CSE 310 – Kotlin Workshop

Example Classroom Code

- Starting Code: https://replit.com/@cmacbeth/CSE310KotlinWorkshop
- Solution Code: https://replit.com/@cmacbeth/CSE310KotlinWorkshopSolution

Useful Reference Links

- https://kotlinlang.org/docs/home.html
- https://play.kotlinlang.org/byExample/overview
- https://www.w3schools.com/kotlin/index.php

Development Environment

Kotlin is a compiled language that is built using the Java JDK. Kotlin was written by JetBrains which is the same provider of the IntelliJ IDE. There are two ways to install Kotlin:

- IntelliJ IDE
  - Pre-packaged with Kotlin Extensions
  - https://www.jetbrains.com/idea/download
- Command Line
  - https://kotlinlang.org/docs/command-line.html

Similair to Java, code is compiled using the `kotlinc` tool. Look in the `bin` directory of your Kotlin installation for this tool. If you are running from the command line, you will want to add this to your environment path. This example compiles three `.kt` files into a `.jar` file.

```
kotlinc -d my_software.jar hello.kt goodbye.kt start.kt
```

After compiling into the `.jar` file, the software can now be executed using the `kotlin` tool which uses the Java Virtual Machine (JVM).

```
kotlin -classpath my_software.jar StartKt
```

In your Kotlin code, one of the files will have a `main` function that defines where the code begins. In our example above, the `start.kt` file has the `main` function. When running the `java` tool, we specify this parameter using the format `[filename]Kt`. Therefore, the last parameter in the kotlin command is `StartKt`.

Kotlin Syntax

Variables

In Kotlin, variables must be declared as either mutable (changable) or immutable (unchangeable). The identification of the data type is usually optional (if setting something to null, you would need to specify the data type).

```
val x : Int = 5
var y = 5            // Kotlin figures out this is an Int automatically

x = 6 // Can't do this
y = 6 // OK
```

If the variable

One of the benfits of Kotlin is the ability for the compiler to check for potential Null Pointer Exceptions. By default, all variables are prevented from being equal to `null`. If we forget to create an object to assign to a variable and then subsequently try to call a function with that variable, the compiler will give us an error.

If we want a variable to have a value of null, then we need to specify this during the declaration.

```
var wordCouldBeNull : String?
```

Conditionals

Kotlin supports standard `if`, `else if` and `else` conditional blocks as other languages. Kotlin also supports a convienant `when` block which in many cases can replace the traditional `if` blocks. The cases in the `when` block are done from top to bottom like an `else if`. The value after the `->` is used to assign the variable.

```
val letterGrade = when {
    grade >= 90  ->  "A"
    grade >= 80  ->  "B"
    grade >= 70  ->  "C"
    grade >= 60  ->  "D"
    else         ->  "F"
}
```

Loops

The `for` keyword can be used to either loop through a range of numbers or a collection of items (e.g. data structure).

```
for (i in 1..10) {
  // Do something 10 times
}
```

```
val data = listOf(1,2,3,4,5)
for (number in data) {
  // Do something with each number in the data list
}
```

Kotlin also supports `while` and `do while` loops.

Functions

Kotlin uses the `fun` keyword to define a function. Functions can have parameters and optional return types. Native types are passed by value while objects are passed by pointer. In other words, all parameters are considered `val`.

```
fun divide(value1 : Float, value2 : Float) : Float? {
  if (value2 == 0.0f) return null

  return value1 / value2
}
```

Data Structures

There are two kinds of lists frequently used in Kotlin:

- List - An immutable list of data type T
- MutableList - A mutable list of data type T

There are built-in-functions to help you create a list of either type.

```
val diceValues = listOf(1,2,3,4,5,6)
val diceRolls : MutableList<Int> = mutableListOf<Int>()
diceRolls.add(3)
diceRolls.add(4)
diceRolls.add(1)
```

In the example above, notice that `diceRolls` needs a data type identified. This is needed because we are starting with an empty list and Kotlin can not automatically identify the data type we want to store. Note that only the `MutableList` object can call the `add` function.

Also notice that both objects above are `val` or immutable. This is referring to the variable and not the ability to mutate or change the object internally.

```
val diceRolls : MutableList<Int> = mutableListOf<Int>()
diceRolls = mutableListOf(2,4,6) // NOT ALLOWED ... CAN'T CHANGE WHAT THE VARIABLE STORES
```

Classes and Inheritance

Kotlin supports classes like C++ and Java. There are multiple ways to define constructors but a convienant method is to include the constructor in the first line of the class definition.

```
class Person(val name : String, var age : Int) {

}

fun main() {
  val me = Person("Bob", 34)
}
```

In this example, `name` and `age` are both attributes of the class. The `age` can be changed.

The default scope for attributes and functions in a class is `public`. The keyword `private` can be used to change this.

To inherit another class, the base class must be identified as `open`. If a function can be overriden in a base class, the function in the base class must also be identified as `open` and the funciton in the derived class must be identified as `override`. To call an overriden function in the base class, the `super` keyword is used.

```
open class Person(val name : String, var age : Int) {

  open fun display() {
    println("$name ($age)")
  }

}

class Student(_name : String, _age : Int, var gpa : Float) : Person(_name, _age) {

  override fun display() {
    super.display()
    println("gpa = $gpa")
  }

}
```