Kotlin Part 2

Upon completion of this module, a student will be able to

- work with Lists in Kotlin
- use Loops in Kotlin
- work with Interfaces, Abstract Classes and Inherited Classes in Kotlin
- use special types of classes in Kotlin
- work with Pairs and Maps in Kotlin
- use Kotlin Extension Functions

https://www.udacity.com/course/kotlin-bootcamp-for-programmers--ud9011

Assignment

- Task
  - Build classes for the Animal Kingdom using Inheritance, Composition, Lists, and Loops
- Repo
  - https://github.com/LambdaSchool/Android_KotlinAnimalKingdom
- Submission
  - Fork on github and submit pull request

A Student Can

Lists

Lists

- List
  - Can’t change
  - Can be a combination of types
- MutableList
  - Changeable
- Array
  - Can’t change size, but can replace values
- Automatic Initializer

```
val numbers = listOf(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
println(numbers[1])
println(numbers.indexOf(3))
println(numbers.subList(1, 6))

val mutableStrings = mutableListOf<String>()
mutableStrings.add("One")
mutableStrings[0] = "Two"

val intArray = intArrayOf(2, 3, 4)
intArray[2]

val initNumbers = Array(6){ it * 3 }
println(initNumbers)
```

https://kotlinlang.org/docs/reference/collections.html

List Methods

- forEach
- filter
- sort - automatic
- sortBy - provide value to compare
- sortWith - provide comparator

```
val rabbits = mutableListOf<Rabbit>()
rabbits.addAll(rabbitRepo.findAll())

rabbits.forEach {
    ...
    val message = RabbitMessage(it.toString(), priority, secret)
    ...
}

rabbits.filter { it.name.charAt(0) == 'c' && it.name.charAt(it.length - 1) == 'e' }

val ints = mutableListOf<Int>()
ints.sort()
ints.sortDescending()

rabbits.sortBy { it.name }

rabbits.sortWith(Comparator { r1, r2 ->
    when {
        r1.weight > r2.weight -> 1
        r1.weight == r2.weight -> 0
        else -> -1
    }
})
```

https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html

More List Methods

- Reverse vs Reversed
- Contains
- SubList
- Sum

```
mylist.reverse() // reverses in place
mylist.reversed() // returns a reversed list
mylist.contains(11)
mylist.subList(0, 5)
mylist.sum()
```

Challenge

- Create a list of integer values and populate it with random numbers using `Random.nextInt(max)`
- Print all the elements in the list using a forEach loop
- Sort the elements and print them again
- Search for an integer value using `.contains(value)`

3 Minutes

Solution

```
fun lists() {
    val randInts = Array(10) {
        Random.nextInt(9)
    }

    val arrayString = StringBuilder()
    randInts.forEach { arrayString.append("$it ") }
    println("Original: $arrayString\n")

    randInts.sort()

    arrayString.clear()
    randInts.forEach { arrayString.append("$it ") }
    println("Sorted: $arrayString\n")

    println(randInts.contains(5))
}
```

A Student Can

Kotlin loops

While Loop

- While
- Do While
  - Expression is inside of loop scope

```
while (x > 0) {
    x--
}

do {
    val y = retrieveData()
} while (y != 0) // y is visible here
```

https://kotlinlang.org/docs/reference/control-flow.html

For Loops

- Range

```
val numbers = listOf(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
numbers.forEach { println(it) }

numbers.forEachIndexed{index, it -> {println("I: $index Val: $it")}

println(numbers[1])
println(numbers.indexOf(3))
println(numbers.subList(1, 6))

val mutableStrings = mutableListOf<String>()
mutableStrings.add("One")
mutableStrings[0] = "Two"

numbers.forEach { println(it) }
mutableStrings.forEachIndexed { i, string -> println("$i : $string") }
```

https://kotlinlang.org/docs/reference/control-flow.html

A Student Can

Inheritance and Polymorphism

Inheritance

- “Any” Class
- Constructor in Class Signature
- Open - Can be extended
- Sealed - Only extended in this file

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}

class LanguageNotFoundException(id: Long) :
        RuntimeException("Could not find Language $id")
```

https://kotlinlang.org/docs/reference/classes.html

Abstract Classes and Interfaces

- Abstract Classes
  - Constructors
  - Default Implementations
- Interfaces
  - No Constructors
  - Default Implementations

```
abstract class Person {

    var age: Int = 40

    fun displaySSN(ssn: Int) {
        println("My SSN is $ssn.")
    }

    abstract fun displayJob(description: String)
}

class Woman: Person() {
    override fun displayJob(description: String) {
        println("I have a great job.")
    }
}

interface Grinder {
    fun load() {
        println("Loaded")
    }
    fun grind()
}

interface SpiceColor {
    val color: Color
}
```

https://kotlinlang.org/docs/reference/classes.html

Inheritance Example

- Sealed
- Abstract
- Interface
- Enum
- Object

```
sealed class Spice(
    val name: String,
    private val spiciness: String = "mild",
    color: SpiceColor = YellowSpiceColor) :
    SpiceColor by color {
    val heat: Int
        get() {
            return when (spiciness) {
                "mild" -> 1
                "medium" -> 2
                "spicy" -> 3
                "very spicy" -> 4
                "extremely spicy" -> 5
                else -> 0
            }
        }

    abstract fun prepareSpice()
}

class Curry(name: String = "Curry", spiciness: String = "mild") :
    Spice(name, spiciness),
    Grinder {
    override fun prepareSpice() {
        grind()
    }
}

interface Grinder {
    fun grind() {
        println("Grind")
    }
}

enum class Color(val rgb: Int) {
    RED(0xFF0000), GREEN(0x00FF00), BLUE(0x0000FF), YELLOW(0xFFFF00)
}

interface SpiceColor {
    val color: Color
}

object YellowSpiceColor : SpiceColor {
    override val color: Color
        get() = Color.YELLOW
}
```

https://kotlinlang.org/docs/reference/classes.html

A Student Can

Special Classes

Special Classes

- Singletons (Static)
- Enum

```
object SingletonObject{
    var staticParameter: Int = 0
}

enum class Color(val rgb: Int) {
    RED(0xFF0000), GREEN(0x00FF00), BLUE(0x0000FF), YELLOW(0xFFFF00)
}

data class SpiceContainer(val spice: Spice) {
    val label = spice.name
}

fun accessSpecialClasses() {
    SingletonObject.staticParameter
    Color.YELLOW
}
```

https://kotlinlang.org/docs/reference/enum-classes.html - https://kotlinlang.org/docs/tutorials/kotlin-for-py/objects-and-companion-objects.html

Data Class

- Data Class
  - equals()
  - copy()
- Decomposition

```
fun dataObjectAccess() {
    val lukeObject = SwApiPerson(
        "Luke Skywalker", 172, 77,
        "blond", "fair", "blue", "19BBY", "male"
    )

    val (lukeName, lukeHeight, lukeMass,
        lukeHairColor, lukeSkinColor, lukeEyeColor,
        lukeBirthYear, lukeGender) = lukeObject

    println(lukeName)
    println(lukeHeight)
    println(lukeMass)
    println(lukeHairColor)
    println(lukeSkinColor)
    println(lukeEyeColor)
    println(lukeBirthYear)
    println(lukeGender)
}
```

https://kotlinlang.org/docs/reference/data-classes.html

Challenge

- Pokemon Class
  - Write an Enum called Type accepts String
    - Values must include at least: “fire”, “water”, “grass”, and “none”
  - Write a Data class called Pokemon
    - Parameters are name (String), number (Int), primaryType (Type), secondaryType (Type)
  - Deompile an instance of the Data

5 Minutes

Solution

```
enum class Type(val type: String) {
    NORMAL("normal"), FIGHTING("fighting"),FLYING("flying"), POISON("poison"), GROUND("ground"),
    ROCK("rock"), BUG("bug"), GHOST("ghost"), STEEL("steel"), FIRE("fire"), WATER("water"),
    GRASS("grass"), ELECTRIC("electric"), PSYCHIC("psychic"), ICE("ice"), DRAGON("dragon"), NONE("none")
}

data class Pokemon(val name: String, val number: Int, val primaryType: Type, val secondaryType: Type)

fun specialClassUsage() {
    val rapidash = Pokemon("Rapidash", 78, Type.FIRE, Type.NONE)

    val (name, number, primaryType, secondaryType) = rapidash

    println(name)
    println(number)
    println(primaryType)
    println(secondaryType)
}
```

A Student Can

Key Value

Pairs

- Return two values
- Store them individually or separately

```
open class Book(val title: String, val author: String, val year: Int = 2019) {

    ...

    fun getCover(): Pair<String, String> {
        return title to author
    }

    fun getDetails(): Triple<String, String, Int> {
        return Triple(title, author, year)
    }

    ...
}

val book = Book("Jed and the Junkyard War", "Stepthen Bohls", 2016)

val (title, author) = book.getCover()
println("Cover: $title - $author")

val details = book.getDetails()
println("Title Page: ${details.first}, ${details.second}, ${details.third}")
```

https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-pair/index.html

Maps

- getOrPut
- putIfAbsent
- getOrDefault
- getOrElse
- any

```
val books = mutableMapOf<String, String>()
books.getOrPut("Dune") { "Herbert" }
books.putIfAbsent("Hamlet", "Shakespeare")

println(books.getOrDefault("Hamlet", "Title"))
books.getOrElse("Bohls") { "Jed" }

books.any { (key, value) -> key.contains("a") || value.contains("a") }
```

https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-mutable-map/index.html

A Student Can

Extension Functions

Extensions

- Additional functionality
- Available as normal class method or property
- Separate from “Core” API
  - Helper Method

```
// Extension functions
fun String.hasSpacesLong(): Boolean {
    val found: Char? = this.find { it == ' ' }
    return found != null
}

fun String.hasSpaces() = find { it == ' ' } != null

// Extension Properties
val String.isLong: Boolean
    get() = length > 10

fun String?.lastCharacter(): Char {
    this?.apply {
        return this[this.length - 1]
    }
}

fun extensionsExample() {
    "Does it have spaces?".hasSpaces() // true
    "adsfasdf".isLong
    "adsfasd".hasSpacesLong()
}

fun nullableExample() {
    val word: String? = null
    println(word)
    word.lastCharacter()
}
```

https://kotlinlang.org/docs/reference/extensions.html

A Student Can

Corountines

Coroutines

https://kotlinlang.org/docs/reference/coroutines-overview.html