

Inleiding

Voor de opdracht van het zelf aanleren van een nieuwe functionele taal voor het vak APP in het ASD semester hebben ik gekozen voor de taal Scala. De reden hiervoor is dat Scala veel wordt gebruikt en ik het idee heb dat ik later in het werkveld Scala zal gebruiken als ontwikkeltaal. Hierdoor lijkt het mij verstandig om mijzelf Scala aan te leren.

Opdracht

Wat ik wil gaan maken in Scala is een versie van het spel: Rock-Paper-Scissors. Hier heb ik voor gekozen omdat het mij erg interessant lijkt om de spel logica te implementeren in Scala zonder gebruik te maken van een object georiënteerde ontwikkelmethode en in plaats daarvan een functionele ontwikkelmethode.

Inhoudsopgave

Opstart	2
IDEA	2
Project	2
Scala	2
Scala Worksheet	3
sbt	3
Testen	4
Opdracht uitprogrammeren	5
Main	5
Variabelen	5
Objecten	6
Methodes	6
Overig	7
Problemen	7
HashMap	7
Conclusie	8

Opstart

IDEA

Als IDEA heb ik gekozen voor IntelliJ met daarbij gebruik makend van een door JetBrains ontwikkelde plugin voor Scala projecten. IntelliJ heb ik in vorige projecten vaak gebruikt en ben ik al bekend mee. Na wat zoeken op het internet kwam ik erachter dat IntelliJ door middel van een plugin ondersteuning biedt voor Scala's syntax, highlighting, auto-complete en veel meer functies die ik ook heb gebruikt voor het ontwikkelen van Java applicaties in IntelliJ

Project

Voor het project heb ik gebruik gemaakt van sbt. Sbt is een tool voor het compilen, runnen en testen van Scala projecten. sbt is vergelijkbaar met Maven en Gradle.

Scala

Om te beginnen heb ik eerst er voor gezorgd dat ik de nieuwste versie van Scala heb geïnstalleerd. Door de tutorial van de link hieronder te volgen wijst het zich vanzelf.

Tijdens de tutorial moest ik een nieuwe Scala Class aanmaken. Ik had de keuze uit twee soorten Scala classes: Een gewone Class en een Case Class. Na opzoeken kwam ik erachter dat een gewone Class hetzelfde is als in Java maar, een Case Class is een gewone Class maar met een paar verschillen. Een Case Class is goed voor het modelleren van onaanpasbare data. Dit is zeer handig voor dit project aangezien de opdracht is om functioneel programmeren en één van de principes van functioneel programmeren is het niet kunnen aanpassen van variabelen. Volgens het document wordt voor variabelen die constant zijn het type *val* gebruikt. Dit datatype is onaanpasbaar in tegenstelling tot *var* in Scala.

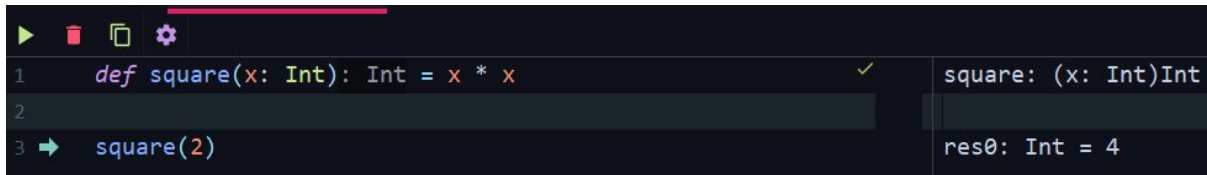
Na de klasse te hebben aangemaakt vraagt de tutorial om deze te vervangen naar een object. Na wat onderzoek lijkt het erop dat class en object weinig van elkaar verschillen. Class is hetzelfde als in Java maar, een object in Scala is een *singleton* object. Een object is ook vooral bedoeld voor statische waarden, wat heel handig is bij functioneel programmeren.

<https://docs.scala-lang.org/getting-started/intellij-track/getting-started-with-scala-in-intellij.html>
!

Scala Worksheet

Vervolgens moest ik van de tutorial een Scala Worksheet maken. Een Scala worksheet is bedoeld om functies in te schrijven. Dit gebeurt dus niet in het object of class zoals bij Java. Als ik het worksheet run krijg ik de uitkomsten van functies die ik heb gemaakt en aanroep terug zonder gebruik te maken van een print statement. Dit vind ik zeer handig omdat je nu niet je volledige applicatie of de debugger hoeft te runnen om een methode te testen.

Voorbeeld:



```
1  def square(x: Int): Int = x * x
2
3  square(2)
```

square: (x: Int)Int
res0: Int = 4

Verder viel het mij op dat Scala geen gebruik maakt van ";" aan het einde van een statement. Dit was voor mij even wennen aangezien ik veel met Java heb gewerkt.

sbt

Bij de tutorial van sbt werd gevraagd om een nieuw project te maken. Dit ging vrij soepel als je goed de tutorial leest. Bepaalde versie van JDK wordt gevraagd en ik werkte eerst met een hogere versie dus dit was niet compatibel maar, het is na goed lezen toch gelukt. Sbt heeft een vergelijkbare file structure als Maven.

Het toevoegen van dependencies in sbt gaat niet volgens de xml structuur zoals in Maven. sbt heeft hier zijn eigen bestand met bestandstype voor namelijk *build.sbt*. De opbouw van het bestand lijkt veel op het declareren van variabelen. Voorbeeld:

```
name := "SbtExampleProject"
```

```
version := "0.1"
```

```
scalaVersion := "2.13.1"
```

<https://docs.scala-lang.org/getting-started/intelliJ-track/building-a-scala-project-with-intelliJ-and-sbt.html>

Testen

Voor het testen van code gebruikt Scala, net als bij testen met Java en JUnit, dependencies. Naast het importeren van de test library, net als bij JUnit, extend de test klasse de klasse van de geïmporteerde library. Daarnaast maak je geen aparte methoden aan per test in de klasse, maar gebruik je het kernwoord 'test' en daar achter tussen haakjes de naam van de test. Vervolgens gebruik je op de gebruikelijke wijze het kernwoord 'assert' en roep je daarin de klasse en de naam van de methode aan met eventuele parameter(s) en wat de uitkomst moet zijn. Voorbeeld:

```
import org.scalatest.FunSuite

class CubeCalculatorTest extends FunSuite {
  test("CubeCalculator.cube") {
    assert(CubeCalculator.cube(3) === 27)
    assert(CubeCalculator.cube(0) === 0)
  }
}
```

CubeCalculator.cube berekent de kubus op basis van de meegegeven waarde, dus 3 wordt $3 * 3 * 3 = 27$. Deze test zal dus slagen en dit wordt op dezelfde wijze, zoals bij JUnit, in IntelliJ weergegeven met een overzicht van alle gedraaide testen onderaan het scherm.

<https://docs.scala-lang.org/getting-started/intellij-track/testing-scala-in-intellij-with-scalatest.html>

Opdracht uitprogrammeren

Voor het uitprogrammeren van Steen-Papier-Schaar maak ik gebruik van een aangepaste versie van een State-Machine. Aan het einde van iedere methode roep ik de volgende methode aan. Hierdoor is het aanroepen van de volgende methode een soort exit state. Wanneer er wordt gewacht op een input door middel van *readLine* in Scala is dit een soort *do state*.

Main

Om te beginnen heb ik een nieuw object gemaakt 'Game'. Dit object extends 'App', dit zorgt ervoor dat ik vanuit Game mijn spel kan runnen. Dit is vergelijkbaar met de Main methode in Java. Voorbeeld:

```
object Game extends App {}
```

Daarnaast heb ik ook een case class gemaakt genaamd GameLogic. Hierin sla ik alle spel logica op bijvoorbeeld het bepalen wie er heeft gewonnen. Vervolgens heb ik een case class GameVariables gemaakt met alle variabelen in het spel erin. Dit om alle variabelen overzichtelijk bij elkaar te hebben staan.

<https://docs.scala-lang.org/tour/case-classes.html>

Variabelen

In GameVariables heb ik alle variabelen binnen het spel staan voor het bijhouden van de score, speler input, enzovoorts. Voor het bijhouden van de score en voor de inputs heb ik het type *var* gebruikt. Normaal gesproken mag je bij functioneel programmeren geen variabelen wijzigen maar in overleg met de leraar is dit goedgekeurd omdat, bij functioneel programmeren dit een veel voorkomend probleem is.

Het aanmaken van variabelen is anders als bij Java. Bij Scala gebruik je voor het aanmaken van variabelen het woord '*var*' of '*val*' zoals eerder al aangegeven. *var* staat voor een wijzigbare waarde (variable) en *val* staat voor een vaste waarde (value). *Val* is vergelijkbaar met het toevoegen van het woord '*final*' voor een variable.

Bij Scala kies je dus eerst '*var*' of '*val*', daarna de naam van je variabele en vervolgens een ':' met daarachter het datatype als klasse. Daarachter komt pas het toewijzen van de variabele. Voorbeeld:

```
var scorePlayer : Int = 0
```

Objecten

In Scala wordt alles gezien als een object, zelfs methodes. Om bijvoorbeeld de methodes van "GameVariables" te kunnen gebruiken in een andere (case) class is het nodig om een instantie van deze klasse te maken. Het aanmaken van een instantie van een klasse gebeurt niet door het key woord *new* te gebruiken zoals bij Java maar, door middel van het aanroepen van de methode *.apply()* die standaard is ingebouwd bij iedere klasse. Voorbeeld hiervan staat hieronder:

```
val logic : GameLogic = GameLogic.apply()
```

```
logic.startGame()
```

Het maken van een instantie van een object of klasse gaat tegen de principes van Functioneel Programmeren in maar, in overleg met de leraar is dit goed gekeurd omdat er anders geen nette manier is om code te scheiden zonder alles onder elkaar neer te zetten

Methodes

In Scala maak je een methode aan door *def* voor de naam van een methode te zetten. Vervolgens zet je tussen haakjes de eventuele parameters op dezelfde manier als bij variabelen, zoals hieronder te zien, en daarna doe je net als bij variabelen een ':' neerzetten met daarachter het datatype dat de methode teruggeeft. Daarachter zet je een '=' en dan de gebruikelijke curly brackets om de body van de methode te schrijven.

Als een methode een waarde terug geeft, zoals bij de onderstaande methode, is het niet nodig om het keyword *return* te gebruiken zoals bij Java.

```
def getValueOfOption(input: String): Int = {  
    variables.valueOfOption.getOrElse(input, 0)  
}
```

Als een methode niks teruggeeft gebruik je het datatype *Unit*. *Unit* is de Scala variant van *void* in Java. Als een methode in Scala als datatype *Unit* heeft wordt de methode ook wel een *procedure* genoemd. Een voorbeeld van z'n methode staat hieronder.

```
def scoreOverview(): Unit = {  
    println("\nScore:")  
    println("=====")  
    println("You: " + variables.scorePlayer)  
    println("Computer: " + variables.scoreComputer + "\n")  
    playerChoice()  
}
```

https://www.tutorialspoint.com/scala/scala_functions.htm

Overig

In dit gedeelte wordt de overige gevonden informatie over Scala beschreven die ik heb gevonden bij het maken van het project.

Problemen

Na het volgen van de tutorial ben ik begonnen met de opdracht, na de standaard stappen kwam ik erachter dat ik geen Scala Class kon maken in mijn scala folder. Het Scala Framework was niet toegevoegd aan de projectstructuur, na googlen kwam ik erachter dat als je rechts klikt op de root folder van alle mappen je het Scala Framework kan vinden in een lijst onder de knop 'Add framework support...'.

HashMap

Voor het koppelen van de keuzes van de computer en speler aan een Integer waarde heb ik gekozen voor een HashMap. Een HashMap kan op dezelfde wijze als in Java worden aangemaakt maar, ik kwam bij het onderzoeken een meer overzichtelijke manier tegen om dit te verwezenlijken. Voorbeeld:

```
val valueOfOption: HashMap[String, Int] =  
    HashMap("Rock" -> 1, "Paper" -> 2, "Scissors" -> 3)
```

Deze manier lijkt veel op een associatieve array in PHP en ziet er naar mijn mening netter uit dan de HashMaps in Java. Dit komt vooral door het gebruik van '->' dat een toewijzing voorstelt.

Bij een verkrijgen van een waarde uit een HashMap lukte het mij niet om gebruik te maken van *get* zoals in Java. Scala gebruikt hiervoor de methode *getOrElse* waarbij je twee parameters meegeeft. De eerste parameter is de key die je wilt opzoeken. De tweede parameter is de waarde die je teruggeeft als de key niet kan worden gevonden. Naar mijn mening is dit veel netter dan een *if(... != null)* constructie te maken om te controleren of de waarde kan worden gevonden in de HashMap. De code wordt hier veel overzichtelijker door naar mijn mening. Voorbeeld:

```
def getValueOfOption(input: String): Int = {  
    variables.valueOfOption.getOrElse(input, 0)  
}  
  
def getOptionOfValue(input: Int): String = {  
    variables.optionOfValue.getOrElse(input, "")  
}
```

Conclusie

Het aanleren van Scala ging zeer vlot omdat, Scala veel weg heeft van Java. Daarnaast biedt Scala bovenop Java meer functionaliteiten en alhoewel ik zeker niet alle functionaliteiten van Scala heb benut vindt ik dat ik toch de conventies en syntax goed beheers.

Scala verschilt op veel aspecten qua syntax van Java maar het voelde naar mijn mening zeer vertrouwd. Ik ben geen abnormale constructies tegenkomen tijdens het leren van Scala. Scala is over het algemeen vergelijkbaar met Java maar, biedt naast de standaard Java taal verschillende uitbreidingen. Dit zorgt ervoor dat Scala een betere versie is van standaard Java.