MOBILE APPLICATION DEVELOPMENT

ANDROID (2017)

LECTURE 02: KOTLIN

WHAT IS KOTLIN?

- The newest official language for Android development, joining Java and C++.
- In development since 2010, officially released in February 2016.
- 100% compatible with Java, and runs on the JVM (as well as other places).
- Attempts to be safer, more expressive, and more concise than Java:
  - Approximately 40% fewer lines of code for similar functionality.
  - Provides non-nullable types and safe, easy null checking.
  - Supports higher-order functions like `map`, `filter`, `reduce`, and `fold`.

KOTLIN FEATURES: BASIC TYPES

- Kotlin supports Byte, Short, Int, Long, Float, and Double as numeric types:
  - These have bit widths of 8, 16, 32, 64, 32, and 64, respectively.
- Char and Boolean are non-numeric types representing characters and booleans.
- The String class provides immutable collections of Chars which represent text.
  - String templates let expressions be evaluated in String literals: "${ 1 + 1 }" == "2"
- The Array class represents a sequential collection of typed items:
  - More efficient representations of Array exist for numeric types, such as IntArray.

KOTLIN FEATURES: SPECIFYING TYPES, MUTABILITY, AND NULLABILITY

- Types in Kotlin can be inferred (`var x = 1`) or explicitly specified (`var x: Int = 1`).
- Kotlin differentiates between mutable and immutable values:
  - `var x = 1` declares a variable initially equal to 1 whose value can change.
  - `val x = 1` declares an immutable value which is always equal to 1.
- Kotlin also differentiates between nullable and non-nullable values:
  - `var x: Int?` declares a variable of type Int which can be null.
  - `var x: Int` declares a variable of type Int which can NEVER be null.

KOTLIN FEATURES: SAFE NULL CHECKING

- Nullable references can be safely checked for null in various ways:
  - Assume we have a nullable String, `a` (`var a: String?`).
  - Writing `val b: Int = a.length` is unsafe! If `a` were null, this could crash.
  - We can check for null like in Java: `val length: Int = if (a != null) a.length else -1`
  - Or we can use a more Kotlin-esque approach: `val b: Int = a?.length ?: -1`
- The `?` operator before calls on nullable items will return that call's result if the item is not null, or will return null if the item is null. If `a` is null, `a?.length` returns null.

KOTLIN FEATURES: RANGES

- Kotlin offers many ways to iterate over ranges:
  - `for (i in 1..4) print(i)` prints "1234".
  - `for (i in 4 downTo 1) print(i)` prints "4321".
  - `for (i in 1..4 step 2) print(i)` prints "13".
  - `for (i in 1 until 4) print(i)` prints "123".
- Ranges can also be used in if expressions and other places:
  - `if (i in 1..10) print(i)` would print the variable `i` if `i` had a value in the range 1-10.

KOTLIN FEATURES: IF EXPRESSIONS

- The keyword `if` is an expression in Kotlin, meaning it returns a value. This value can be Unit (Kotlin's rough equivalent to void, meaning no value is returned), or it can be a value the programmer can assign or use elsewhere:
  - Using `if` as an expression: `val max = if (a > b) a else b`
  - Using `if` as a 'statement' (expression whose result is Unit):

```
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}
```

KOTLIN FEATURES: WHEN EXPRESSIONS

- The keyword `when` replaces the traditional `switch` statement and is an expression in Kotlin, meaning that like the `if` expression it returns a value (or Unit). Cases in a when expression can take on a variety of forms, and be grouped together with commas:

```
when (x) {
    0, 1 -> print("x == 0 or 1.")
    in 2..10 -> print("x is in the range 2-10.")
    x.isOdd() -> print("x is odd")
    is String -> print("x is a String.")
    else -> print("x didn't meet any criteria.")
}
```

```
val y: String = when (x) {
    0, 1 -> "x == 0 or 1."
    in 2..10 -> "x is in the range 2-10."
    x.isOdd() -> "x is odd"
    is String -> "x is a String."
    else -> "x didn't meet any criteria."
}
```

KOTLIN FEATURES: FUNCTIONS

- In Kotlin, functions are written as follows:

```
fun NAME(PARAMETER: TYPE): RETURN_TYPE {
    // Code...
}
```

- Functions can return values directly if the entire function body is an expression:

```
fun double(x: Int): Int = x * 2

fun oddString(x: Int): String = "${ x.isOdd() }"

fun overFive(x: Int): Bool = x > 5
```

```
fun describe(x): String = when (x) {
    0, 1 -> "x == 0 or 1."
    in 2..10 -> "x is in the range 2-10."
    x.isOdd() -> "x is odd"
    is String -> "x is a String."
    else -> "x didn't meet any criteria."
}
```

KOTLIN FEATURES: SMART CASTING

- Kotlin uses the `is` keyword to determine if a value is a certain type. As soon as this check returns true, a checked value may be used from that point on as a value of the checked type, until the enclosing scope is exited.

- An `is` check will be remembered within a given scope even on the other side of an `||` or `&&` operator or after an `if` expression in which it appeared. As long as the value can be guaranteed not to change by the compiler, the `is` check will be remembered for a given value.

```
if (x is String) {
    print(x.length) // Automatically casts x.
}

if (x is String && x.length > 0) {
    print(x.length) // Again, automatically casts.
}

fun printIfString(x: Any) {
    if (x !is String) return
    print(x.length) // Still remembers the check!
}
```

KOTLIN FEATURES: HIGHER-ORDER FUNCTIONS

- Kotlin supports classic higher-order functions such as `map`, `filter`, `reduce`, and `fold`. Given an Array of Ints [1, 2, 3], these functions would do the following:
  - Map: `array.map { it + 1 }` // Returns the List<Int> [2, 3, 4].
  - Filter: `array.filter { it > 1 }` // Returns the List<Int> [2, 3].
  - Reduce: `array.reduce { sum, element -> sum + element }` // Returns the Int 6.
  - Fold: `array.fold(1) { sum, element -> sum + element }` // Returns the Int 7.