Programmation et interfaces Android

Introduction à Kotlin

Sommaire

1. Kotlin qu’est ce que c’est ?
2. Kotlin VS Java
3. La syntaxe
4. Un peu d'exercice
5. Un peu plus d’exercice

Kotlin qu’est ce que c’est ?

- Un langage de programmation orienté objets et fonctionnel

Kotlin qu’est ce que c’est ?

- Un langage de programmation orienté objets et fonctionnel
- Développé par JetBrains (IntelliJ)

Kotlin qu’est ce que c’est ?

- Un langage de programmation orienté objets et fonctionnel
- Développé par JetBrains (IntelliJ)
- Il utilise la machine virtuelle Java (JVM)

Kotlin qu’est ce que c’est ?

- Un langage de programmation orienté objets et fonctionnel
- Développé par JetBrains (IntelliJ)
- Il utilise la machine virtuelle Java (JVM)
- Utilisé pour faire de l’Android, ou pour développer des applications Back End (Côté serveur)

Kotlin qu’est ce que c’est ?

- Un langage de programmation orienté objets et fonctionnel
- Développé par JetBrains (IntelliJ)
- Il utilise la machine virtuelle Java (JVM)
- Utilisé pour faire de l’Android, ou pour développer des applications Back End (Côté serveur)
- Peut s’utiliser conjointement avec Java

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Tout d’abord l'inférence de type (le typage est déduit par la valeur)

```kotlin
fun main(){
    var a = 2
    a = "un texte"  // Erreur : type mismatch: inferred type is String but Int was expected
}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Tout d'abord l'inférence de type (le typage est déduit par la valeur)

```
fun main(){
    var a = 2
    a = "un texte"  // Erreur : type mismatch: inferred type is String but Int was expected
}
```

Tout en maintenant la possibilité de définir un type :

```
fun main(){
    var a : Int = 2
}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"

```
fun main(){
    var a : Int? = null //Cette variable peut - être NULL et le sera tant qu'elle n'aura pas été affectée

}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"

```
fun main(){
    var a : Int? = null //Cette variable peut – être NULL et le sera tant qu'elle n'aura pas été affectée

    var b : Int = 5

    var c : Int = null //Erreur de compilation
}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"
  - fini les **if(a == null){}** pour tester si une variable est null, grâce à 2 opérateur on peut aisément jouer avec une variable **null**

```
fun main(){
    var maVoiture : Voiture? = null //Cette variable peut - être NULL et le sera tant qu'elle n'aura pas été affectée

    maVoiture?.roule()
    //Si l'objet maVoiture de type Voiture à été instancié alors la fonction roule() est joué
    //sinon la ligne sera ignoré
}
```

L'opérateur **?.** permet d'accéder à une propriété / méthode de l'objet si il a été instancié sinon retourne **null**

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"
  - fini les **if(a == null){}** pour tester si une variable est null, grâce à 2 opérateur on peut aisément jouer avec une variable **null**

```
fun main(){
    var a : Int? = null //Cette variable peut - être NULL et le sera tant qu'elle n'aura pas été affectée

    println(a?.toString() ?: "INCONNU")
}
```

L'opérateur **?:** permet d'assigner une réponse par défaut si une réponse null est reçu de l'objet

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"
  - fini les **if(a == null){}** pour tester si une variable est null, grâce à 2 opérateur on peut aisément jouer avec une variable **null**

```
fun main(){
    var a : Int? = null //Cette variable peut – être NULL et le sera tant qu'elle n'aura pas été affectée

    println(a!!) //NullPointerException

}
```

L'opérateur **!!** permet de forcer le cast vers un type non-null mais peut causer un null pointer exception si l'objet est effectivement null

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - Il apporte la nullabilité grâce à l'opérateur "?"
  - fini les **if(a == null){}** pour tester si une variable est null, grâce à 2 opérateur on peut aisément jouer avec une variable **null**

```
fun main(){
    var a : Int? = null //Cette variable peut – être NULL et le sera tant qu'elle n'aura pas été affectée

    var b : Int = 2

    a?.let{
        b = a
    }

}
```

L'opérateur **?.let{}** permet d'exécuter du code uniquement si l'objet n'est pas NULL

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - il est facile de concaténer des chaînes de caractère grâce à **${}**

```
fun main(){
    var firstname = "Benjamin"
    var lastname = "Metaut"

    println("${firstname} ${lastname}")
    //affiche Benjamin Metaut

}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - le smart cast permet tout d'abord un cast simple avec **as**

```
fun main(){
    var name:Any = "Benjamin"

    println((name as String).toUpperCase())

}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - le smart cast permet aussi un cast nullable (qui retourne **null** si le cast échoue)

```
fun main(){
    var name:Any = "Benjamin"

    println((name as? Int)?.plus(2))
    //Affiche null

}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - le smart cast permet de déduire le type grâce à **is** qui remplace le **instanceof** de java

```
fun main(){
    var name:Any = "Benjamin"

    if(name is String){
        println(name.toUpperCase()) //Ici pas besoin de reprecisé que name est de type String
    }

}
```

Kotlin VS Java

- Kotline apporte plusieurs chose que Java ne permet pas :
  - il permet d'ajouter des fonctionnement à des types natif grâce aux extensions !

```kotlin
fun String.toUpperCaseOnlyIfBenjamin() : String{
    if(this == "Benjamin"){
        return this.toUpperCase()
    }
    return this
}

fun main(){
    var name:String = "Benjamin"

    var name2:String = "Jean"

    println(name.toUpperCaseOnlyIfBenjamin()) //BENJAMIN
    println(name2.toUpperCaseOnlyIfBenjamin()) //Jean


}
```

Kotlin VS Java

- Mais surtout Kotlin apporte un allègement du code (moins de code pour autant de résultat)

```java
public class JavaPerson {

    private final int id;
    private String firstname;
    private String lastname;
    private int age;

    public JavaPerson(final int id, String firstname, String lastname, int age){
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
    }

    public int getId(){
        return this.id;
    }

    public String getFirstName(){
        return this.firstname;
    }

    public String getLastName(){
        return this.lastname;
    }

    public int getAge(){
        return age;
    }

    public void setFirstName(String firstname){
        this.firstname = firstname;
    }

    public void setLastName(String lastname){
        this.lastname = lastname;
    }

    public void setAge(int age){
        this.age = age;
    }

}
```

Kotlin VS Java

- Mais surtout Kotlin apporte un allègement du code (moins de code pour autant de résultat)

```java
public class JavaPerson {

    private final int id;
    private String firstname;
    private String lastname;
    private int age;

    public JavaPerson(final int id, String firstname, String lastname, int age){
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
        this.age = age;
    }

    public int getId(){
        return this.id;
    }

    public String getFirstName(){
        return this.firstname;
    }

    public String getLastName(){
        return this.lastname;
    }

    public int getAge(){
        return age;
    }

    public void setFirstName(String firstname){
        this.firstname = firstname;
    }

    public void setLastName(String lastname){
        this.lastname = lastname;
    }

    public void setAge(int age){
        this.age = age;
    }
}
```

```kotlin
class KotlinPerson(
    val id : Int,
    var firstname : String,
    var lastname : String,
    var age : Int)
```

Une classe Java vs la même classe en Kotlin

La syntaxe

- Voici les petits détails propre à la syntaxe de Kotlin

La syntaxe

- Voici les petits détails propre à la syntaxe de Kotlin
  - les fichiers kotlin termine par **.kt**
  - pas de **;**
  - les variable commence par **var** et les variable immutable (non modifiable) par **val**
  - les fonction sont déclaré par **fun**
  - pas de **new** pour instancier un objet
  - extension de classe avec **:** plutôt que extend en java

La syntaxe

- Les structures de contrôle change un peu avec Kotlin
  - le **if** ne change pas

La syntaxe

- Les structures de contrôle change un peu avec Kotlin
  - les boucles **for** sur tout type d'objet itérables

```
fun main(){
    val mesAmis = listOf("Jean", "Luc", "Marc")

    for(i in mesAmis){
        println(i)
    }
    //affiche Jean Luc Marc

}
```

```
fun main(){

    for(i in 0 until 10){
        println(i)
    }
    //affiche 1 2 3 4 5 6 7 8 9

}
```

La syntaxe

- Les structures de contrôle change un peu avec Kotlin
  - le **switch** est remplacé par le **when**

```
fun main(){
    val maVal = 2

    when(maVal){
        1 -> println("C'est 1")
        2 -> println("C'est 2")
        else -> println(" c'est autre chose")
    }

}
```

La syntaxe

- Les énumération sont similaire à java représenté par des classes

```
enum class COLOR{
    BLUE, RED, GREEN
}

fun main() {
    val myColor = COLOR.RED

}
```

Un peu d'exercice

- Avec tout ça on a assez d’information pour faire un petit exercice d'entraînement

Un peu d'exercice

- Avec tout ça on a assez d’information pour faire un petit exercice d'entraînement
  - Faire un programme qui simule un portique de fac il :
  - Contient une énumération **STATUS** avec les attributs suivant :
    - STUDENT
    - PROFESSOR
    - OTHER
  - Contient une classe **Person** avec les attributs suivant :
    - firstname de type String immutable
    - lastname de type String immutable
    - age de type Optionnel Int immutable
    - status de type STATUS immutable
    - isAuthorized de type Boolean mutable
    - une methode : isMajor() qui retourne un boolean (true si age est >= à 18 et false si age < 18)
  - Contient une extension du type **Int** isMajor() qui retourne un boolean (true si l’Int est >= 18 et false si l’Int est < 18)
  - Le **main** doit :
    - Instancier une personne avec les attributs de votre choix
    - Vérifier que la personne est majeur et afficher un message en conséquence (si la personne est mineur isAuthorized est remis à false)
    - Vérifier que la personne est un professeur ou un élève et afficher un message en conséquence (elle doit aussi remettre à false l’attribut isAuthorized si la personne est du status OTHER)
    - Afficher un message final si la personne est autorisé à rentrer ou non

Un peu plus d'exercice

- Avec tout ça on a assez d’information pour faire un petit exercice d'entraînement
  - modifier le programme qui simule un portique de fac :
  - La classe **Person** n’a désormais plus d’attribut **isAuthorized()**
  - Contient une fonction qui permettra de détecter si une personne est valide
    - Elle reçoit un objet de type personne et retourne un boolean
    - Vérifier que la personne est majeur si non afficher un message et renvoie faux
    - Vérifier que la personne est un professeur ou un élève et afficher un message en conséquence si la personne n’est aucun des deux elle renvoie faux
  - Le **main** doit :
    - Instancier une liste mutable de personne vide (elle contiendra les personnes autorisé)
    - Instancier une liste immutable de personne avec des personnes dedans
    - Vérifier pour chaque personne si celle ci est authorisé à rentrer (si oui cette personne doit désormais faire partie de la liste des personnes autorisé crée au préalable)
    - Afficher la liste de chaque personne authorisé à rentré dans la fac