail de synthèse

2. Pour l'apprenant :

- Manipuler les plugins et modules Kotlin

3. Conditions de réalisation :

- Support de résumé théorique accompagnant
- Environnement de développement installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - ➢ Manipuler les plugins et modules Kotlin

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Exercice 1 :

Créer une application Android simple dotée d'un label de texte d'heure actuelle et d'un bouton pour obtenir l'heure. Sur le clic du bouton obtenir l'heure, l'heure actuelle sera récupérée et le label de texte de l'heure continuera d'afficher l'heure actuelle.

L'interface utilisateur n'aura qu'un appel de fonction à la bibliothèque et toute la logique métier sera écrite dans une bibliothèque Android.

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

Étape 1 : Démarrez Android Studio et créez un nouveau projet avec une activité principale vide.

Étape 2 : Ajoutez un bouton et une vue texte de l'heure. Pour moi, l'interface utilisateur ressemble à celle ci-dessous.

Étape 3 : Ajoutez le code pour effectuer le clic sur le bouton et d'autres détails du code seront expliqués plus loin.

Remarques

Les étapes de base liées à la création d'un projet et à l'écriture du code pour l'interface utilisateur, ne sont pas expliquées dans cette partie.

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

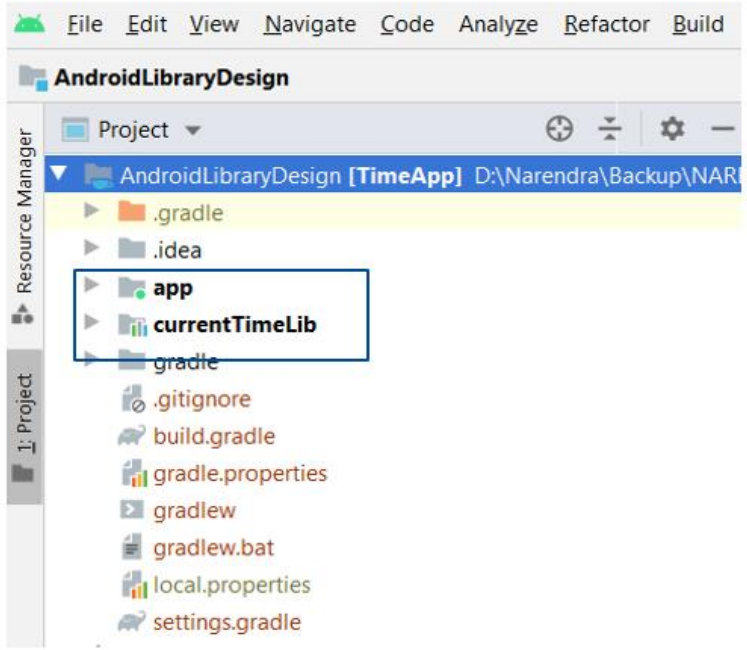
Solution 1 :

Étape 4 : Construisez maintenant la bibliothèque >> Suivez les étapes ci-dessous :

4.1 Cliquer sur **File > New > New Module**

4.2 Dans la fenêtre **Create New Module** qui apparaît, cliquez sur **Android Library**, puis cliquez sur **Suivant**.

Enfin, cliquez sur le bouton **Terminer** et le module de bibliothèque sera ajouté à la structure du projet comme indiqué dans l'image ci-dessous :



Remarques

app module est un module d'interface utilisateur et currentTimeLib est le module de bibliothèque.

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

Étape 5 : Ajoutez votre bibliothèque en tant que dépendance

Dans cet exemple, nous avons créé le module de bibliothèque dans le projet afin que le module de bibliothèque soit déjà disponible et inclus dans le fichier **settings.gradle**. S'il n'est pas disponible, incluez le module de bibliothèque dans le fichier **settings.gradle**.

Vérifiez si vous obtenez comme dans l'image ci-dessous :

settings.gradle

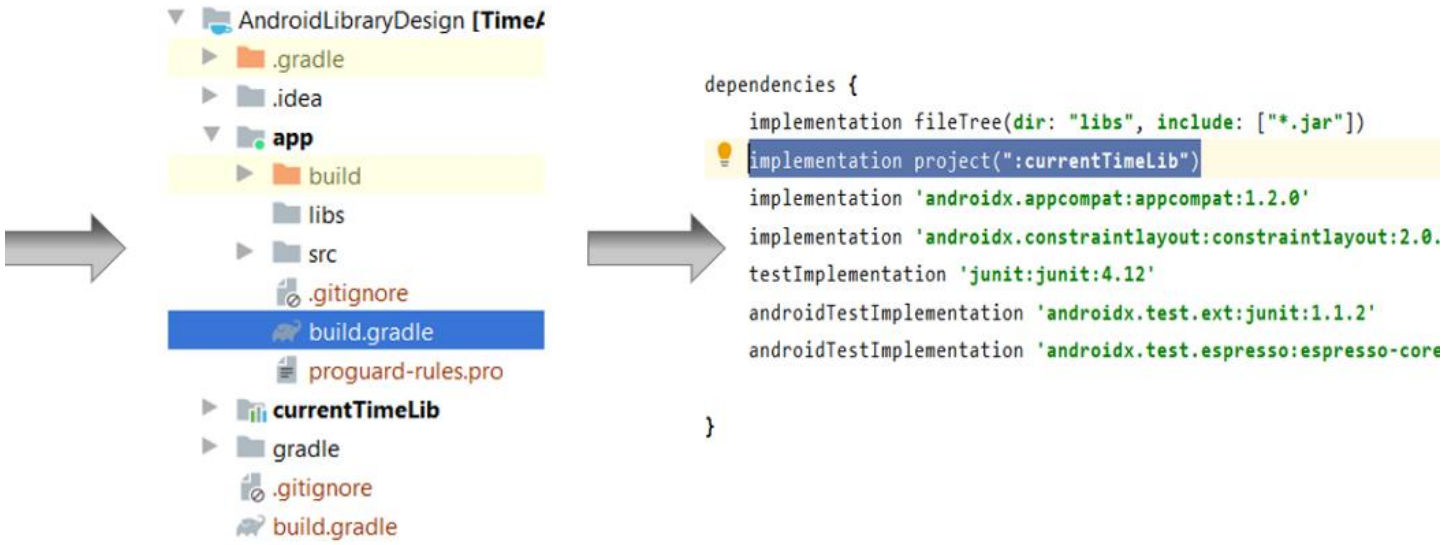
```
1  include ':currentTimeLib'
2  include ':app'
3  rootProject.name = "TimeApp"
```

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

5.1 Ajoutez le module de bibliothèque dans le fichier Gradle du module d'application.



Synchronisez le projet et exécutez-le, il devrait fonctionner avec Gradle réussi afin que nous puissions nous assurer que toutes les étapes sont correctement effectuées.

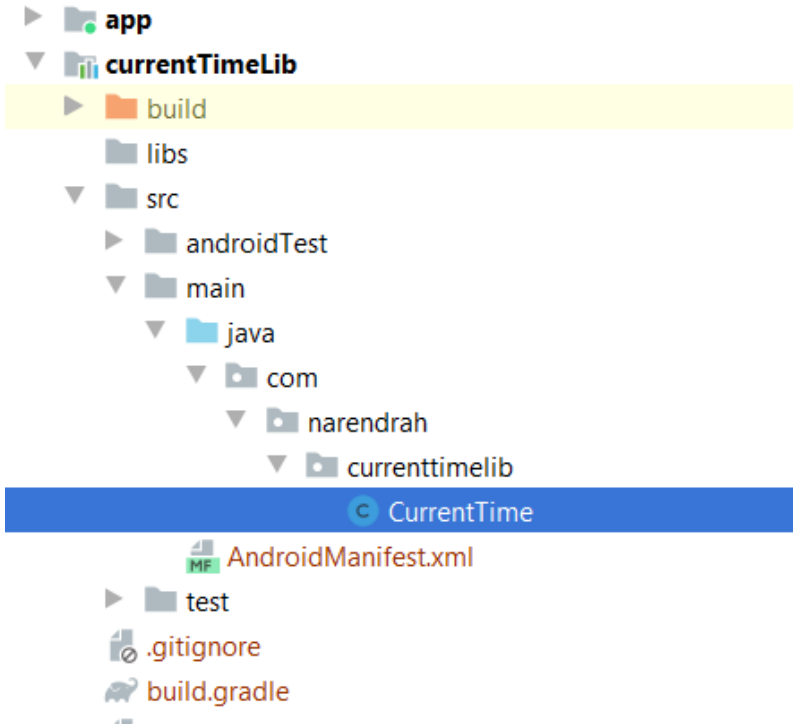
PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

Étape 6 : Nous allons maintenant écrire la logique métier pour récupérer l'heure actuelle dans le module de bibliothèque.

6.1 Créez une classe Java nommée CurrentTime.

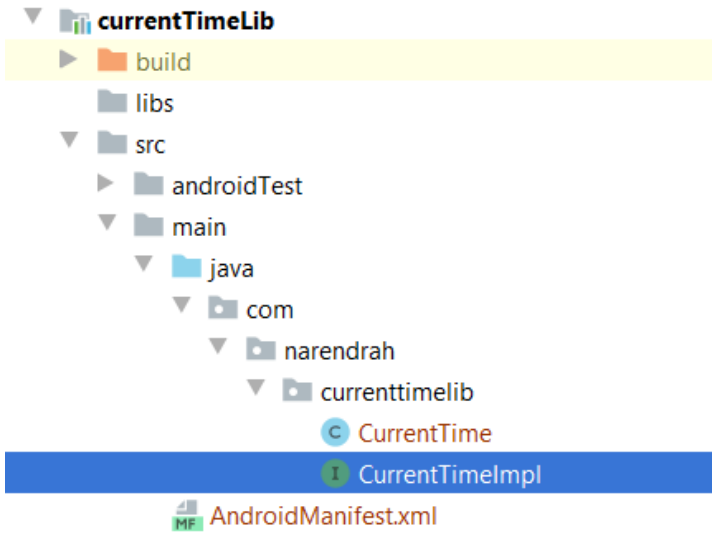


PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

6.2 : Créez une interface pour avoir un callback dans MainActivity pour recevoir l'heure actuelle.



Nous avons maintenant les classes suivantes et le code de ces classes ressemble à celui présenté ci-dessous. Pour la sortie finale, veuillez ajouter le code dans les fichiers respectifs.

Remarques

Veuillez lire les commentaires associés aux fonctions pour comprendre le code.

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

CurrentTime :

```
import java.util.*
/**
* This class is implementing
* the @[Runnable] interface.
*/
class CurrentTime(ct: CurrentTimeImpl) : Runnable {
/**
* The @[CurrentTimeImpl] Interface instance.
* This interface calls a method setCurrentTimeToView(time)
* The setCurrentTimeToView(time) method
* is implemented in @MainActivity.
*/
private val currentTimeImpl: CurrentTimeImpl
override fun run() {
while (true) {
/**
* The business logic to fetch
* the current time.
*/
try {
Thread.sleep(100)
} catch (e: InterruptedException) {
e.printStackTrace()
}
```

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

CurrentTime :

```
val currentTime = Calendar.getInstance().time
val time = currentTime.toString()
/**
* The call back to MainActivity
* with current time.
*/
currentTimeImpl.setCurrentTimeToView(time)
}
}

/**
* @param ct The activity instance will
* be passed and init from @MainActivity
*/
init {
currentTimeImpl = ct
}
}
```

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

CurrentTimeImpl :

```
interface CurrentTimeImpl {
/**
* This method will be called by Current time
* class on every time current time is fetched.
*
* @param time the current time
*/
fun setCurrentTimeToView(time: String?)
}
```

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

PARTIE 5

Solution 1 :

MainActivity :

```
import android.view.View
import com.narendrah.currenttimelib.CurrentTime

class MainActivity : AppCompatActivity(), CurrentTimeImpl {
private var tvTime: TextView? = null
protected override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
tvTime = findViewById<TextView>(R.id.tvTimeText)
}

fun getCurrentTime(view: View?) {
val thread = Thread {

val currentTime = CurrentTime(this@MainActivity)

currentTime.run()
}
thread.start()
}

fun setCurrentTimeToView(time: String?) {
runOnUiThread(Runnable {
tvTime.setText(time)
})
}
}
```

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

Solution 1 :

activity_main.xml :

```
<?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:id="@+id/tvTimeText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Time Text"
      android:textSize="22dp"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintHorizontal_bias="0.485"
```

PARTIE 5

Activité 2 :
Créer un plugin Kotlin réutilisable et personnalisé

PARTIE 5

Solution 1 :

activity_main.xml :

```
app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.356"
    android:textColor="@color/colorPrimary"/>
  <Button  android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="getCurrentTime"
    android:text="Get Time"
    tools:ignore="MissingConstraints"
    tools:layout_editor_absoluteX="161dp"
    tools:layout_editor_absoluteY="420dp"
    app:layout_constraintTop_toBottomOf="@+id/tvTimeText"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginTop="30dp"
    android:textColor="@color/colorPrimary"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```