Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)
Dr.

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

JS 2 K - Part 2

JavaScript to Kotlin

Collections

JavaScript

```
const houses = [ "Stark", "Lannister", "Tyrell", "Arryn", "Targaryen", "Baratheon" ];
houses[2]; // "Tyrell"
houses.push("Martell");
houses.length; //7
```

Kotlin

```
val houses = mutableListOf("Stark", "Lannister", "Tyrell", "Arryn", "Targaryen", "Mart
houses[2] // "Tyrell"
houses.add("Martell")
houses.size //7
```

Collections


```
// mapOf is the read-only version of mutableMapof
val colors =  mapOf(
  "red" to 0xff0000,
  "green" to 0x00ff00,
  "blue" to 0x0000ff
)
val updatedColors = colors.plus("teal" to 0x008080) // doesn't change the original - it ret


// listOf is the read-only version of mutableListof
val houses = listOf("Stark", "Lannister", "Tyrell", "Arryn", "Targaryen", "Martell", "Barat

// Methods that return a new list instead of modifying it are still available:
var updatedHouses = houses.take(3).map {it.toUpperCase()} //["STARK", "LANNISTER", "TYRELL"

// Adding new items requires copying the whole original one and making sure the new copy is
var updatedHouses = houses.toMutableList().apply{ add("Martell") }.toList()
```

Objects (key-value maps)

JavaScript

```
const colors = {
  "red":  0xff0000,
  "green":  0x00ff00,
  "blue":  0x0000ff,
  "cyan":  0x00ffff,
  "magenta":  0xff00ff,
  "yellow":  0xffff00
};
colors.hasOwnProperty("yellow"); // true
colors.yellow; // 0xffff00
```

Kotlin

```
val colors =  mutableMapOf(
  "red" to 0xff0000,
  "green" to 0x00ff00,
  "blue" to 0x0000ff,
  "cyan" to 0x00ffff,
  "magenta" to 0xff00ff,
  "yellow" to 0xffff00
)
colors.contains("yellow") // true
colors.get("yellow") // 0xffff00
```

Destructuring Assignment

Kotlin by JetBrains

JavaScript

```
const coordinates = [5, 10, 15];
const [x, y, z] = coordinates;
```

Kotlin

```
val coordinates = arrayOf(5, 10, 15)
val (x, y, z) = coordinates
```

Destructuring Assignment

JavaScript

```
function weatherReport(location) {
  // Make an Ajax request to fetch the weather...
  return [72, "Mostly Sunny"];
}
const [temp, forecast] = weatherReport("Berkeley, CA");
```

Kotlin

```
fun weatherReport(location) {
  // Make an Ajax request to fetch the weather...
  return Pair(72, "Mostly Sunny") // Pair is a standard class in Kotlin that repres
}
val (temp, forecast) = weatherReport("Berkeley, CA")
```

JavaScript

Classes

Kotlin by JetBrains

```
class Monster {
  constructor(name, color, numEyes) {
    this.name = name;
    this.color = color;
    this.numEyes = numEyes;
  }
  speak(likes) {
      return `My name is ${this.name} and I like ${likes}`;
  }
}
var nhama = new Monster("Nhama", "red", 1);
nhama.speak("guacamole")
// "My name is Nhama and I like guacamole"
```

Classes


```
class Monster(val name: String, val color: String, val numEyes: Int) {
  fun speak(likes: String):String {
      return "My name is $name and I like $likes"
  }
}
var nhama = Monster("Nhama", "red", 1)
// Kotlin doesn't have a `new` keyword - you instantiate a class by calling it directly
nhama.speak("guacamole")
// "My name is Nhama and I like guacamole"
```

Data Containers

JavaScript

```
const movie1 = {
    name: "Back to the Future",
    rating: 5,
    director: "Bob Zemeckis"
}
const movie2 = {
    name: "Star Wars: Episode IV - A New Hope",
    rating: 5,
    director: "George Lucas"
}
```

Kotlin

```
data class Movie(
  val name: String,
  val rating: Int,
  val director: String
)
val movie1 = Movie("Back to the Future", 5, "Bob Zemeckis")
val movie2 = Movie("Star Wars: Episode IV - A New Hope", 5, "George Lucas")
```

Nullability


```
var greeting: String = "Hello, World"
greeting = null // Compilation Error
```

By default, Kotlin assumes that greeting cannot be null:
To allow null values, you have to declare a variable as nullable by appending a question mark in its type declaration:

```
var nullableGreeting: String? = "Hello, World"
nullableGreeting = null // Works
```

Kotlin

For example, The following method access works because Kotlin knows that the variable greeting can never be null:

```
val len = greeting.length
```

But the same method call won't work with nullableGreeting variable -

```
val len = nullableGreeting.length // Compilation Error
```

Safe Call Operator


```
val a = "Kotlin"
val b: String? = null
println(a?.length) // 6
println(b?.length) // null
```

That's great but that's not all. You can chain multiple safe calls like this:

```
val currentCity: String? = user?.address?.city
```

Such a chain returns null if any of the properties in it is null.

Elvis Operator

Kotlin

If you want to provide a default value if some variable is null, you can use the Elvis operator `?:`

```
val name = nullableUserName ?: "Guest"
```

You can use the safe call operators (or any other expressions) on the left side of Elvis operator:

```
val name = nullableUser?.name ?: "Guest"
```

Asynchronous Programming

JavaScript

```
async function getStatus() {
    const currentUserPromise = someApi.fetchUser();
    const currentCompanyPromise = someApi.fetchCompany();
    return await Promise.all([currentUserPromise, currentCompanyPromise]);
}
```

Kotlin

```
suspend fun getStatus(): List<String> {
    val currentUserDeferred = someApi.fetchUser()
    val currentCompanyDeferred = someApi.fetchCompany()
    return listOf(currentUserDeferred.await(), currentCompanyDeferred.await())
}
```

References

Sources: https://dev.to/cassiozen/kotlin-for-js-devs-part-1-5bld
https://dev.to/cassiozen/kotlin-for-js-devs-part-2-fam


Thanks.