Olasunkanmi Olayinka – SEC01 (NUID 001512266)

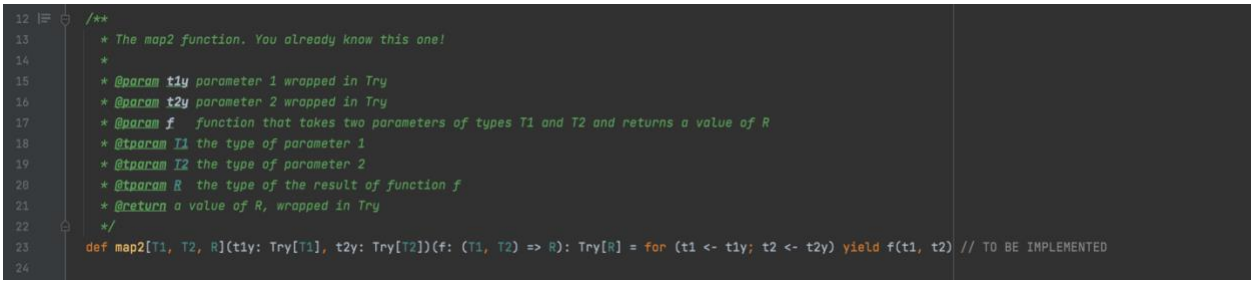
Big Data System Engineering with Scala
Fall 2022
Assignment No. 05

-List of Tasks Implemented

- Implement Function.scala:
    1. map2
    2. map3
    3. map7
    4. lift
    5. lift2
    6. lift3
    7. lift7
    8. invert2
    9. invert3
    10. invert4
    11. uncurried2
    12. uncurried3
    13. uncurried7
- Implement Movie.scala:
    1. MoviesProtocol
    2. testSerializationAndDeserialization

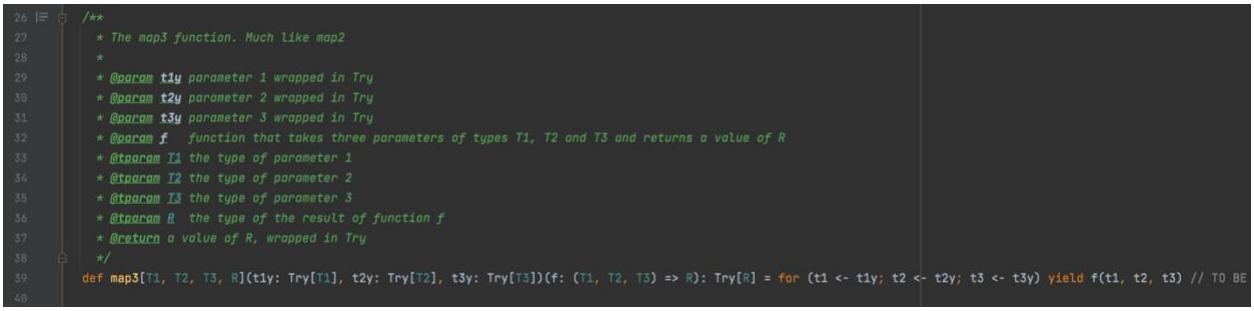
-Code

map2:



```scala
/**
 * The map2 function. You already know this one!
 *
 * @param t1y parameter 1 wrapped in Try
 * @param t2y parameter 2 wrapped in Try
 * @param f   function that takes two parameters of types T1 and T2 and returns a value of R
 * @tparam T1 the type of parameter 1
 * @tparam T2 the type of parameter 2
 * @tparam R  the type of the result of function f
 * @return a value of R, wrapped in Try
 */
def map2[T1, T2, R](t1y: Try[T1], t2y: Try[T2])(f: (T1, T2) => R): Try[R] = for (t1 <- t1y; t2 <- t2y) yield f(t1, t2) // TO BE IMPLEMENTED
```

map3:



```scala
/**
 * The map3 function. Much like map2
 *
 * @param t1y parameter 1 wrapped in Try
 * @param t2y parameter 2 wrapped in Try
 * @param t3y parameter 3 wrapped in Try
 * @param f   function that takes three parameters of types T1, T2 and T3 and returns a value of R
 * @tparam T1 the type of parameter 1
 * @tparam T2 the type of parameter 2
 * @tparam T3 the type of parameter 3
 * @tparam R  the type of the result of function f
 * @return a value of R, wrapped in Try
 */
def map3[T1, T2, T3, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3])(f: (T1, T2, T3) => R): Try[R] = for (t1 <- t1y; t2 <- t2y; t3 <- t3y) yield f(t1, t2, t3) // TO BE
```

map7:

```
/**
  * You get the idea...
  */
def map7[T1, T2, T3, T4, T5, T6, T7, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3], t4y: Try[T4], t5y: Try[T5], t6y: Try[T6], t7y: Try[T7])
                                       (f: (T1, T2, T3, T4, T5, T6, T7) => R): Try[R] = for (t1 <- t1y; t2 <- t2y; t3 <- t3y; t4 <- t4y; t5 <- t5y; t6 <- t6y; t7 <- t7y
```

lift:

```
/**
  * Lift function to transform a function f of type T=>R into a function of type Try[T]=>Try[R]
  *
  * @param f the function we start with, of type T=>R
  * @tparam T the type of the parameter to f
  * @tparam R the type of the result of f
  * @return a function of type Try[T]=>Try[R]
  */
// You know this one
def lift[T, R](f: T => R): Try[T] => Try[R] = _ map f // TO BE IMPLEMENTED
```

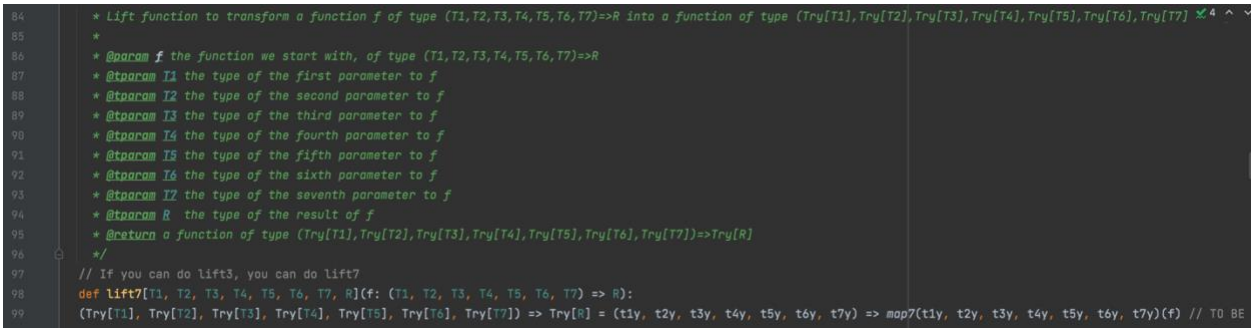
lift2:

```
/**
  * Lift function to transform a function f of type (T1,T2)=>R into a function of type (Try[T1],Try[T2])=>Try[R]
  *
  * @param f the function we start with, of type (T1,T2)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2])=>Try[R]
  */
// Think Simple, Elegant, Obvious
def lift2[T1, T2, R](f: (T1, T2) => R): (Try[T1], Try[T2]) => Try[R] = (t1y, t2y) => map2(t1y, t2y)(f) // TO BE IMPLEMENTED
```

lift3:

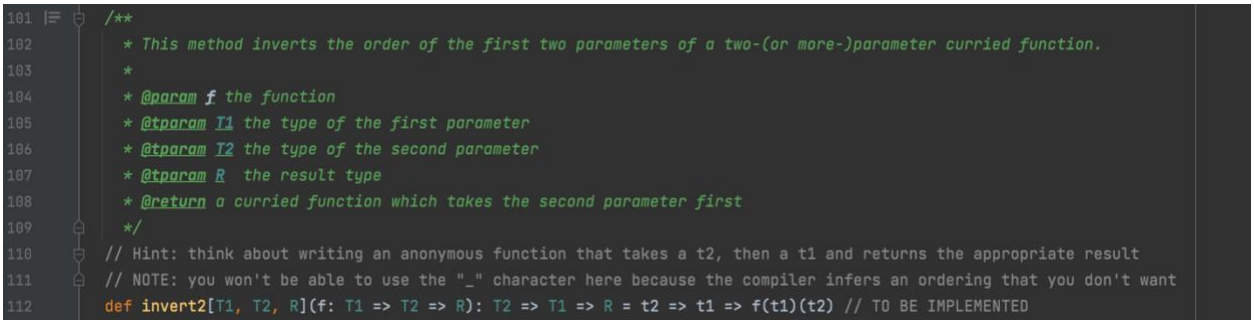
```
/**
  * Lift function to transform a function f of type (T1,T2,T3)=>R into a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
  *
  * @param f the function we start with, of type (T1,T2,T3)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam T3 the type of the third parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2],Try[T3])=>Try[R]
  */
// If you can do lift2, you can do lift3
def lift3[T1, T2, T3, R](f: (T1, T2, T3) => R): (Try[T1], Try[T2], Try[T3]) => Try[R] = (t1y, t2y, t3y) => map3(t1y, t2y, t3y)(f) // TO BE IMPLEMENTED
```

lift7:



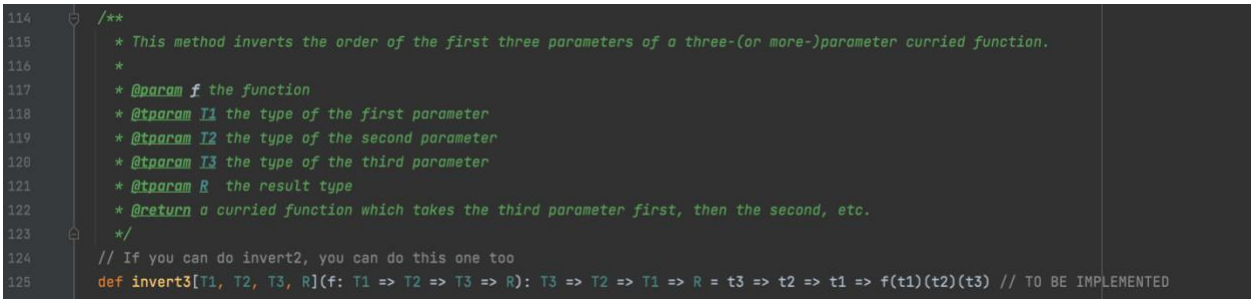
```
  * Lift function to transform a function f of type (T1,T2,T3,T4,T5,T6,T7)=>R into a function of type (Try[T1],Try[T2],Try[T3],Try[T4],Try[T5],Try[T6],Try[T7]
  *
  * @param f the function we start with, of type (T1,T2,T3,T4,T5,T6,T7)=>R
  * @tparam T1 the type of the first parameter to f
  * @tparam T2 the type of the second parameter to f
  * @tparam T3 the type of the third parameter to f
  * @tparam T4 the type of the fourth parameter to f
  * @tparam T5 the type of the fifth parameter to f
  * @tparam T6 the type of the sixth parameter to f
  * @tparam T7 the type of the seventh parameter to f
  * @tparam R  the type of the result of f
  * @return a function of type (Try[T1],Try[T2],Try[T3],Try[T4],Try[T5],Try[T6],Try[T7])=>Try[R]
  */
// If you can do lift3, you can do lift7
def lift7[T1, T2, T3, T4, T5, T6, T7, R](f: (T1, T2, T3, T4, T5, T6, T7) => R):
(Try[T1], Try[T2], Try[T3], Try[T4], Try[T5], Try[T6], Try[T7]) => Try[R] = (t1y, t2y, t3y, t4y, t5y, t6y, t7y) => map7(t1y, t2y, t3y, t4y, t5y, t6y, t7y)(f) // TO BE
```

invert2:



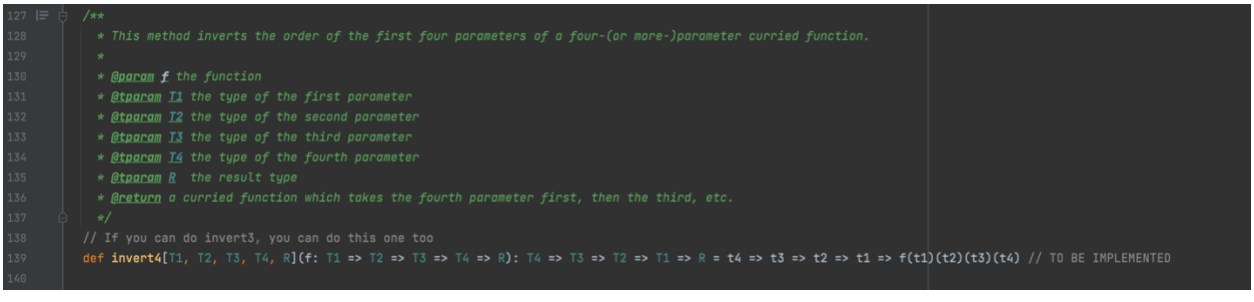
```
/**
  * This method inverts the order of the first two parameters of a two-(or more-)parameter curried function.
  *
  * @param f the function
  * @tparam T1 the type of the first parameter
  * @tparam T2 the type of the second parameter
  * @tparam R  the result type
  * @return a curried function which takes the second parameter first
  */
// Hint: think about writing an anonymous function that takes a t2, then a t1 and returns the appropriate result
// NOTE: you won't be able to use the "_" character here because the compiler infers an ordering that you don't want
def invert2[T1, T2, R](f: T1 => T2 => R): T2 => T1 => R = t2 => t1 => f(t1)(t2) // TO BE IMPLEMENTED
```

invert3:



```
/**
  * This method inverts the order of the first three parameters of a three-(or more-)parameter curried function.
  *
  * @param f the function
  * @tparam T1 the type of the first parameter
  * @tparam T2 the type of the second parameter
  * @tparam T3 the type of the third parameter
  * @tparam R  the result type
  * @return a curried function which takes the third parameter first, then the second, etc.
  */
// If you can do invert2, you can do this one too
def invert3[T1, T2, T3, R](f: T1 => T2 => T3 => R): T3 => T2 => T1 => R = t3 => t2 => t1 => f(t1)(t2)(t3) // TO BE IMPLEMENTED
```

invert4:



```
/**
  * This method inverts the order of the first four parameters of a four-(or more-)parameter curried function.
  *
  * @param f the function
  * @tparam T1 the type of the first parameter
  * @tparam T2 the type of the second parameter
  * @tparam T3 the type of the third parameter
  * @tparam T4 the type of the fourth parameter
  * @tparam R  the result type
  * @return a curried function which takes the fourth parameter first, then the third, etc.
  */
// If you can do invert3, you can do this one too
def invert4[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): T4 => T3 => T2 => T1 => R = t4 => t3 => t2 => t1 => f(t1)(t2)(t3)(t4) // TO BE IMPLEMENTED

```

uncurried2:

```
/**
 * This method uncurries the first two parameters of a three- (or more-)
 * parameter curried function.
 * The result is a (curried) function whose first parameter is a tuple of the first two parameters of f;
 * whose second parameter is the third parameter, etc.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam R  the result type of function f
 * @return a (curried) function of type (T1,T2)=>T4=>R
 */
// This one is a bit harder. But again, think in terms of an anonymous function that is what you want to return
def uncurried2[T1, T2, T3, R](f: T1 => T2 => T3 => R): (T1, T2) => T3 => R = (t1, t2) => t3 => f(t1)(t2)(t3) // TO BE IMPLEMENTED
```

uncurried3:

```
/**
 * This method uncurries the first three parameters of a four- (or more-)
 * parameter curried function.
 * The result is a (curried) function whose first parameter is a tuple of the first three parameters of f;
 * whose second parameter is the third parameter, etc.
 *
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam T4 the type of the fourth parameter
 * @tparam R  the result type of function f
 * @return a (curried) function of type (T1,T2,T3)=>T4=>R
 */
// If you can do uncurried2, then you can do this one
def uncurried3[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): (T1, T2, T3) => T4 => R = (t1, t2, t3) => t4 => f(t1)(t2)(t3)(t4) // TO BE IMPLEMENTED
```

uncurried7:

```
 * @param f the function
 * @tparam T1 the type of the first parameter
 * @tparam T2 the type of the second parameter
 * @tparam T3 the type of the third parameter
 * @tparam T4 the type of the fourth parameter
 * @tparam R  the result type of function f
 * @return a (curried) function of type (T1,T2,T3)=>T4=>R
 */
// If you can do uncurried3, then you can do this one
def uncurried7[T1, T2, T3, T4, T5, T6, T7, T8, R](f: T1 => T2 => T3 => T4 => T5 => T6 => T7 => T8 => R): (T1, T2, T3, T4, T5, T6, T7) => T8 => R =
  (t1, t2, t3, t4, t5, t6, t7) => t8 => f(t1)(t2)(t3)(t4)(t5)(t6)(t7)(t8) // TO BE IMPLEMENTED

```

MoviesProtocol:

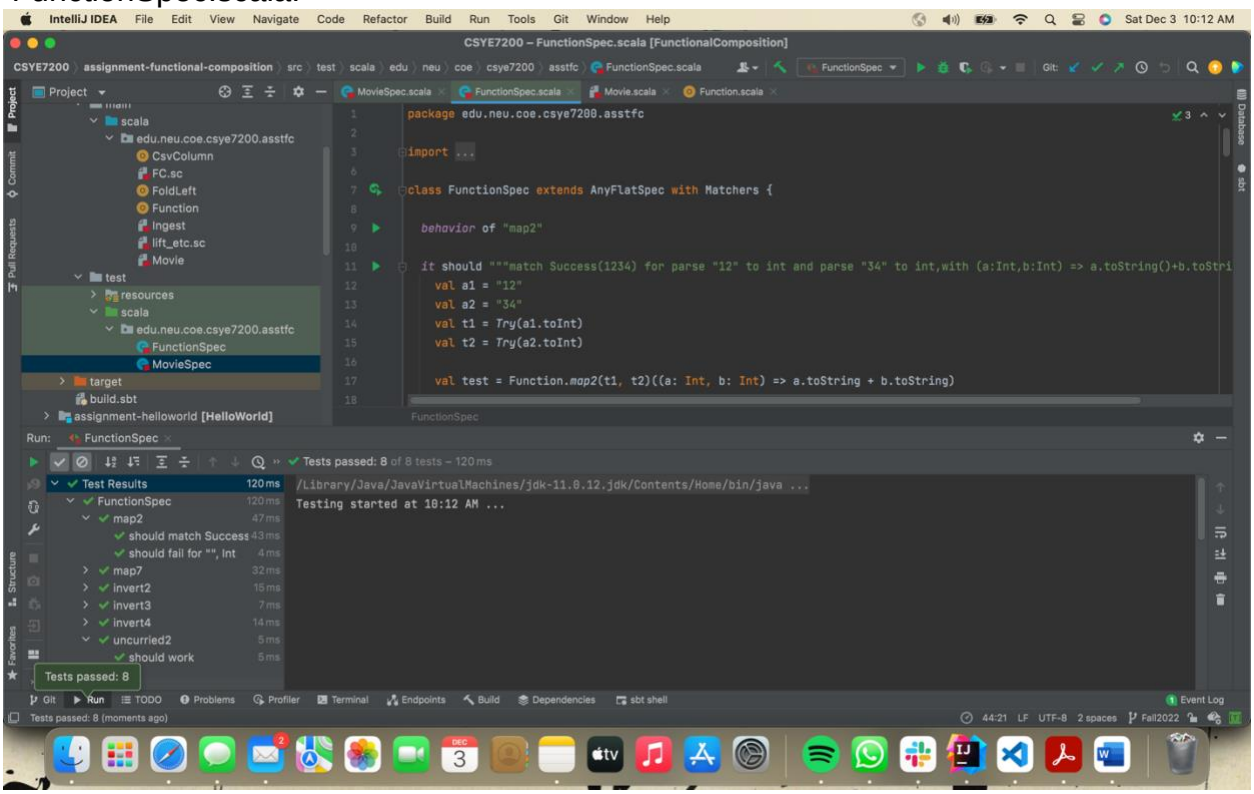
```
//Hint: You may refer to the slides discussed in class for how to serialize object to json
object MoviesProtocol extends DefaultJsonProtocol {
  // 20 points
  // TO BE IMPLEMENTED
  implicit val formatFormat = jsonFormat4(Format.apply)
  implicit val nameFormat = jsonFormat4(Name.apply)
  implicit val ratingFormat = jsonFormat2(Rating.apply)
  implicit val reviewFormat = jsonFormat7(Reviews.apply)
  implicit val productionFormat = jsonFormat4(Production.apply)
  implicit val principalFormat = jsonFormat2(Principal.apply)
  implicit val movieJsonFormat = jsonFormat11(Movie.apply)
}
```

testSerializationAndDeserialization:

```scala
//Hint: Serialize the input to Json format and deserialize back to Object, check the result is still equal to original input.
def testSerializationAndDeserialization(ms: Seq[Movie]): Boolean = {
  // 5 points
  // TO BE IMPLEMENTED
  import MoviesProtocol._
  val json = ms.toJson
  val movies = json.convertTo[Seq[Movie]]
  ms == movies
}
```

-Unit tests

FunctionSpec.scala:



MovieSpec.scala: