Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)
Dr. Siobhan Drohan (sdrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introducing Kotlin Syntax - Part 1.1

Agenda

- ❑ Basic Types
- ❑ Local Variables (val & var)
- ❑ Functions
- ❑ Control Flow (if, when, for, while)
- ❑ Strings & String Templates
- ❑ Ranges (and the *in* operator)
- ❑ Type Checks & Casts
- ❑ Null Safety
- ❑ Comments

Agenda

- ❑ Basic Types
- ❑ Local Variables (val & var)
- ❑ Functions
- ❑ Control Flow (if, when, for, while)
- ❑ Strings & String Templates
- ❑ Ranges (and the *in* operator)
- ❑ Type Checks & Casts
- ❑ Null Safety
- ❑ Comments

Basic Types

Numbers, Characters & Booleans

Basic Types

*In Kotlin, **everything** is an object in the sense that we can call member functions and properties on **any** variable.*

Basic Types – Numbers

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Basic Types – Numbers

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

```
val doubleNumber: Double = 100.45
val floatNumber: Float = 100.45f
val longNumber: Long = 100L
val intNumber: Int = 100
val shortNumber: Short = 100
val byteNumber: Byte = 100
```

Explicitly defining a numeric type

Basic Types – Numbers

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

```
val doubleNumber = 100.45
val floatNumber = 100.45f
val longNumber = 100L
val intNumber = 100
val shortNumber = 100
val byteNumber = 100
```

Type Inference

Basic Types – Numbers

Type Inference

```
val doubleNumber = 100.45
val floatNumber = 100.45f
val longNumber = 100L
val intNumber = 100
val shortNumber = 100
val byteNumber = 100
```

```
Console
<terminated> Config - Main.kt [Java Appl
doubleNumber type is: double
floatNumber type is:  float
longNumber type is:   long
intNumber type is:    int
shortNumber type is:  int
byteNumber type is:   int
```

```
println("doubleNumber type is: " + doubleNumber.javaClass)
println("floatNumber type is:  " + floatNumber.javaClass)
println("longNumber type is:   " + longNumber.javaClass)
println("intNumber type is:    " + intNumber.javaClass)
println("shortNumber type is:  " + shortNumber.javaClass)
println("byteNumber type is:   " + byteNumber.javaClass)
```

Basic Types – Numbers

```
val oneMillion = 1_000_000
val threeThousand = 3_000
val creditCardNumber = 1234_4321_5678_8765

fun main(args : Array<String>)
{
  println("" + oneMillion + " - the type is: " + oneMillion.javaClass)
  println("" + threeThousand + " - the type is: " + threeThousand.javaClass)
  println("" + creditCardNumber + " - the type is: " +
creditCardNumber.javaClass)
}
```

You can use underscores to make number constants more readable.

Console

```
<terminated> Config - Main.kt [Java Application] C:\Progra
1000000 - the type is: int
3000 - the type is: int
1234432156788765 - the type is: long
```

Basic Types – Numbers: Explicit Conversions

- In Kotlin, there are no implicit widening conversions for numbers i.e. smaller types (e.g. Byte) are not subtypes of bigger ones (e.g. Int)

→ smaller types are NOT implicitly converted to bigger types.

Basic Types – Numbers: Explicit Conversions

- In Kotlin, there are no implicit widening conversions for numbers i.e. smaller types (e.g. Byte) are not subtypes of bigger ones (e.g. Int)

→ smaller types are NOT implicitly converted to bigger types.

```
val byteNumber: Byte = 10        //static type check: OK
val intNumber: Int = byteNumber //syntax error
```

BUT, we can use explicit conversions to widen numbers

```
val byteNumber: Byte = 10        //static type check: OK
val intNumber: Int = byteNumber.toInt()   //OK
```

Basic Types – Numbers: Explicit Conversions

Every number type supports the following conversions:

- `toByte(): Byte`
- `toShort(): Short`
- `toInt(): Int`
- `toLong(): Long`
- `toFloat(): Float`
- `toDouble(): Double`
- `toChar(): Char`

```
//Explicit Conversion
val intNumber: Int = byteNumber.toInt()
val floatNumber: Float = byteNumber.toFloat()
```

Basic Types – Characters

```
val aChar = 'a'
val bChar: Char = 'b'

fun main(args : Array<String>)
{
  println("" + aChar + " - the type is: " + aChar.javaClass)
  println("" + bChar + " - the type is: " + bChar.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0
a - the type is: char
b - the type is: char
```

Basic Types – Booleans

```
val aFlag = true
val bFlag: Boolean = false

fun main(args : Array<String>)
{
  println("" + aFlag + " - the type is: " + aFlag.javaClass)
  println("" + bFlag + " - the type is: " + bFlag.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\
true - the type is: boolean
false - the type is: boolean
```

Basic Types – Escape Characters

Special characters can be escaped using a backslash:

\t \b \n \r \’ \“ \\ \$

```
val aFlag= true
val bFlag: Boolean = false

fun main(args : Array<String>) {
 println("" + aFlag + " - the type is: \n\t\t" + aFlag.javaClass)
 println("" + bFlag + " - the type is: \n\t\t" + bFlag.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0_
true - the type is:
                boolean
false - the type is:
                boolean
```

Local Variables

val (read-only) and var (mutable)

Local Variables – val (read-only)

- Defined using the keyword **val**. They can be assigned a value only once.

```
fun main() {
    val a: Int = 1  // immediate assignment
    val b = 2   // `Int` type is inferred
    val c: Int  // Type required when no initializer is provided
    c = 3       // deferred assignment
    println("a = $a, b = $b, c = $c")
}
```

```
a = 1, b = 2, c = 3
```

Local Variables – val (read-only)

- Defined using the keyword **val**. They can be assigned a value only once.

```
1  fun main() {
2      val a: Int = 1  // immediate assignment
3      val b = 2   // `Int` type is inferred
4      val c: Int  // Type required when no initializer is provided
5      c = 3       // deferred assignment
6      c = 4
7      println("a = $a, b = $b, c = $c")
8  }
```

Val cannot be reassigned

Local Variables – var (mutable)

- Variables that can be reassigned use the **var** keyword:

```
fun main() {
    var x = 5 // `Int` type is inferred
    x += 1
    println("x = $x")
}
```

```
x = 6
```

Local Variables – var (mutable)

- Variables that can be reassigned use the **var** keyword:

```
fun main() {
    var x = 5 // `Int` type is inferred
    x += 1
    x = 10
    println("x = $x")
}
```

```
x = 10
```

References

Sources:

http://kotlinlang.org/docs/reference/basic-syntax.html
http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/
https://www.programiz.com/kotlin-programming
https://medium.com/@napperley/kotlin-tutorial-5-basic-collections-3f114996692b


Thanks.