

# Kotlin for Java Developers



what every java developer should know about kotlin

# Today I would like to share with you

- Why I care about Kotlin?
- 20 features I love
- 10 features you need to be aware of
- Ecosystem (platforms, tools, documentation, courses)

# Next JVM language?

- Statically typed programming language for multi platform applications
  - concise
  - safe
  - interoperable with Java
  - built together with tooling support
  - open source under Apache 2.0 license
- Version 1.0 released in 2015, current version 1.1.2-2 (April 2017)
- Google announced first-class support for Kotlin on Android
- Pivotal will introduce Kotlin support in Spring Framework 5.0
- Easy to learn if you know Java

# Kotlin basics

```
1  class Foo(val x : String, var y: Int) {  
2      fun sayHello() = print("Hello")  
3  }  
4  
5  fun main(args: Array<String>) : Unit {  
6      print("Hello world!")  
7  
8      val foo = Foo("JUG ZG", 14)  
9      foo.sayHello()  
10 }  
11
```

**Features I love**

# Extension functions and properties

```
1  import java.io.File
2  import java.math.BigDecimal
3
4  fun Int.toBigDecimal() = BigDecimal.valueOf(this.toLong())
5
6  val Int.bd : BigDecimal
7      get() = BigDecimal.valueOf(this.toLong())
8
9
10 fun String.reverse() = StringBuilder(this).reverse().toString()
11
12
13 fun main(args: Array<String>) {
14     print(1.bd + 2.toBigDecimal())
15
16     print("String to reverse".reverse())
17
18     val lines = File("test").bufferedReader().lines()
19     print(lines)
20 }
21
```

# Data classes

```
1 data class Account(val email: String)
```

- equals() / hashCode() pair
- toString() “Account[email=test]”
- componentN() functions in their order of declaration
- copy() function

```
1 import java.util.Objects;
2
3 public class AccountJava {
4
5     private String email;
6
7     public AccountJava(String email) {
8         this.email = email;
9     }
10
11    public String getEmail() {
12        return email;
13    }
14
15    public void setEmail(String email) {
16        this.email = email;
17    }
18
19    @Override
20    public boolean equals(Object o) {
21        if (this == o)
22            return true;
23        if (o == null || getClass() != o.getClass())
24            return false;
25        AccountJava that = (AccountJava) o;
26        return Objects.equals(email, that.email);
27    }
28
29    @Override
30    public int hashCode() {
31        return Objects.hash(email);
32    }
33
34    @Override
35    public String toString() {
36        return "AccountJava{" + "email='" + email + '\'' + '}';
37    }
38 }
39
```

# String templates

```
1 fun main(args: Array<String>) {  
2  
3     val name = "JUG Zielona Góra!"  
4     val message = "Hello: $name"  
5     print(message)  
6  
7     val s = "abc"  
8     print("String length is ${if (args.isNotEmpty()) args[0].length else s.length}")  
9  
10 }
```



# Null safety

- Types defines nullability
  - Platform types
- Safe calls
- Elvis operator
- !! operator
- Safe casts

```
1 fun main(args: Array<String>) {
2
3     // Non null type
4     var a: String = "abc"
5     //a = null // compilation error
6
7
8     // Nullable type
9     var b: String? = "abc"
10    b = null // ok
11
12
13    // Safe calls
14    val l = if (b != null) b.length else -1
15    b?.length
16
17
18    // Elvis operator
19    val account : Account? = Account("email", null)
20    print(account?.externalId ?: "Not a number")
21
22
23    // The !! operator
24    b!!.length
25
26
27    // Safe casts - return null if cast was not successful
28    val aInt: Int? = a as? Int
29 }
30
```

# when

```
1 fun getScore(score: Int) = when (score) {  
2     9, 10 -> "Excellent"  
3     in 6..8 -> "Good"  
4     4, 5 -> "Ok"  
5     in 1..3 -> "Fail"  
6     else -> "Fail"  
7 }  
8  
9 fun whenShowcase(x : Any?) {  
10     val validNumbers = listOf(21, 22, 23)  
11  
12     val result = when (x) {  
13         0, 1 -> "x == 0 or x == 1"  
14         is String -> x.startsWith("prefix")  
15         in 1..10 -> "x is in the range"  
16         in validNumbers -> "x is valid"  
17         !in 10..20 -> {  
18             "x is outside the range"  
19         }  
20         else -> "Else branch"  
21     }  
22     print(result)  
23 }
```

```
1 public class ScoreSwitch {  
2     String getScore(int score) {  
3         String grade;  
4         switch (score) {  
5             case 10:  
6             case 9:  
7                 grade = "Excellent";  
8                 break;  
9             case 8:  
10            case 7:  
11            case 6:  
12                grade = "Good";  
13                break;  
14            case 5:  
15            case 4:  
16                grade = "Ok";  
17                break;  
18            case 3:  
19            case 2:  
20            case 1:  
21                grade = "Fail";  
22                break;  
23            default:  
24                grade = "Fail";  
25            }  
26            return grade;  
27        }  
28    }  
29 }
```

# Operator overloading

- `+`, `-`, `*`, `/`, `%`, `..`
  - `a + b -> a.plus(b)`
  - `a..b -> a.rangeTo(b)`
- `in`, `!in`
  - `a.contains(b)`
- Indexed access `[]`
  - `a[i] -> a.get(i)`
  - `a[i] = b -> a.set(i, b)`
- Invoke
  - `a(i, j) -> a.invoke(i, j)`
- `a == b`
  - `a?.equals(b) ? : (b === null)`
- `a > b`, `a < b`, `a >= b`, `a <= b -> a.compareTo(b)`

# Operator overloading (1)

```
3 // + on BigDecimal
4 print(123.bd + 234.bd)
5
6 // []
7 val list = arrayOf(1, 2, 3)
8 list[0] = list[2]
9
10 // in, !in
11 println(4 !in list)
12 println('f' in 'a'..'z')
13
14 // invoke
15 val f = { a: Int, b: Int -> a + b }
16 println(f.invoke(2,3))
17 val f1 = f
18 println(f1(2,3))
19
20 // ==, ===
21 val account1 = Account("test", "foo")
22 val account2 = Account("test", "foo")
23 println(account1 == account2)
24 println(account1 === account2)
25
26 // comparison
27 println("1" > "2")
```

# Default and named parameters

```
1  @JvmOverloads
2  fun List<Any?>.join(separator: String = ",", prefix: String = "[", postfix: String = "]"): String {
3      val sb = StringBuilder()
4      sb.append(prefix)
5      for ((idx, e) in this.withIndex()) {
6          sb.append(e)
7          if (idx != this.lastIndex) {
8              sb.append(separator)
9          }
10     }
11     sb.append(postfix)
12     return sb.toString()
13 }
14
15 fun main(args: Array<String>) {
16     val list = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9)
17     println(list.join("|", ">", "<"))
18     println(list.join(prefix = ">", postfix = "<"))
19     println(list.join())
20 }
```

# Smart casts

```
1 fun smartFun(arg: Any) {
2     when (arg) {
3         is Int -> print(arg + 1)
4         is String -> print(arg.length + 1)
5         is IntArray -> print(arg.sum())
6     }
7 }
8
9 fun smartCasts(arg: Any?) {
10    if (arg !is String) return
11    println("$arg is String and its length is ${arg.length}")
12
13    if (arg is CharSequence && arg.length > 4) {
14        println(arg.length)
15    }
16 }
17
18 fun main(args: Array<String>) {
19
20    smartFun(1)
21    smartFun("JUG Zielona Góra")
22    smartFun(intArrayOf(1, 2, 3, 4, 5))
23
24    val x : Int? = null
25    val y = if (x != null) {
26        x + 4
27    } else {
28        0
29    }
30 }
```

# Destructuring objects

- underscore for unused variables (1.1)
- destructuring in lambdas (1.1)

```
1 fun main(args: Array<String>) {
2
3     // destructuring objects
4     val sample = Account("t@gest.com", "L-123")
5     val (email, externalId) = sample
6     println("$email $externalId")
7
8     // componentN convention
9     val email1 = sample.component1()
10    val externalId1 = sample.component2()
11
12    // destructuring in for
13    val list = listOf(sample, sample.copy("f@gest.com"))
14    for ((e, id) in list) {
15        println("$e $id")
16    }
17
18    // unused parameters
19    val (result, _) = compute()
20 }
21
22 fun compute() : Result {
23     return Result(0, true)
24 }
25 data class Result(val result: Int, val status : Boolean)
```

# Lambda and closures

```
1 fun up(arg: String): String = arg.toUpperCase()
2
3 fun main(args: Array<String>) {
4     val countries = listOf("Poland", "Germany", "France", "Italy")
5
6     // function getting String and returning String
7     val toUpperCase: (String) -> String = ::up
8
9     println(countries.map { element: String -> up(element) })
10
11     println(countries.map { it -> up(it) })
12
13     println(countries.map { toUpperCase.invoke(it) })
14
15     println(countries.map(toUpperCase))
16
17     println(countries.map { toUpperCase })
18
19     var count = 0
20     countries.forEach { count += it.length }
21     println(count)
22
23     countries.map { print("$it "); if (it == "Germany") return }
24
25     countries.map mapping@ { print(it); if (it == "Germany") return@mapping }
26
27     countries.map { print(it); if (it == "Germany") return@map }
28 }
```



# Expressions and statements

- if and when are expressions, not statements
  - `val length = if (a is String) a.length else -1`
  - `val action = when (test) {  
    in 0.5 -> OPEN  
    else -> CLOSE  
}`
- assignment is a statement, not an expression
  - `if (a = b)` does not compile
  - `while ((line = bufferedReader.readLine()) != null)` does not compile

# Packages and source code structure

- packages
- import allows to import classes, functions, \*
- type aliases
- multiple classes in one file
- arbitrary file names
- arbitrary directory structure
- visibility modifiers: private, protected, internal, public

# Other languages have all these features

- Null safety
- No checked exceptions
- Extension functions
- Function types and lambdas
- Default and named parameters
- Properties
- Operator overloading
- Smart casts
- Data classes
- Immutable collections
- Enhanced switch-case
- String templates
- Ranges
- Infix notation
- Inline functions
- Coroutines (async/await)
- Great standard library
- Sealed classes
- Delegated and lazy properties
- Class delegation
- Singletons
- Nested functions
- Object decomposition
- Top-level functions
- Reified generics
- Raw strings
- 100% interoperable with Java 6
- And more...

# Compile and run with Java code

- you can mix Java and Kotlin code in one project
- experiment with new language without breaking or rewriting the whole application
- small memory footprint of the Kotlin standard library

**Understand decisions**

# What you need to know

- final by default
- platform types and nullability
- no primitives, no implicit widening conversions for numbers
- bytecode
- function names - conventions
- standard library

# Final by default

- all classes, methods are final by default
  - tedious opening via 'open' keyword
  - interference with AOP (CGLIB), workarounds as compiler plugins 'kotlin-spring', 'all-open'
- 'override' is a required keyword, not an annotation
- designing for inheritance

# Platform types and nullability (!)

- any reference in Java may be null
- types of Java declarations are called platform types
- can be assigned to nullable or non-null type
- compiler, tools refers to them using as **T!** which means **T** or **T?**
- nullability annotations (JSR-305, Android, Lombok, JetBrains, Eclipse)

```
1 fun main(args: Array<String>) {  
2  
3     val list = ArrayList<String>() // non-null (constructor result)  
4     list.add("Item")  
5     val size = list.size // non-null (primitive int)  
6     val item = list[0] // platform type inferred (ordinary Java object)  
7  
8     item.substring(1) // allowed, may throw an exception if item == null  
9     val nullable: String? = item // allowed, always works  
10    val notNull: String = item // allowed, may fail at runtime  
11 }
```



# Bytecode generation

- kotlinc generates Java 6 or Java 8 bytecode
- on JVM lambdas does not use 'invokedynamic'
- <https://www.slideshare.net/intellyole/kotlin-bytecode-generation-and-runtime-performance>

# Function naming conventions

- `a()` -> invoke
- `[]` -> `a.set`, `a.get`
- `==`, `!=` -> equals
- `for (element in container)` -> uses `iterator()`
- `in` -> `a.contains(b)`, `!in`
- infix notation `1.shl(2)` -> `1 shl 2`

# Standard library

- `kotlin-runtime` and `kotlin-stdlib` -> `kotlin-stdlib`
  - < 1MB jar (JVM)
- Kotlin classes and extension functions to Java classes
  - `kotlin`
  - `kotlin.collections`
  - `kotlin.comparisons`
  - `kotlin.concurrent`
  - `kotlin.io`
  - `kotlin.streams`
  - `kotlin.text`
- `kotlin.jvm`
- `kotlin.js`

# Platforms and tooling

# Platforms and tooling

- JVM
  - a. Java 6 and 8
- Android
- JavaScript (ES5.1)
  - a. compatible with module systems like AMD, CommonJS
- native (LLVM)
  - a. LLVM is used to compile Kotlin into native code
  - b. technology preview for iOS, linux, MAC, (windows in the work)
- IntelliJ IDEA (Java to Kotlin converter), Eclipse
- Gradle, Maven, Ant

# Where to start?

- Try online <https://try.kotlinlang.org/>
- Kotlin is Awesome! <https://kotlin.link/>
- [This presentation](#)
- [And code examples](#)

What will be your next JVM  
language?

...

[https://en.wikipedia.org/wiki/List\\_of\\_JVM\\_languages](https://en.wikipedia.org/wiki/List_of_JVM_languages)