JÖNKÖPING UNIVERSITY

School of Engineering

# KOTLIN

**Peter Larsson-Green** 

Jönköping University

Spring 2020



## DATATYPES

• Same basic datatypes as in Java.

Datatype	Size	Min value	Max value
byte	8 bits	-128	127
short	16 bits	-32 768	32 767
int	32 bits	-2 147 483 648	2 147 483 647
long	64 bits	<b>-</b> 2 <sup>63</sup>	2 <sup>63</sup> -1
float	32 bits		
double	64 bits		
char	16 bits		
boolean			

#### VARIABLES

- In Java, global variables do not exist.
  - Use class variables instead (static instance variables).
- In Kotlin, global variables do exist.

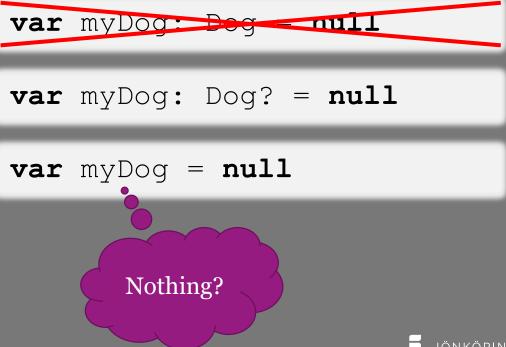
theNumber = 44 theNumber = 44



## VARIABLES

- In Java, we can use **null**.
- In Kotlin, we avoid using **null**.

```
Dog myDog = null;
```





#### VARIABLES

- In Java, we can use **null**.
- In Kotlin, we avoid using **null**.

```
Dog myDog = null;
// ...
myDog.bark();
if(myDog != null) {
  myDog.bark();
}
```

```
var myDog: Dog? = null
// ...
mypog.back()
myDog?.bark()
myDog!!.bark()
val theDog: Dog = !!myDog
if (myDog != null) {
   myDog.bark()
```

#### FUNCTIONS

- In Java, global functions do not exist.
  - Use class functions instead (static instance methods).
- In Kotlin, global functions do exist.

```
fun isPositive(x: Int): Boolean {
  return 0 <= x
}

fun isPositive(x: Int = 5): Boolean {
  return 0 <= x
}</pre>
```



#### FUNCTIONS

- In Java, global functions do not exist.
  - Use class functions instead (static instance methods).
- In Kotlin, global functions do exist.

```
fun isPositive(x: Int): Boolean {
  return 0 <= x
}</pre>
```

```
fun isPositive(x: Int) = 0 \le x
```

### FUNCTIONS

- In Java, global functions do not exist.
  - Use class functions instead (static instance methods).
- In Kotlin, global functions do exist.

```
fun printTwice(text: String): Unit {
  print(text); print(text)
}
```

• A function can access variables created in the scopes surrounding the function.

```
var counter = 0
fun incCounter(): Int {
  counter += 1
  return counter
}
print(incCounter()) // 1
print(incCounter()) // 2
```

• Functions are values that can be tossed around, e.g. returned.

```
fun createCounter(): () -> Int {
 var counter = 0
  fun incCounter(): Int {
    counter += 1
    return counter
                         val incCounterA = createCounter()
                         val incCounterB = createCounter()
 return ::incCounter
                         print(incCounterA()) // 1
                         print(incCounterB()) // 1
```

```
fun areBoth(a: Int, b: Int, test: (Int) -> Boolean): Boolean {
  return test(a) && test(b)
}
fun isOdd(x: Int) = x % 2 == 1
val areBothOdd = areBoth(3, 5, ::isOdd)
```



```
fun areBoth(a: Int, b: Int, test: (Int) -> Boolean): Boolean {
  return test(a) && test(b)
}
val areBothOdd = areBoth(3, 5, fun(x: Int) = x % 2 == 1)
```

```
fun areBoth(a: Int, b: Int, test: (Int) -> Boolean): Boolean {
  return test(a) && test(b)
}
val areBothOdd = areBoth(3, 5, { x -> x % 2 == 1})
```

```
fun areBoth(a: Int, b: Int, test: (Int) -> Boolean): Boolean {
  return test(a) && test(b)
}
val areBothOdd = areBoth(3, 5, { it % 2 == 1})
```

```
fun areBoth(a: Int, b: Int, test: (Int) -> Boolean): Boolean {
   return test(a) && test(b)
}
val areBothOdd = areBoth(3, 5) {
   it % 2 == 1
}
```

```
fun callTwice(theFunction: () -> Unit): Unit {
  theFunction()
  theFunction()
callTwice(){
 print("Hi!")
callTwice{
 print("Hi!")
```

## THE SPECIAL LET & RUN FUNCTION

```
val fileManager = FileManager()
fileManager.deleteFile("data.txt")
```

```
FileManager().let {
  it.deleteFile("data.txt")
}
```

```
FileManager().run {
  deleteFile("data.txt")
}
```

## THE SPECIAL LET & RUN FUNCTION

```
val fileManager = FileManager()
val wasDeleted = fileManager.deleteFile("data.txt")
val wasDeleted = FileManager().let {
  it.deleteFile("data.txt")
val wasDeleted = FileManager().run {
  deleteFile("data.txt")
```



### THE SPECIAL APPLY & ALSO FUNCTION

```
val fileManager = FileManager()
fileManager.deleteFile("data.txt")
val fileManager = FileManager().also {
  it.deleteFile("data.txt")
val fileManager = FileManager().apply {
  deleteFile("data.txt")
```



## DATA CLASSES

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

- Compiler generates:
  - equals()
  - hashCode()
  - toString()
  - copy()



## CLASSES

```
class Human(val name: String, val age: Int) {
  val city = "Jönköpoing"
  fun getPresentation() = "Hi! My name is $name."
}
```

#### INHERITANCE

```
open class Human(val name: String, val age: Int) {
  val city = "Jönköpoing"
  open fun getPresentation() = "Hi! My name is $name."
}
```

```
class Superman : Human("Superman", 26) {
  override fun getPresentation() = "I'm $name!"
}
```