Institut Spécialisé en Nouvelles Technologies de l'Information et de la Communication

Béni Mellal

M202
Programmation KOTLIN

Yassine AFOUDI

yassine.afoudi@ofppt.ma

2024-2025

Objectifs – acquis d’apprentissage

- **Découvrir les fondamentaux de Kotlin**
- **S’initier à la programmation Kotlin**
- **Maîtriser les fonctions et lambdas**
- **Maîtriser les aspects avancés de Kotlin**
- **Utiliser les Outils Android et kotlin**

90 Heures

Partie 1:

Découvrir les fondamentaux de Kotlin

CHAPITRE N°1
INTRODUIRE LES DIFFÉRENCES MAJEURES AVEC JAVA

Ce que vous allez apprendre dans ce chapitre :

1. Découvrir les fonctionnalités de Kotlin
2. Comprendre l'interopérabilité de Kotlin avec Java

CHAPITRE N°1
INTRODUIRE LES DIFFÉRENCES MAJEURES AVEC JAVA

PLAN

1. **Généralités sur les fonctionnalités KOTLIN**
2. Communication entre JAVA et Kotlin
3. Interopérabilité avec Java

Généralités sur les fonctionnalités KOTLIN

Qu'est ce que Kotlin ?

• Kotlin est un langage de programmation open source, orienté objet et fonctionnel, avec un typage statique, qui cible la machine virtuelle Java (JVM), Android, JavaScript et le code natif. Kotlin a été développé sous la licence Apache 2.0, qui est une licence open-source permissive.

• Il est développé par JetBrains, par une équipe de programmeurs basée à Saint-Pétersbourg en Russie. Le nom a été inspiré du nom de l'île de Kotline, située près de Saint-Pétersbourg.

• Le projet a démarré en 2010 et était open source dès le début. La première version officielle 1.0 a été publiée en février 2016.

• Kotlin supporte le développement des solutions backend, frontend et le développement mobile.

Généralités sur les fonctionnalités KOTLIN

Applications développées avec Kotlin

Kotlin est le langage officiel pour le développement mobile, Actuellement plusieurs applications sont développé avec Kotlin.

Par exemple :


Pinterest

Uber


Coursera


Atlassian | Trello

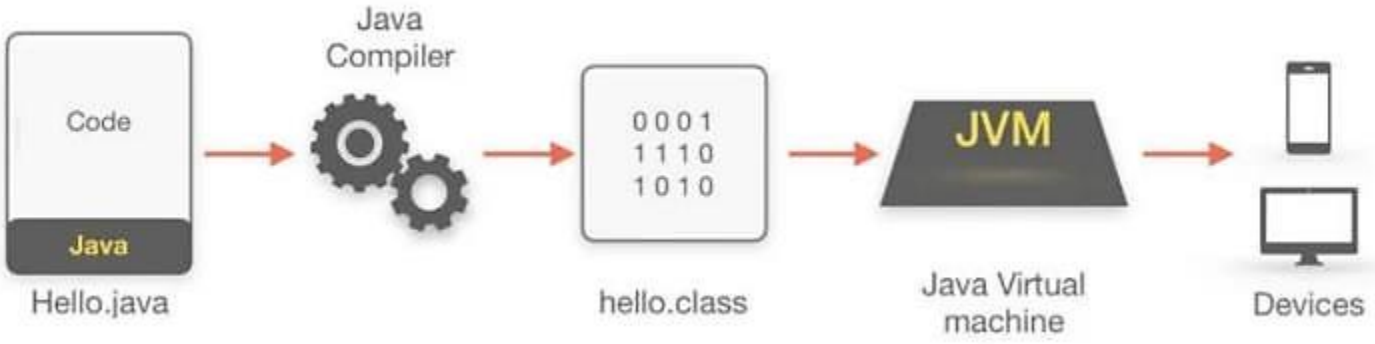

Corda


Evernote

Généralités sur les fonctionnalités KOTLIN

Kotlin est-il un langage orienté objet ou un langage fonctionnel ?

- Kotlin est à la fois un langage orienté objet et un langage fonctionnel. Vous pouvez programmer en orienté objet ou en procédural / fonctionnel. Il est aussi possible de construire une application avec un mixte de ces deux styles de programmation.
- **L'exemple suivant montre comment le code est compilé par le Java compiler et le JVM :**



Questions / Réponses

JAVA Vs KOTLIN

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Concision du code

- La comparaison d'une classe Java avec une classe Kotlin équivalente démontre la concision du code Kotlin. Pour effectuer la même opération que la classe Java, une classe Kotlin nécessite moins de code.
- **Exemple :** En Java, la création d'une classe avec des getters, setters, et une méthode toString() nécessite beaucoup de code, tandis qu'en Kotlin, une simple data class suffit.

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

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Concision du code

- Kotlin nécessite relativement moins de ligne de code que Java, ce qui le rend plus lisible et compréhensible.
- Par exemple, le 'Switch case' en Java a été converti en quelques lignes de code en utilisant ‘when' en Kotlin.

Kotlin

```
when(days)
{
  lundi -> println("premier jour")
  Mardi -> println("deuxième jour")
  Mercredi -> println("deuxième jour")
  else -> println("n'existe pas ")
}
```

JAVA

```
switch(days)
{
  case lundi :
  System.out.println("premier jour");
  break;
  case mardi :
  System.out.println("deuxième jour");
  break;
  case mardi :
  System.out.println("troisième jour");
  break;
  default:
  System.out.println("n'existe pas ");
  break;
}
```

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Coroutines

- Les travaux intensifs du CPU et les E/S du réseau sont des opérations de longue haleine. Le thread appelant est bloqué jusqu'à la fin de l'opération. Comme Android est monofil par défaut, l'interface utilisateur d'une application est complètement gelée dès que le fil principal est bloqué.
- La solution traditionnelle à ce problème en Java consiste à créer un thread d'arrière-plan pour les travaux longs ou intensifs. Cependant, la gestion de plusieurs threads entraîne une augmentation de la complexité ainsi que des erreurs dans le code.
- En **Kotlin**, les **coroutines** sont une manière simple et efficace de gérer l'exécution asynchrone et non bloquante. Elles permettent d'exécuter des tâches de longue durée (comme les opérations réseau ou les opérations d'entrée/sortie) sans bloquer le thread principal.

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Fonctions d'extension

- Kotlin permet d'ajouter des fonctions à des classes existantes (même celles des bibliothèques Java) sans les modifier, grâce aux extensions.
- Ces fonctions, bien que disponibles dans d'autres langages de programmation comme C#, ne sont pas disponibles en Java.
- La création d'une fonction d'extension est facile en Kotlin. Il suffit de préfixer le nom de la classe qui doit être étendue au nom de la fonction créée.
- **Exemple:**

```
fun main() {
  val str1 = "I"
  val str2 = "love"
  val str3 = "Kotlin"
  print(str1.add(str2, str3))
}
fun String.add(str2: String, str3:String): String{
  return this + str2 + str3
}
```

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Support natif de la délégation

• En Kotlin, le support natif de la délégation permet de déléguer certaines responsabilités ou comportements d'une classe à un autre objet. C'est une fonctionnalité puissante qui facilite la composition et réutilisation de code, en permettant à une classe d'utiliser les méthodes et propriétés d'une autre classe sans avoir à les réimplémenter.

• La délégation de classe est une alternative à l'héritage dans Kotlin.

• Elle permet d'utiliser des héritages multiples. En outre, les propriétés déléguées de Kotlin empêchent la duplication du code.

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Champs non privés

- L'encapsulation est essentielle dans tout programme pour atteindre un niveau souhaitable de maintenabilité.
- Un **champ non privé** fait référence à n'importe quelle propriété ou membre de classe qui **n'est pas déclaré private**. Cela inclut les champs avec les modificateurs de visibilité suivants :
- **public** (par défaut) : Accessible partout dans le programme.
- **internal** : Accessible seulement à l'intérieur du même module.
- **protected** : Accessible dans la classe où il est défini et dans ses sous-classes, mais pas en dehors.

Donc, tous les champs publics sont non privés, mais tous les champs non privés ne sont pas nécessairement public

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin : Sécurité de la valeur Nulle

- Kotlin intègre un système de sécurité des nulls qui aide à éviter les erreurs courantes liées aux références nulles, comme les NullPointerException(NPE) en Java.
- Contrairement à Java, tous les types sont non nuls par défaut dans Kotlin. Si les développeurs essaient d'assigner ou de retourner des valeurs nulles dans le code Kotlin, ils échoueront au moment de la compilation. Cependant, il existe un moyen de contourner ce problème. Afin d'assigner une valeur nulle à une variable en Kotlin, il est nécessaire de marquer explicitement cette variable comme nullable.

Généralités sur les fonctionnalités KOTLIN

Java vs Kotlin

Paramètres	JAVA	Kotlin
Facile à utiliser	Difficile	Simple
Performance	Presque pareil	Haut niveau
Popularité	Haut niveau	Haut niveau
Évolutivité	Niveau modéré	Haut niveau
Communauté	Haut niveau	Haut niveau
Cross platform	Plate-forme limitée	Plusieurs plate-forme
Documentation	Haut niveau	Haut niveau

CHAPITRE N°1
INTRODUIRE LES DIFFÉRENCES MAJEURES AVEC JAVA

PLAN

1. Généralités sur les fonctionnalités KOTLIN
2. **Communication entre JAVA et Kotlin**
3. Interopérabilité avec Java

Communication entre JAVA et Kotlin

Communication entre JAVA et Kotlin

- La communication entre Java et Kotlin est l'un des aspects les plus importants dans le développement Android moderne, car il permet une interopérabilité fluide entre les deux langages. Kotlin a été conçu pour fonctionner de manière transparente avec le code Java, ce qui signifie que vous pouvez appeler du code Java depuis Kotlin, et inversement, sans trop d'efforts.

Communication entre JAVA et Kotlin et Interopérabilité avec Java

Appeler du code Java depuis Kotlin

Kotlin est totalement compatible avec Java, ce qui signifie que vous pouvez facilement utiliser des classes, méthodes, et autres éléments Java dans votre code Kotlin.

- **Exemple :**

Supposons que vous avez une classe Java simple avec une méthode :

```
public class JavaClass {
public String getGreeting()
        {
        return "Hello from Java!";
        }
}
```

Communication entre JAVA et Kotlin et Interopérabilité avec Java

Appeler du code Java depuis Kotlin

- **Exemple :**

Vous pouvez appeler cette méthode dans Kotlin de manière très simple :

```
fun main() {
    val javaObj = JavaClass()
    println(javaObj.greeting)  // Sortie : Hello from Java!
}
```

Kotlin traite automatiquement les getters et setters Java comme des propriétés, donc getGreeting() devient greeting en Kotlin.

Communication entre JAVA et Kotlin et Interopérabilité avec Java

Appeler du code Kotlin depuis Java

Lorsque vous avez du code Kotlin, vous pouvez également l'appeler depuis Java. Cependant, il y a quelques différences à prendre en compte, notamment liées aux fonctionnalités spécifiques de Kotlin comme les null safety, data classes, et les fonctions d'extension.

- **Exemple :**

Voici une simple classe Kotlin avec une méthode :

```kotlin
// Kotlin
class KotlinClass {
    fun sayHello(): String {
        return "Hello from Kotlin!"
    }
}
```

Communication entre JAVA et Kotlin et Interopérabilité avec Java

Appeler du code Kotlin depuis Java

- **Exemple :**

Vous pouvez l'appeler depuis du code Java :

```
// Java
public class Main {
    public static void main(String[] args) {
        KotlinClass kotlinObj = new KotlinClass();
        System.out.println(kotlinObj.sayHello());  // Sortie : Hello from Kotlin!
    }
}
```

CHAPITRE N°1
INTRODUIRE LES DIFFÉRENCES MAJEURES AVEC JAVA

PLAN

1. Généralités sur les fonctionnalités KOTLIN
2. Communication entre JAVA et Kotlin
3. **Interopérabilité avec Java**

Interopérabilité avec Java

Interopérabilité avec Java

- Lorsque Kotlin a été développé, il fonctionnait uniquement sur JVM , il fournit donc un ensemble complet de fonctionnalités qui facilitent l'appel de Kotlin depuis Java.
- Par exemple, les objets des classes Kotlin peuvent être facilement créés et leurs méthodes peuvent être invoquées dans les méthodes Java. Cependant, il existe certaines règles sur la façon dont le code Kotlin peut être utilisé en Java .
- **Exemple**

Exemple de null safety

Interopérabilité avec Java

Fonctions au niveau du package

Toutes les fonctions définies dans un fichier Kotlin, au sein d'un package, sont compilées en méthodes statiques en Java au sein d'une classe dont le nom de classe est une combinaison du nom du package et du nom du fichier.

Par exemple, s'il existe un package nommé kotlinPrograms et un fichier Kotlin nommé firstProgram.kt avec le contenu suivant

```
package kotlinPrograms
class myClass {
fun add(val a:Int, val b:Int): Int {
  return a + b;
  }
}
```

Cette fonction peut être invoquée en Java à l'aide de la syntaxe suivante :

```
new kotlinPrograms.firstProgram.myClass()
kotlinPrograms.firstProgramkt.add(3,5);
```

Interopérabilité avec Java

Fonctions au niveau du package

Nous pouvons changer le nom de la classe Java générée en utilisant l'annotation **@JvmName**.

```
//Kotlin file
@file: Jvmname("Sample")
package kotlinPrograms
class myClass {
   fun add(val a: Int, val b: Int): Int {
       return a + b;
   }
}
```

Cette fonction peut être invoquée en Java à l'aide de la syntaxe suivante :

```
//JAVA
new kotlinPrograms.firstProgram.myClass();
kotlinPrograms.Sample.add(3,5);
```

Interopérabilité avec Java

Question 1:

Kotlin est développé par ?

a) Google

b) JetBrains

c) Microsoft

d) Adobe

Interopérabilité avec Java

Question 1:

Kotlin est développé par ?

a) Google

b) JetBrains

c) Microsoft

d) Adobe

Interopérabilité avec Java

Question 2:

Pouvons-nous utiliser Kotlin dans le langage de programmation Java ?

a) Oui

b) Non

Interopérabilité avec Java

Question 2:

Pouvons-nous utiliser Kotlin dans le langage de programmation Java ?

a) Oui

b) Non

Interopérabilité avec Java

Question 3:

Pouvons-nous exécuter le code Kotlin sans Jvm ?

a) Oui

b) Non

Interopérabilité avec Java

Question 3:

Pouvons-nous exécuter le code Kotlin sans Jvm ?

a) Oui

b) Non

Interopérabilité avec Java

Question 4:

Kotlin a-t-il le mot-clé statique ?

a) Oui

b) Non

Interopérabilité avec Java

Question 4:

Kotlin a-t-il le mot-clé statique ?

a) Oui

b) Non

Interopérabilité avec Java

Question 5:

Kotlin a été développé sous la licence ..... ?

a) Apache 1.0

b) Apache 2.0

c) Apache 1.1

d) Aucune de ces réponses

Interopérabilité avec Java

Question 5:

Kotlin a été développé sous la licence ..... ?

a) Apache 1.0

b) Apache 2.0

c) Apache 1.1

d) Aucune de ces réponses

Interopérabilité avec Java

Question 6:

Quelle est la méthode pour imprimer un message à la console en Kotlin ?

a) println()

b) echo()

c) write()

Interopérabilité avec Java

Question 6:

Quelle est la méthode pour imprimer un message à la console en Kotlin ?

a) println()

b) echo()

c) write()

Interopérabilité avec Java

Question 7:

Comment déclare-t-on une fonction en Kotlin ?

a) function myFunction() {}

b) def myFunction() {}

c) fun myFunction() {}

d) method myFunction() {}

Interopérabilité avec Java

Question 7:

Comment déclare-t-on une fonction en Kotlin ?

a) function myFunction() {}

b) def myFunction() {}

c) fun myFunction() {}

d) method myFunction() {}

Interopérabilité avec Java

Question 8:

Quel mot-clé est utilisé pour définir une classe en Kotlin ?

a) create

b) class

c) new

d) define

Interopérabilité avec Java

Question 8:

Quel mot-clé est utilisé pour définir une classe en Kotlin ?

a) create

b) class

c) new

d) define