Hector Sanchez

Samantha Felzein

CSc 372

Dr. Colberg

20 April 2020

Scala: Subprograms

Scala has many useful subroutine features that may often seem familiar to more common languages such as Java or C. First, we start with what is regarded to as a trait in Scala. A trait in Scala is similar in appearance to a struct in C programming, however they are not similar beyond the appearance. A trait in Scala serves as a form of encapsulation for methods and fields, and unlike classes in Java which can only inherit from a single superclass, multiple traits can be inherited at once. Traits are somewhat similar to abstract classes in Java.

```
trait Derp {
  val phrase: String
  def isDigit(x: Int): Boolean
  def isNotDigit : (x: Int): Boolean =
  !isDigit(x)
}
```

Above we can see the unique syntax of the Scala trait in which phrase simply defines a feature of the trait Derp, while the following lines define two functions. A trait can be invoked within the function declaration by simply adding *extends {trait name}* to the end of the function. From here you can use the above defined "isDigit" method (for example) just as any other method extension available to you (i.e. *val valWithinFunc = i.isDigit).*

Coming back around to the essential subprogram features, we have the function. In Scala the function is much the same as functions in just about any other programming language, it requires a function declaration, a body, and some kind of invocation from somewhere else.

```
def addInt(a: Int, b: Int): Int = {
    var sum: Int = 0
    sum = a + b
    return sum
  }
```

Here we have a simple declaration of a function that adds two integers. We can note the function declaration and its syntax is very familiar to us in the sense that we must declare our parameters and their types and what type of response we will send. Scala, allows us to name these incoming variables within this declaration for later use within the body. As for the declaration of what we will return, this can be found right after our parameters, in this case we have *:Int*, therefore we can expect and integer output. When left blank, this function will return nothing. As for calling a function it is the standard syntax (*addInt(1,2),* for this example*)*.

Finally, we come to the Scala class. In Scala class we find many useful features such as the ability to create a blueprint for objects and define a method within the class, thus allowing you to use this method as an extension of any object you implement of this class. Similar to the Java class, classes in Scala also have constructor declarations which must be set upon invoking a new object.

```
class Example(val x: Int, val y: Int) {
        var a: Int = x
        var b: Int = y

        def add(c: Int, d: Int) {
            a = a + c
            b = b + d
            println ("a: " + a);
            println ("b: " + b);
        }
}
```

Here we see the class “Example” in which we define the two parameters x and y, later assigned to a variable and furthermore used in the add function available when you implement this object. The “Example” object can later be created using the following syntax: *val ex = new*

Example(1, 2); From here we can then invoke the "add" method found within "Example" by saying *ex.add(10,10);*. Extending and inheriting from a class in Scala is also identical to Java since we must also use the *extends* keyword within our function declaration. Within Scala classes you can also create singleton objects and implicit classes by declaring them as such or following the necessary syntax.

Sources:

"Scala - Classes & Objects." *Tutorialspoint*, www.tutorialspoint.com/scala/scala_classes_objects.htm.

"Scala - Functions." *Tutorialspoint*, www.tutorialspoint.com/scala/scala_functions.htm.

"Scala - Traits." *Tutorialspoint*, www.tutorialspoint.com/scala/scala_traits.htm.

"Traits." *Scala Documentation*, docs.scala-lang.org/tour/traits.html.