Introduction To Kotlin

Monday, 1 June 2020 11:54 PM

- Developed by jetbrains in 2011.
- Code compiles to java bytecode
- Interopable with Java

★ **Mutability** ⇒ In java we have an optional "final" but in Kotlin it is much more robust. We need to clarify the mutability of a variable in declaration itself.

★ **Null safety** ⇒ In kotlin we have to explicitly declare if a variable can be null or not.

Basic data type:

double	64 bit
float	32 bit
long	64 bit
int	32 bit
short	16 bit
byte	8 bit
boolean	8 bits — default – True

In Kotlin there are no primitive data types There are just objects. Although primitive data types work faster and are used under the hood.

Declaring the variables –

<val/var> <varname> : <datatype> = value

- val → can't be reassigned (immutable)
- var → mutable
- datatype → optional

★ **Null safety** →

In kotlin we have 2 types

1. Non nullable – String , Int
2. Nullable – String? , Int?

for nullable type we need to use safe call operator → '?'

Example → `name?.length`
- → if name is not null print length
- → else nothing happens.

// elvis operator — ?:

Example → `name?.length ?: 0`
- → if name is null print 0
- → else print length

★ **Not null assertion operator (!!)**

```
fun printName(firstName: String?, lastName: String?)
{
    if(firstName != null) {
        println("Name: ${firstName.length}
            ${lastName!!.length}")
    }
}
```

if let's say we know if firstName is not null then lastName can't be null, then if we don't use not null assertion operator we will get error because compiler doesn't know about this logic & says lastName can be null.

Type Conversion

```
val a: Int = 10
val b: Int = a.toLong()
```

Type Checking

```
val name: Any = "John"
name is String   ⟶ True
```

Type Casting

```
val name: Any = "John"
val name1: String = name as String

val x: Int = 7
val y: Int = x as Long   //error
```

//error ↓ because we can't cast Int to Long

```
val z: Long? = x as? Long
println(z)
```

- println(z) ↓ prints (null)
- as? ↳ Safe cast operator

Example →

1.
```
val number1: Double = 10.0
val sval: Long = number1
println(sval)
```
⇒ Type Mismatch Error

2.
```
val x: Int = 7
val y: Long? = x as? Long
println(x)
```
⇒ null

★ **Functions**

```
fun add(x: Int, y: Int): Int {
    return x + y
}
```

- add → function name
- x: Int, y: Int → params
- Int → return type

- - - - - - - -

```
fun printUser(name: String): Unit {
    println(name)
}
```

Unit → Default return type, no need to explicitly write

- - - - - - - -

```
fun printUserName(firstName: String = "Doe") {
    println("The name is $firstName")
}
```

- firstName: String = "Doe" → Default params
- $firstName → string interpolation

- - - - - - - -

```
fun main() {
    println(volume("Box", 1, height=10, width=5))
}
fun volume(boxname: String, length: Int, width: Int, height: Int): Int {
    return length * width * height
}
```

height=10, width=5 → named params/args

- - - - - - - -

```
fun addNumbers(x: Int, y: Int) = x + y
```

↳ Single Expression

Q ⇒ Can a function with a return type 'Void' return a null value?

⇒ No → void is a non-nullable type

Arrays

```
val numbers = arrayOf(1,2,3,4)
```

- numbers → array Name
- (1,2,3,4) → Default value

```
println("${numbers[1]} ${numbers.get(0)}")
numbers[1] = 22
numbers.set(3,5)
```

- numbers[1], numbers.get(0) → getters
- numbers[1] = 22, numbers.set(3,5) → setters

- - - - - -

```
val nums = Array(4) {0}
```

- (4) → size
- {0} ↳ default value for all indices

```
val arr = intArrayOf(1,2,3) ; val arr1 = IntArray(5) {i+1}
```

String Comparison

① "==" operators

Used to check if two strings are structurally equal, equivalent to equals method in Java

Example

```
val first = "Kotlin"
val second = "Kotlin"
val third = "KOTLIN"
first == second    // true
second == third    // false
```

② "===" referential equality operator

Return true if the 2 variables are pointing to the same object. Equivalent to == in Java

Example

```
first === second    // true
val fourth = String("Kotlin".toCharArray())
first == fourth     // true
first === fourth    // false
```

③ Comparing with equals

works same as "==" operator.

It has second optional parameter for case-insensitive comparison.

Example

```
first.equals(third, true)   // true
```

④ Comparing with compareTo

Kotlin has a compareTo method which we can use to compare the order of 2 strings. Similarly as the equals method, the compareTo method also comes with an optional ignore case argument.

Example –

```
first.compareTo(second)       // 0
first.compareTo(third)        // 32
third.compareTo(first)        // -32
first.compareTo(third, true)  // 0
```

- // 32 → arguments ascii values are less
- // -32 → arguments ascii values are greater

String Substring

```
str.subSequence(startIdx: Int, endIndex: Int)
```

- startIdx ↓ inclusive
- endIndex ↓ exclusive