Introduction to Kotlin

Victor Kropp

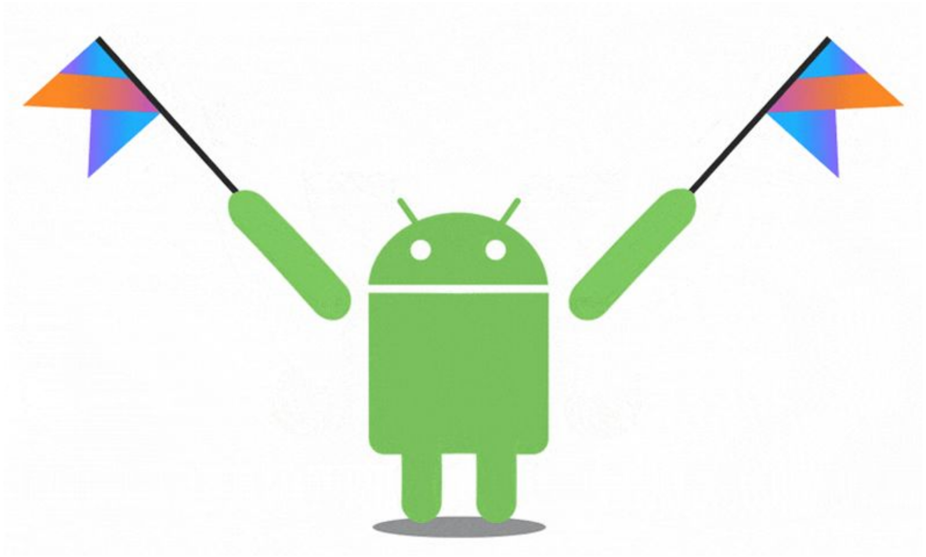
@kropp

Who am I?

At JetBrains since 2008

victor.kropp.name

Kotlin



Kotlin

Sample Java App

```
package kropp.name.myapp;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
   }
}
```

Sample Kotlin App

```kotlin
package kropp.name.myapp
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
   }
}
```

Kotlin

Make val not var

```
var   mutable: String
val immutable: String
```

Properties

```kotlin
class C {
  var prop: String = ""
}
```

Properties

```
public class C {
  private String prop;
  public String getProp() {
    return prop;
  }
  public void setProp(String prop) {
    this.prop = prop;
  }
  public C() { this.prop = ""; }
}
```

Kotlin

Properties

```
class C {
  val prop: String
    get() {
      return ""
    }
}
```

Properties

```
class C {
  val prop: String
    get() = ""
}
```

Properties

```kotlin
class C {
 private var myProp: String = ""
 val prop: String
   get() = myProp
}
```

Kotlin

Properties

```
public class C {
  private String myProp;
  public String getProp() {
    return myProp;
  }
  public C() {
    this.myProp = "";
  }
}
```

Properties

```
class C {
  lateinit var prop: String
}
```

Primary constructor

```
class Person(
  firstName: String,
  lastName: String,
  age: Int
) {
  …
}
```

Primary constructor

```
class Person(
  var firstName: String,
  var lastName: String,
  var age: Int
)
```

Data classes

```kotlin
data class Person(
  var firstName: String,
  var lastName: String,
  var age: Int
)
```

Java equivalent

```java
package kotlindemo;

import java.util.Objects;

public class tmp {
 private String firstName;
 private String lastName;
 private int age;

 public tmp(String firstName, String lastName, int age) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.age = age;
 }

 public String getFirstName() {
   return firstName;
 }

 public void setFirstName(String firstName) {
   this.firstName = firstName;
 }

 public String getLastName() {
   return lastName;
 }

 public void setLastName(String lastName) {
   this.lastName = lastName;
 }

 public int getAge() {
   return age;
 }

 public void setAge(int age) {
   this.age = age;
 }

 @Override
 public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;
   tmp tmp = (tmp) o;
   return age == tmp.age &&
       Objects.equals(firstName, tmp.firstName) &&
       Objects.equals(lastName, tmp.lastName);
 }

 @Override
 public int hashCode() {
   return Objects.hash(firstName, lastName, age);
 }
}
```

String templates

```
val list = listOf<String>()

val count = list.size
val template = "$count items"

val template = "${list.size} items"
```

Kotlin

Kotlin is fun

```
fun f() {
}
```

Kotlin

Kotlin is fun

```
fun max(a: Int, b: Int): Int {
  if (a > b) {
    return a
  } else {
    return b
  }
}
```

Kotlin is fun

```kotlin
fun max(a: Int, b: Int): Int {
  val max = if (a > b) {
    a
  } else {
    b
  }
  return max
}
```

Kotlin is fun

```kotlin
fun max(a: Int, b: Int): Int {
  return if (a > b) {
    a
  } else {
    b
  }
}
```

Kotlin is fun

```
fun max(a: Int, b: Int): Int = if (a > b) a else b
```

Kotlin is fun

```
fun max(a: Int, b: Int): Int = if (a > b) a else b
```

Kotlin is fun

```kotlin
fun max(a: Int, b: Int) = if (a > b) a else b
```

Generics

```kotlin
fun <T> singletonList(item: T): List<T> = ...
```

Kotlin

Default arguments

```kotlin
fun reformat(str: String,
             normalizeCase: Boolean = true,
             upperCaseFirstLetter: Boolean = true,
             divideByCamelHumps: Boolean = false,
             wordSeparator: Char = ' ') {
}
```

Default arguments

```
reformat(str)
```

Default arguments are used

```
reformat(str, true, true, false, '_')
```

Kotlin

Named arguments

```
reformat(str,
    normalizeCase = true,
    upperCaseFirstLetter = true,
    divideByCamelHumps = false,
    wordSeparator = '_'
)
```

Kotlin

varargs

```
fun foo(vararg strings: String) {}

foo("a")
foo("a", "b")
foo(*arrayOf("a", "b", "c"))
```

Extension functions

```kotlin
fun Int.days(): Period = ...

fun Period.ago(): Date = ...



3.days().ago()
2.months().later()
```

Extension properties

```
val Int.days: Period
  get() = ...
val Period.ago: Date
  get() = ...

3.days.ago
2.months.later
```

Kotlin

Nullable receiver

```
fun Any?.toString(): String {
  if (this == null) return "null"
  return toString()
}
```

after the null check, `this` is autocast to a non-null type, so the `toString()` call resolves to the member function of the Any class

Operator overloading

```
public inline operator fun BigInteger.plus(other: BigInteger) :
                                          BigInteger = this.add(other)
```

```
val i1 = BigInteger.valueOf(1)
val i2 = BigInteger.valueOf(2)
val sum = i1 + i2
```

Kotlin

Operator overloading

```
+a   a.unaryPlus()       a + b a.plus(b)          a += b a.plusAssign(b)
-a   a.unaryMinus()      a - b a.minus(b)         a -= b a.minusAssign(b)
!a   a.not()             a * b a.times(b)         a *= b a.timesAssign(b)
a++  a.inc()             a / b a.div(b)           a /= b a.divAssign(b)
a--  a.dec()             a % b a.rem(b)           a %= b a.remAssign(b)
                         a..b  a.rangeTo(b)
```

```
a > b  a.compareTo(b) > 0
a < b  a.compareTo(b) < 0
a >= b a.compareTo(b) >= 0
a <= b a.compareTo(b) <= 0
```

Equality

```
a == b // a.equals(b)

a === b
```

get()/set() convention

```
val map = mutableMapOf<String,Any>()

map["key"] = "value"
val value = map["key"]
```

invoke() convention

```
dependencies.compile("org.jetbrains.kotlinx:kotlinx-html-jvm:0.6.4")

dependencies {
  compile("org.jetbrains.kotlinx:kotlinx-html-jvm:0.6.4")
}
```

invoke() convention

```
fun DependencyObj.invoke(builder: DependencyObj.() -> Unit)
        = this.apply(builder)
```

Kotlin

Infix notation

```
public infix fun <A, B> A.to(that: B): Pair<A, B>
                                  = Pair(this, that)


"key".to("value")
"key" to "value"
```

Kotlin

Infix notation

```
for (i in 0 until 10 step 2) {
   // 0, 2, 4, 6, 8
}
```

Lambda expressions

```
val sum = { x: Int, y: Int -> x + y }

val sum: (Int, Int) -> Int = { x, y -> x + y }
```

Kotlin

Lambda expressions

```
val sum = { x: Int, y: Int -> x + y }

val sum: (Int, Int) -> Int = { x, y -> x + y }

val sum : Int.(Int) -> Int = { n -> this + n }
```

Lambda expressions

```
val sum = { x: Int, y: Int -> x + y }

val sum: (Int, Int) -> Int = { x, y -> x + y }

val sum : Int.(Int) -> Int = { n -> this + n }

val sum : Int.(Int) -> Int = { this + it }
```

Lambda expressions

```
val list = listOf<Int>()

list.filter({ it > 0 })
```

Lambda expressions

```kotlin
val list = listOf<Int>()

list.filter { it > 0 }
```

Lambda expressions

```
val list = listOf<Int>()

list.filter { it > 0 }.map { it*2 }
```

inline functions

```
inline fun <T> Iterable<T>.filter(predicate: (T) -> Boolean):
                                                    List<T> {
  val result = mutableListOf<T>()
  for (it in this) {
    if (predicate(it)) {
      result.add(it)
    }
  }
  return result
}
```

Null safety

```
val canBeNull: String?
val notNull: String
```

Null safety

```
fun nullability(str: String?) {
  val dot = str.indexOf(".")
}
```

Null safety

```
fun nullability(str: String?) {
  val dot = str.indexOf(".")
}
```

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?

Null safety

```
fun nullability(str: String?) {
    val dot = str!!.indexOf(".")
}
```

Non-null asserted call

May throw `NullPointerException`

Usually a bad style,
use only when you know what you are doing

Kotlin

Null safety

```
fun nullability(str: String?) {
    val dot = str?.indexOf(".")
}
```

Safe call

The result will be `null` if `str` is `null`

Kotlin

Null safety

```
fun nullability(str: String?) {
    val dot = str?.indexOf(".") ?: 0
}
```

Elvis operator

The result will be `0` if `str?.indexOf()` returns `null`

Type casts

```
fun cast(obj: Any) {
    if (obj is String) {
        val dot = obj.indexOf(".")
    }
}
```

Smart cast

obj is String inside ‘then’ branch

Kotlin

Type casts

```kotlin
fun cast(obj: Any) {
    val str = obj as String
    val dot = str.indexOf(".")
}
```

Type casts

```
fun cast(obj: Any) {
   val str = obj as? String
```

Safe cast

`str` is `null` if `obj` is not a `String`

```
   val dot = str.indexOf(".")
}
```

Kotlin

when expression

```kotlin
class Expr
class Const(val number: Double) : Expr()
class Sum(val e1: Expr, val e2: Expr) : Expr()
object NotANumber : Expr()

fun eval(expr: Expr): Double = when(expr) {
  is Const -> expr.number
  is Sum -> eval(expr.e1) + eval(expr.e2)
  NotANumber -> Double.NaN
  else -> 0
}
```

Sealed types

```kotlin
sealed class Expr
class Const(val number: Double) : Expr()
class Sum(val e1: Expr, val e2: Expr) : Expr()
object NotANumber : Expr()

fun eval(expr: Expr): Double = when(expr) {
  is Const -> expr.number
  is Sum -> eval(expr.e1) + eval(expr.e2)
  NotANumber -> Double.NaN
}
```

Delegates

```kotlin
val lazyValue: String by lazy {
    // some long computation
    "Hello World!"
}
```

Kotlin

Delegates

```
class User(val map: Map<String, Any?>) {
    val name: String by map
    val age: Int     by map
}
```

Delegates

```
interface ReadOnlyProperty<in R, out T> {
  operator fun getValue(thisRef: R, property: KProperty<*>): T
}

interface ReadWriteProperty<in R, T> {
  operator fun getValue(thisRef: R, property: KProperty<*>): T
  operator fun setValue(thisRef: R, property: KProperty<*>, value: T)
}
```

Coroutines (Kotlin 1.1)

Asynchronous programming made easy

Will cover in details in a separate talk later today

Kotlin

Kotlin for Android

Kotlin in Android Studio

Supported out of the box since 3.0

Code samples are available in Kotlin too

Kotlin Android Extensions

Anko

Kotlin

Anko

```
verticalLayout {
  val name = editText()
  button("Say Hello") {
    onClick { toast("Hello, ${name.text}!") }
  }
}
```

Links

Kotlin
https://kotlinlang.org

Kotlin Koans

https://try.kotl.in

Kotlin

Kotlin Community

slack

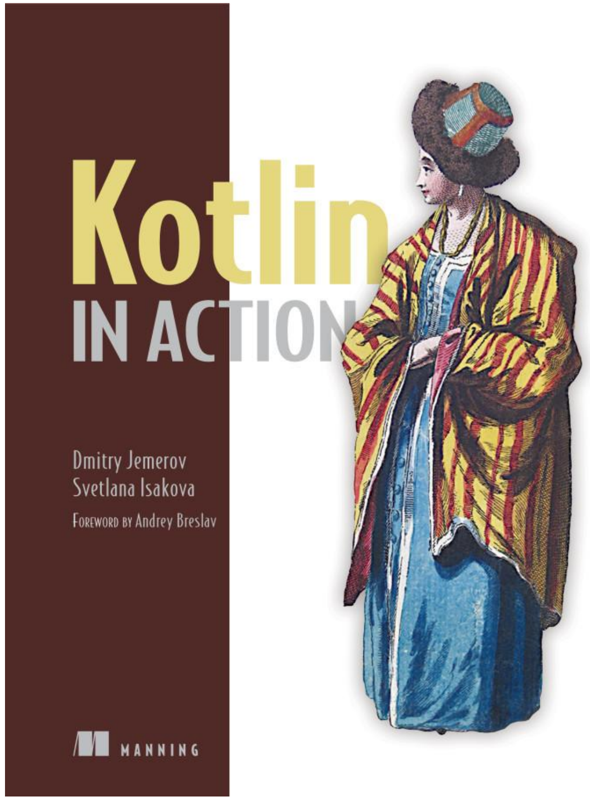
https://kotlinlang.slack.com/

Get invite at

http://slack.kotlinlang.org/

Kotlin

Kotlin in Action



Kotlin
IN ACTION

Dmitry Jemerov
Svetlana Isakova
Foreword by Andrey Breslav

MANNING

Kotlin

Thank you!

Victor Kropp

@kropp

victor.kropp.name

Questions?

Victor Kropp

@kropp

victor.kropp.name