E V

O L

V E

An introduction to Kotlin

What is Kotlin?

- A statically-typed programming language that runs on the JVM
- Developed by JetBrains
- Unveiled 2011, development started about one year earlier
- First-class support for Android since Google I/O 2017
- Full support from Android Studio 3

Why Kotlin?

- Concise - Reduce the amount of boilerplate code
- Safe - Avoid null pointer errors, use immutable types to improve concurrency
- Interoperable - Consume existing Java code/libraries, write code that can be used from Java
- Allows gradual migrations from Java to Kotlin
- Build for Java, Android, JavaScript and more...
- Powerful IDE:s

Syntax & Semantics

- Pascal-type declaration style for variable and parameter lists:

```
val myValue: Int = 10
```

- Semicolon termination optional, should not be used
- Object-oriented style and procedural programming with functional support
- Type inference support:

```
val myInt = 10
val myString = “Test”
val myClass = MyClass()
```

- Distinction between nullable and non-nullable types:

```
var nullable: Int?
var nonNullable: Int
```

Syntax & Semantics

- Interaction with nullable objects needs special handling:
  ?. - safe navigation operator
  ?: - null coalescing operator
  !! - assert if expression is null
- Compile-time error checks for null violations
- Unique default constructors:
  ```
  class Test(private val firstName: String, private val secondName: String)
  ```
- Everything are functions, there's no void:
  ```
  fun doStuff() { … } // (Unit)
  fun add(x: Int, y: Int, z: Int = 0): Int = x+y+c
  ```

Syntax & Semantics

- Properties are simple
- Extension functions (similar to c#), Anko:

```
fun Context.toast(message: CharSequence, duration: Int = Toast.LENGTH_SHORT)
{Toast.makeText(this, message, duration).show()}
```

- doAsync(), uiThread - will not execute if activity.isFinishing() is true
- Data classes, simplifies POJO classes:

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

- Declaration deconstructing:

```
val o = myModel("Mikael", 40)
val (name, age) = o
```

- Object, Companion objects - easy singleton, "static" properties
- Operator overloading - Fixed number of symbolic operators to overload

Syntax & Semantics

- Lambdas - First class citizens

```
class GamesAdapter(private val games: List<Game>, private val itemClick: (Game) -> Unit)
…
val adapter = GamesAdapter(viewModel.getGamesForView(), {game -> openDetail(game) })
val adapter = GamesAdapter(viewModel.getGamesForView(), {openDetail(it)})
```

- Inline functions - Reduce memory footprint, increase performance
- Lazy instantiation:

```
private val allGames: List<Game> by lazy {
     repository.getGames()
}
```

- Lateinit:

```
private lateinit var viewModel: LoginViewModel
```

Syntax & Semantics

- Collections - LINQ style: Aggregate, Filtering, Mapping, Elements, Generations, Ordering
- Flow control specials: When

```
 when(requestCode) {
      RequestSortCode -> {
            updateView(data?.getStringExtra(SortingActivity.SortingId).toString())
      }
      else -> super.onActivityResult(requestCode, resultCode, data)
}
```

- Generics - similar to c#
- Coroutines - async/await similarity - write async code in sequence, and more

Advantages

- Fully native code in a modern way
- Reduce coding, increase readability
- Stable code (almost) by default. Use of immutable, null checks
- Synthetic imports (avoid findViewById)
- Use existing components (Picasso, Mockito, jUnit...)
- Extremely powerful IDEs ranging from syntax to integration (Google play etc)
- Official language for Android

Disadvantages

- No namespaces
- Java to Kotlin converter not 100% accurate
- Method count will increase in compiled code (backing fields, inline)

Links

- Java version
  https://github.com/MikaelStalvik/javagameapp
- Kotlin version
  https://github.com/MikaelStalvik/kotlingameapp