DUT : Génie Informatique

Cours Kotlin

Pr. Lamia ZIAD
École Supérieure de Technologie d'Essaouira

Table des matières

1 Introduction à Kotlin

1.1 Qu'est-ce que Kotlin ?

Kotlin est un langage de programmation moderne, concis et sécurisé développé par JetBrains. Il est :

- **Interopérable** : Fonctionne parfaitement avec Java
- **Concis** : Réduit la verbosité du code
- **Sécurisé** : Évite les erreurs courantes (NullPointerException)
- **Multiplateforme** : Peut être utilisé sur Android, Web, Desktop, etc.

1.2 Configuration de l'environnement

```
// Installation avec SDKMAN!
sdk install kotlin

// Verification
kotlin -version
```

2 Les Bases de Kotlin

2.1 Hello World

```
fun main() {
    println("Hello, World!")
}
```

2.2 Variables et Types

```
// Variables immuables (recommande)
val nom: String = "Alice"
val age: Int = 25

// Inference de type
val ville = "Paris"  // Type String infere

// Variables mutables
var compteur = 0
compteur = 10

// Types de base
val nombre: Int = 42
val decimal: Double = 3.14
val bool: Boolean = true
val caractere: Char = 'A'
val texte: String = "Kotlin"
```

2.3 Null Safety

```
// Variable non-nullable (ne peut pas etre null)
val nom: String = "Alice"

// Variable nullable (peut etre null)
val nomNullable: String? = null

// Safe call operator
val longueur = nomNullable?.length

// Elvis operator
val longueurSafe = nomNullable?.length ?: 0

// Assertion non-null (a utiliser avec precaution)
val longueurForced = nomNullable!!.length
```

3 Structures de Contrôle

3.1 Conditions

```
// If-else classique
val note = 85
if (note >= 90) {
    println("Excellent")
} else if (note >= 70) {
    println("Bon")
} else {
    println("A ameliorer")
}

// If comme expression
val resultat = if (note >= 50) "Reussi" else "Echoue"
```

3.2 When (switch amélioré)

```
val jour = 3
val nomJour = when (jour) {
    1 -> "Lundi"
    2 -> "Mardi"
    3 -> "Mercredi"
    4 -> "Jeudi"
    5 -> "Vendredi"
    in 6..7 -> "Weekend"
    else -> "Jour invalide"
}

// When avec plusieurs conditions
when {
    note >= 90 -> println("A")
    note >= 80 -> println("B")
    note >= 70 -> println("C")
    else -> println("D")
}
```

4 Fonctions

4.1 Déclaration de fonctions

```
// Fonction simple
fun direBonjour(nom: String) {
    println("Bonjour $nom")
}

// Fonction avec retour
fun additionner(a: Int, b: Int): Int {
    return a + b
}

// Expression function (une seule ligne)
fun multiplier(a: Int, b: Int): Int = a * b

// Fonction avec parametres par defaut
fun saluer(nom: String, message: String = "Bonjour") {
    println("$message $nom")
}
```

4.2 Fonctions d'extension

```
// Ajouter une fonction a la classe String
fun String.estEmailValide(): Boolean {
    return this.contains("@") && this.contains(".")
}

// Utilisation
val email = "test@example.com"
println(email.estEmailValide()) // true
```

5 Collections

5.1 Listes

```
// Liste immuable
val noms = listOf("Alice", "Bob", "Charlie")

// Liste mutable
val nombres = mutableListOf(1, 2, 3)
nombres.add(4)

// Operations sur les listes
val nombresPairs = nombres.filter { it % 2 == 0 }
val nombresDoubles = nombres.map { it * 2 }
val somme = nombres.reduce { acc, n -> acc + n }
```

5.2 Tableaux et Maps

```
// Tableaux
val tableau = arrayOf(1, 2, 3, 4, 5)
```

```
3
4  // Maps
5  val ages = mapOf(
6      "Alice" to 25,
7      "Bob" to 30,
8      "Charlie" to 35
9  )
10
11 // Map mutable
12 val capitales = mutableMapOf(
13     "France" to "Paris",
14     "Allemagne" to "Berlin"
15 )
16 capitales["Espagne"] = "Madrid"
```

6 Programmation Orientée Objet

6.1 Classes et Objets

```
1  // Classe simple
2  class Personne(val nom: String, var age: Int) {
3      fun saluer() {
4          println("Bonjour, je m'appelle $nom")
5      }
6  }
7
8  // Utilisation
9  val personne = Personne("Alice", 25)
10 personne.saluer()
11 personne.age = 26
```

6.2 Propriétés et Getters/Setters

```
1  class Rectangle(val largeur: Int, val hauteur: Int) {
2      val surface: Int
3          get() = largeur * hauteur
4
5      var couleur: String = "blanc"
6          set(value) {
7              if (value.isNotBlank()) {
8                  field = value
9              }
10         }
11 }
```

6.3 Héritage

```
1  // Classe ouverte (peut etre heritee)
2  open class Animal(val nom: String) {
3      open fun faireDuBruit() {
4          println("L'animal fait du bruit")
5      }
6  }
7
```

```
class Chien(nom: String) : Animal(nom) {
    override fun faireDuBruit() {
        println("$nom aboie: Wouf Wouf!")
    }
}
```

7 Programmation Fonctionnelle

7.1 Lambdas et Fonctions d'ordre supérieur

```
// Lambda simple
val addition = { a: Int, b: Int -> a + b }

// Fonction prenant une lambda en parametre
fun operation(a: Int, b: Int, op: (Int, Int) -> Int): Int {
    return op(a, b)
}

// Utilisation
val resultat = operation(5, 3) { x, y -> x * y }
```

7.2 Scope Functions

```
data class Utilisateur(val nom: String, var age: Int)

val user = Utilisateur("Alice", 25)

// let - execute un bloc sur un objet non-null
user.let {
    println("Nom: ${it.nom}")
    println("Age: ${it.age}")
}

// apply - configure un objet
val newUser = Utilisateur("Bob", 30).apply {
    age = 31
}

// with - execute un bloc avec l'objet comme receiver
with(user) {
    println("$nom a $age ans")
}
```

8 Coroutines (Programmation Asynchrone)

8.1 Introduction aux Coroutines

```
import kotlinx.coroutines.*

// Fonction suspendue
suspend fun fetchData(): String {
    delay(1000) // Simulation d'une operation longue
    return "Donnees recuperees"
```

```
}

// Utilisation
fun main() = runBlocking {
    println("Debut")

    val result = async { fetchData() }

    println("En attente...")
    println(result.await())

    println("Fin")
}
```

8.2 Parallélisme avec les Coroutines

```
suspend fun getTemperature(): Int {
    delay(500)
    return 25
}

suspend fun getHumidity(): Int {
    delay(700)
    return 60
}

fun main() = runBlocking {
    val temps = measureTimeMillis {
        val temperature = async { getTemperature() }
        val humidity = async { getHumidity() }

        println("Temperature: ${temperature.await()}C")
        println("Humidite: ${humidity.await()}%")
    }
    println("Temps total: ${temps}ms")
}
```

9 Exercices Pratiques

9.1 Exercice 1 : Calculatrice

```
fun calculatrice(a: Double, b: Double, operation: String): Double {
    return when (operation) {
        "+" -> a + b
        "-" -> a - b
        "*" -> a * b
        "/" -> if (b != 0.0) a / b else throw IllegalArgumentException("
   Division par zero")
        else -> throw IllegalArgumentException("Operation non supportee")
    }
}
```

9.2 Exercice 2 : Gestion d'étudiants

```
data class Etudiant(val nom: String, val notes: List<Int>) {
    val moyenne: Double
        get() = notes.average()

    fun aReussi(seuil: Int = 50): Boolean {
        return moyenne >= seuil
    }
}

class Promotion {
    private val etudiants = mutableListOf<Etudiant>()

    fun ajouterEtudiant(etudiant: Etudiant) {
        etudiants.add(etudiant)
    }

    fun meilleurEtudiant(): Etudiant? {
        return etudiants.maxByOrNull { it.moyenne }
    }
}
```

9.3 Exercice 3 : API Simulée

```
// Simulation d'appel API
suspend fun fetchUserData(userId: Int): String {
    delay(2000) // Simulation de latence reseau
    return "Donnees de l'utilisateur $userId"
}

// Gestion des erreurs
suspend fun fetchUserDataSafe(userId: Int): Result<String> {
    return try {
        Result.success(fetchUserData(userId))
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

10 Bonnes Pratiques

10.1 Conseils de Développement

- **Préférez `val` à `var`** : L'immutabilité rend le code plus prévisible
- **Utilisez les data classes** : Pour les classes qui stockent principalement des données
- **Évitez `!!`** : Préférez les safe calls ou le Elvis operator
- **Utilisez les scope functions** : Pour un code plus lisible et concis
- **Coroutines pour l'asynchrone** : Évitez les callbacks complexes

10.2 Motifs de Conception Courants

```
// Singleton
object DatabaseManager {
```

```
    fun connect() { /* ... */ }
}

// Builder pattern avec apply
class Config {
    var host: String = "localhost"
    var port: Int = 8080
}

val config = Config().apply {
    host = "example.com"
    port = 9000
}
```

11 Annexe : Cheatsheet Rapide

Concept	Syntaxe Kotlin
Variable immuable	`val x = 5`
Variable mutable	`var x = 5`
Fonction	`fun nom() { ... }`
Classe	`class Nom(val prop: Type)`
Condition	`if (condition) { ... } else { ... }`
Boucle	`for (item in collection) { ... }`
Lambda	`{ x -> x * 2 }`
Null safety	`val len = text?.length ?: 0`