KOTLIN

FROM A JAVA DEVELOPER'S PERSPECTIVE

Kotlin

Dennis Lotz

WHAT IS KOTLIN?

- cross plattform
- statically typed with type inference
- concise
- compiles to JVM byte code

Kotlin

JAVA INTEROPERABILITY

" Kotlin is designed with Java Interoperability in mind. Existing Java code can be called from Kotlin in a natural way, and Kotlin code can be used from Java rather smoothly as well. "

kotlinlang.org

Kotlin

JAVA INTEROPERABILITY

```kotlin
import java.util.Calendar

fun calendarDemo() {
    val calendar = Calendar.getInstance()
    if (calendar.firstDayOfWeek == Calendar.SUNDAY) {
        calendar.firstDayOfWeek = Calendar.MONDAY
    }
}
```

Kotlin

Dennis Lotz

LESS VERBOSE

" Kotlin's modern language features allow you to focus on expressing your ideas and write less boilerplate code. Less code written also means less code to test and maintain. "

developer.android.com

Kotlin

LESS VERBOSE

Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println(“Hello World!”);
    }
}
```

Kotlin

```
fun main() {
    println(“Hello World”)
}
```

Kotlin

SEMICOLONS ARE OPTIONAL

```
println(“Hello World”);
```

is equivalent to

```
println(“Hello World”)
```

Kotlin

RETURN TYPES

`Unit` return type can be omitted

```
fun main(): Unit {
    println("Hello World")
}
```

is equivalent to

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

Kotlin

TYPE INFERENCE

```
val a: Int = 3
val b: Int
b = 5
```

is equivalent to

```
val a = 3 // type Int gets inferred
val b: Int
b = 5
```

Kotlin

VISIBILITY MODIFIERS

- `private`
- `protected`
- `internal`
- `public` (default)

```
private val a = 3
val b = 5 // public by default
```

Kotlin

ACCESSING PROPERTIES

Java

```
String name = somePerson.getName();
```

Kotlin

```
val name = somePerson.name
```

Kotlin

ASSIGNING PROPERTIES

Java

```
somePerson.setName("Dennis");
```

Kotlin

```
somePerson.name = "Dennis"
```

Kotlin

CREATING NEW INSTANCES

Java

```
Person somePerson = new Person("Dennis");
```

Kotlin (no `new` keyword)

```
val somePerson = Person("Dennis")
```

Kotlin

NAMED ARGUMENTS

```
class User(val username: String, val password: String)

val admin = User(username = "admin", password = "12345")
```

order doesn't matter

```
val user = User(password = "54321", username = "user")
```

Kotlin

DEFAULT ARGUMENTS

```
class User(
    val username: String,
    val password: String = "not-so-secure-password"
)

val admin = User(username = "admin")
```

Kotlin

SINGLE-EXPRESSION FUNCTIONS

```
fun theAnswer(): Int {
    return 42
}
```

is equivalent to

```
fun theAnswer() = 42
```

Kotlin

`var` AND `val`

```
var a = "mutable String"
val b = "immutable String"

a = "new value"
b = "new value" // compilation error
```

Kotlin

NULL SAFETY

" I call it my billion-dollar mistake. It was the invention of the null reference in 1965. "

Tony Hoare

Kotlin

NULL SAFETY

```
var a: String = "abc"
a = null // compilation error

var b: String? = "abc"
b = null // ok
```

Kotlin

SAFE CALL OPERATOR - ?.

```
val a: String = "abc"
a.length

val b: String? = "abc"

b.length // compilation error

b?.length
```

Kotlin

ELVIS OPERATOR - `?:`

If the expression to the left of `?:` is not null, the elvis operator returns it, otherwise it returns the expression to the right.

```
val b: String? = "abc"

val l = b?.length ?: -1
```

Kotlin

TEMPLATE STRINGS

```
val name = "Dennis"
println("My name is $name")
```

```
$ My name is Dennis
```

Kotlin

DATA CLASSES

- `equals()`
- `hashCode()`
- `toString()`
- `copy()`
- Destructuring Declarations

Kotlin

DATA CLASSES

```kotlin
data class User(
    val username: String,
    val password: String
)

val admin = User("admin", "12345")
```

Kotlin

DESTRUCTURING DECLARATIONS

```
data class User(
    val username: String,
    val password: String
)

val (username, password) = User("admin", "12345")
```

Kotlin

DESTRUCTURING DECLARATIONS

very handy in maps

```
for ((key, value) in map) {
    println("Key: $key, value: $value")
}
```

Kotlin

SMART CASTS

`is` check

```
if (x is String) {
    println(x.length) // x is automatically cast to String
}
```

negative `is` check

```
if (x !is String) return
println(x.length) // x is automatically cast to String
```

Kotlin

SMART CASTS

when-expressions

```
when (x) {
    is Int -> println(x + 1) // x is Int
    is String -> println(x.length + 1) // x is String
    is IntArray -> println(x.sum()) // x is IntArray
}
```

Kotlin

COLLECTIONS

Immutable

```
val x = listOf(a, b, c)
val y = setOf(a, b, c)
val z = mapOf(
    Pair("a", a),
    Pair("b", b),
    Pair("c", c)
)
```

Kotlin

COLLECTIONS

Mutable

```
val x = mutableListOf(a, b, c)
val y = mutableSetOf(a, b, c)
val z = mutableMapOf(
    Pair("a", a),
    Pair("b", b),
    Pair("c", c)
)
```

Kotlin

RANGES

ascending

```
for(i in 1..4) {
    println("Current Value is $i")
}
```

descending

```
for(i in 4 downTo 1) {
    println("Current Value is $i")
}
```

Kotlin

RANGES WITH STEP

ascending

```
for(i in 1..4 step 2) {
    println("Current Value is $i")
}
```

descending

```
for(i in 4 downTo 1 step 2) {
    println("Current Value is $i")
}
```

Kotlin

EXTENSION FUNCTIONS

" Provide the ability to extend a class with new functionality without having to inherit from the class or use any type of design pattern such as Decorator."

kotlinlang.org

Kotlin

UTIL CLASSES IN JAVA

```java
public class CollectionUtil {
    public static void swap(
        List<Integer> list,
        int index1,
        int index2
    ) {
        Integer tmp = list.get(index1);
        list.set(index1, list.get(index2));
        list.set(index2, tmp);
    }
}
```

```java
List<Integer> list = Arrays.asList(1,2,3);
CollectionUtil.swap(list, 0, 2);
```

Kotlin

Dennis Lotz

EXTENSION FUNCTIONS

Replacement of Java Util function

```
fun <T> MutableList<T>.swap(index1: Int, index2: Int) {
  val tmp = this[index1]
  this[index1] = this[index2]
  this[index2] = tmp
}
```

```
val list = mutableListOf(1, 2, 3)
list.swap(0, 2)
```

Kotlin

Dennis Lotz

MORE AWESOME FEATURES

- Coroutines
- Kotlin Native
- Scope functions
- Infix notation
- Ability to create DSLs
- Contracts
- ...

Kotlin

FURTHER INFORMATION

- https://kotlinlang.org/docs/reference/
- https://play.kotlinlang.org/koans/overview
- https://kotlinlang.org/docs/books.html
- Medium - Roman Elizarov
- YouTube - KotlinConf 2018

Kotlin

THANK YOU!

Questions?

Kotlin