Unit 3d: Scala Classes

1 Classes

In Java source code is written as classes; objects are instances of classes created with the *new* key word. Methods and values that aren't associated with individual instances of a class are declared as static.

In Scala source code can be written as classes and objects are instances of classes created with *new* key word, or instances of value types such as Int. You can define *singleton objects*, denoted by using the keyword *object* instead of *class*; unlike instances of classes, there can only be one singleton object of a given type. Singleton objects usually contain methods and values that aren't associated with individual instances of a class. You may sometimes create *companion objects* for classes.

1.1 Main method

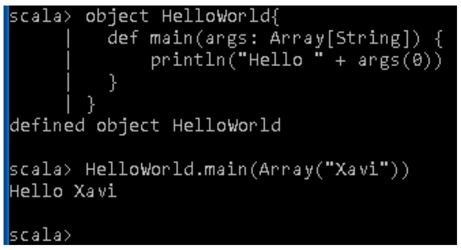
In Java the *main* method is the entry point for a Java application and is declared inside a class; it doesn't matter which class as it doesn't belong to any object – it is a static method. Commonly we have an application class which exists to host the *main* method.

In Scala the *main* method is the entry point for a Scala application and is declared inside a singleton object. Alternatively we can put entry point code inside an object that extends *App*.

```
object Hello {
    def main(args: Array[String]) {
        println("Hello, world")
    }
}
```

OR

```
object Hello extends App {
    println("Hello, world")
}
```



```
scala> object HelloWorld{
     |   def main(args: Array[String]) {
     |       println("Hello " + args(0))
     |   }
     | }
defined object HelloWorld

scala> HelloWorld.main(Array("Xavi"))
Hello Xavi

scala>
```

1.2 Constructors

Java constructor(s) have same the name as the class and are often used to initialise fields.

In Scala the primary constructor is the class body, parameters come after the class name. Constructor parameters are fields, val by default but can be declared as var.

```
class Greeter(message: String) {
    println("A greeter is being instantiated")
    def SayHi() = println(message)
}
```

This class has a field called *message* and a method called *SayHi* and a line of code that runs as object is created.

You can have auxiliary constructors, to support different parameter lists, named *this*, that call the primary constructor.

1.3 Getters and setters

In Java the most common access modifiers for fields are *public* and *private*; you need to manually create getters and/or setters for private fields (although this is a common refactoring in IDEs).

In Scala the compiler can generate getters/setters, although you can add your own. If a field is declared as a *var*, the compiler generates both getter and setter methods for that field. If the field is a *val*, the compiler generates only a getter method for it. If a field doesn't have a *var* or *val* modifier, compiler doesn't generate a getter or setter method for the field. Additionally, *var* and *val* fields can be modified with the *private* keyword, which prevents getters and setters from being generated.

2 Inheritance

Java has single inheritance i.e. a class can be a subclass of one superclass. A class can implement multiple interfaces and needs to implement the methods declared in each one (In Java 8, though, interfaces can contain implementations of *default* (or *defender*) methods).

A Scala class can be a subclass of one superclass; multiple inheritance is supported by *traits*. Traits are like interfaces but implement methods. A class can *extend* a trait and *"mix-in"* other traits – we will look at this in detail later.