Sameeksha Gurrala – SEC01 (NUID 002922061)

Big Data System Engineering with Scala
Spring 2023
Assignment No. #5

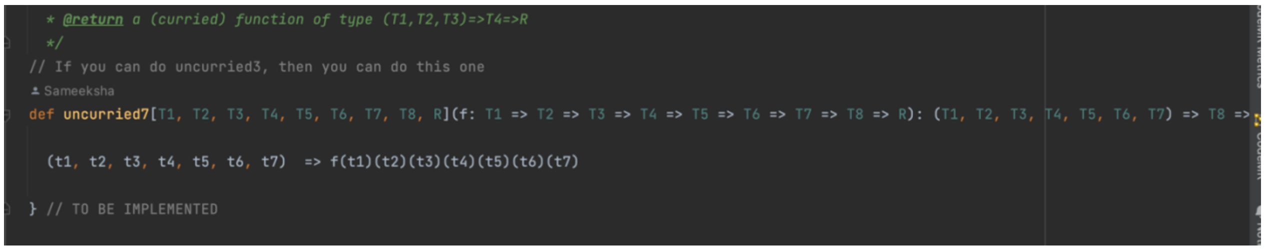
-List of Tasks Implemented

There are 13 TODOs in *Function.scala* and 2 TODOs in *Movie.scala*.

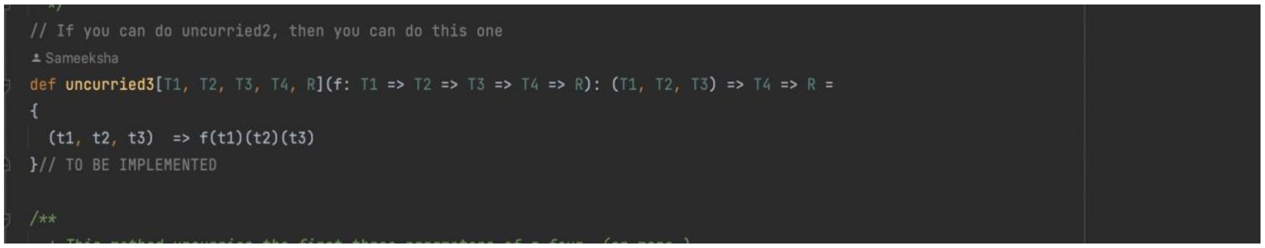
-Findings and analysis

-Code

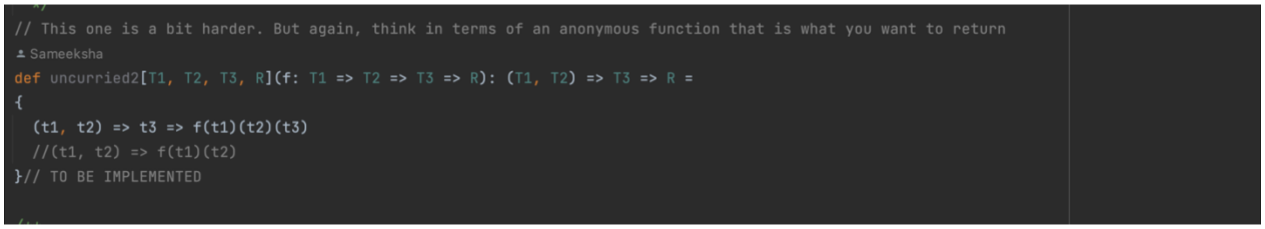
https://github.com/Sameeksha-11/CSYE7200/tree/a-5



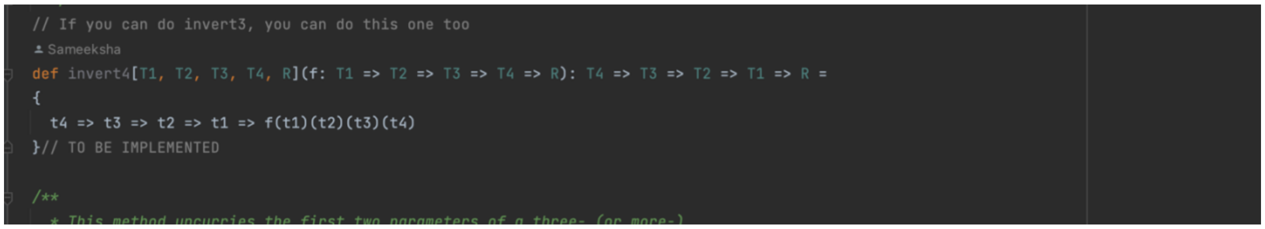
```
  * @return a (curried) function of type (T1,T2,T3)=>T4=>R
  */
// If you can do uncurried3, then you can do this one
Sameeksha
def uncurried7[T1, T2, T3, T4, T5, T6, T7, T8, R](f: T1 => T2 => T3 => T4 => T5 => T6 => T7 => T8 => R): (T1, T2, T3, T4, T5, T6, T7) => T8 =>

  (t1, t2, t3, t4, t5, t6, t7)  => f(t1)(t2)(t3)(t4)(t5)(t6)(t7)

} // TO BE IMPLEMENTED
```



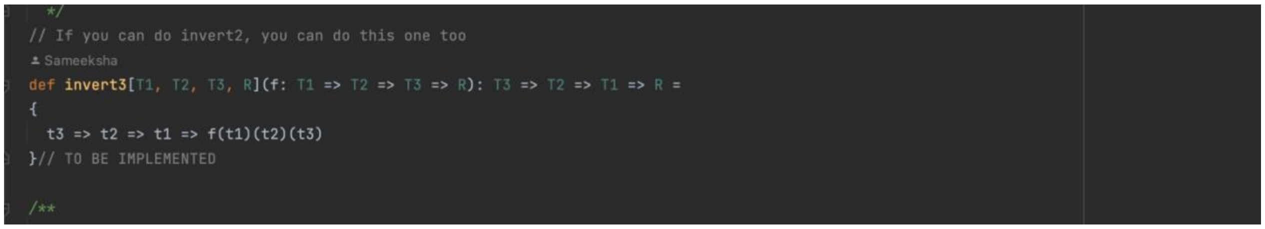
```
// If you can do uncurried2, then you can do this one
Sameeksha
def uncurried3[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): (T1, T2, T3) => T4 => R =
{
  (t1, t2, t3)  => f(t1)(t2)(t3)
}// TO BE IMPLEMENTED

/**
```



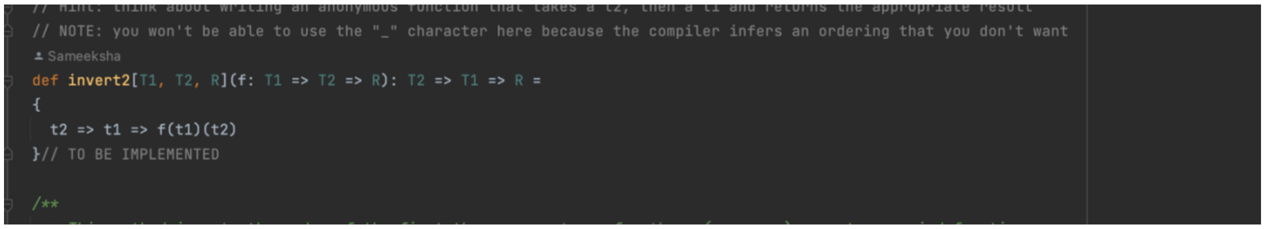
```
// This one is a bit harder. But again, think in terms of an anonymous function that is what you want to return
Sameeksha
def uncurried2[T1, T2, T3, R](f: T1 => T2 => T3 => R): (T1, T2) => T3 => R =
{
  (t1, t2) => t3 => f(t1)(t2)(t3)
  //(t1, t2) => f(t1)(t2)
}// TO BE IMPLEMENTED
```



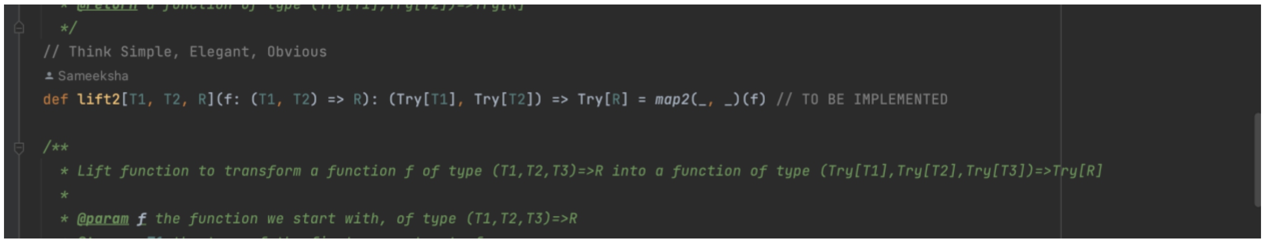
```
// If you can do invert3, you can do this one too
Sameeksha
def invert4[T1, T2, T3, T4, R](f: T1 => T2 => T3 => T4 => R): T4 => T3 => T2 => T1 => R =
{
  t4 => t3 => t2 => t1 => f(t1)(t2)(t3)(t4)
}// TO BE IMPLEMENTED

/**
  * This method uncurries the first two parameters of a three- (or more-)
```



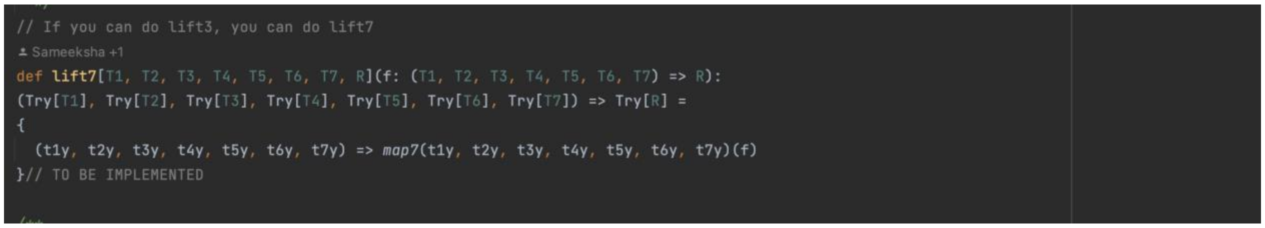
```
  */
// If you can do invert2, you can do this one too
Sameeksha
def invert3[T1, T2, T3, R](f: T1 => T2 => T3 => R): T3 => T2 => T1 => R =
{
  t3 => t2 => t1 => f(t1)(t2)(t3)
}// TO BE IMPLEMENTED

/**
```



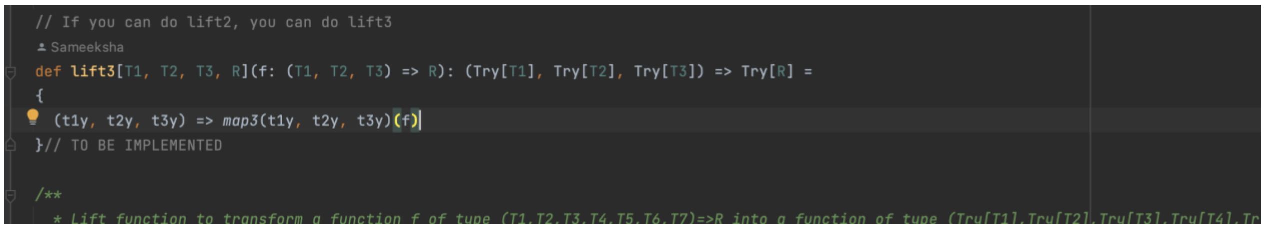
```
// Hint: think about writing an anonymous function that takes a t2, then a t1 and returns the appropriate result
// NOTE: you won't be able to use the "_" character here because the compiler infers an ordering that you don't want
Sameeksha
def invert2[T1, T2, R](f: T1 => T2 => R): T2 => T1 => R =
{
  t2 => t1 => f(t1)(t2)
}// TO BE IMPLEMENTED

/**
```



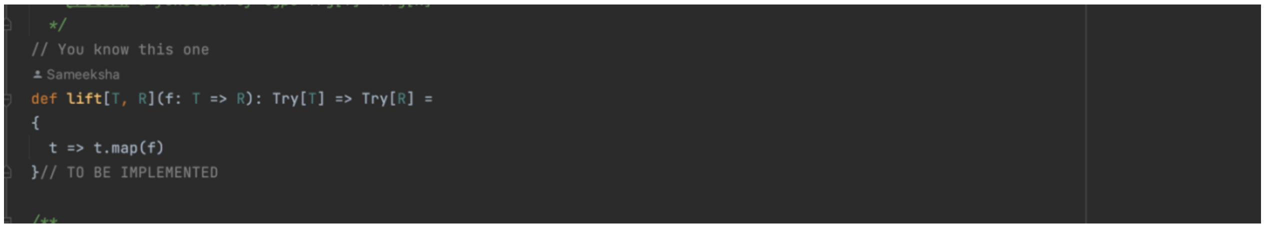
```
  * @return a function of type (Try[T1],Try[T2])=>Try[R]
  */
// Think Simple, Elegant, Obvious
Sameeksha
def lift2[T1, T2, R](f: (T1, T2) => R): (Try[T1], Try[T2]) => Try[R] = map2(_, _)(f) // TO BE IMPLEMENTED

/**
  * 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
```



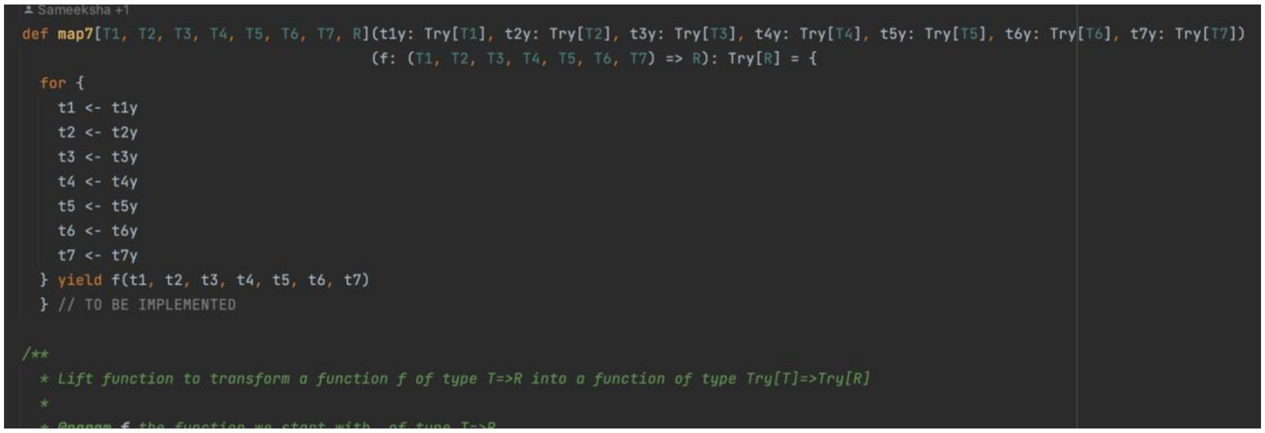
```
// If you can do lift3, you can do lift7
Sameeksha +1
def lift7[T1, T2, T3, T4, T5, T6, T7, R](f: (T1, T2, T3, T4, T5, T6, T7) => R):
(Try[T1], Try[T2], Try[T3], Try[T4], Try[T5], Try[T6], Try[T7]) => Try[R] =
{
  (t1y, t2y, t3y, t4y, t5y, t6y, t7y) => map7(t1y, t2y, t3y, t4y, t5y, t6y, t7y)(f)
}// TO BE IMPLEMENTED
```



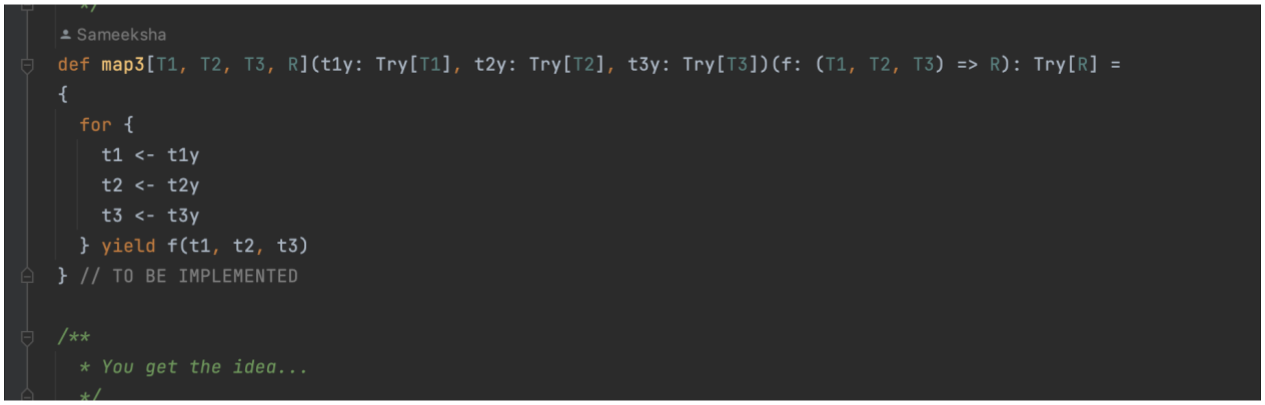
```
// If you can do lift2, you can do lift3
Sameeksha
def lift3[T1, T2, T3, R](f: (T1, T2, T3) => R): (Try[T1], Try[T2], Try[T3]) => Try[R] =
{
  (t1y, t2y, t3y) => map3(t1y, t2y, t3y)(f)
}// TO BE IMPLEMENTED

/**
  * 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],Tr
```



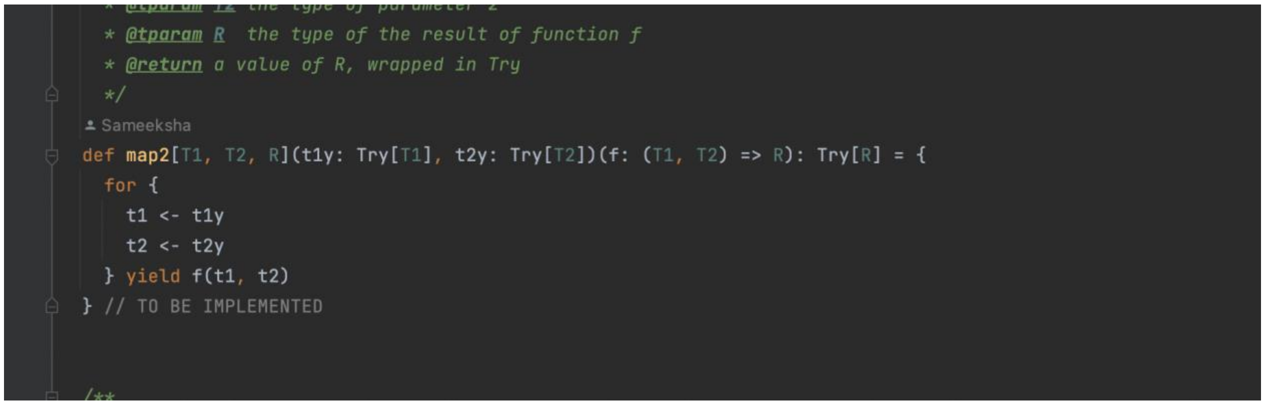
```
  */
// You know this one
Sameeksha
def lift[T, R](f: T => R): Try[T] => Try[R] =
{
  t => t.map(f)
}// TO BE IMPLEMENTED

/**
```



```
Sameeksha +1
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
  } yield f(t1, t2, t3, t4, t5, t6, t7)
  } // TO BE IMPLEMENTED

/**
  * 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
```

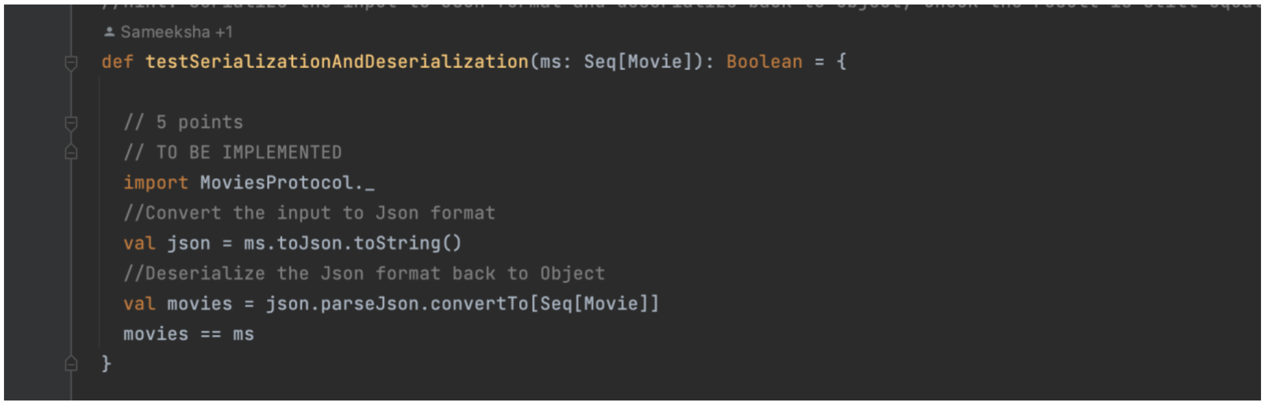


```scala
  */
Sameeksha
def map3[T1, T2, T3, R](t1y: Try[T1], t2y: Try[T2], t3y: Try[T3])(f: (T1, T2, T3) => R): Try[R] =
{
  for {
    t1 <- t1y
    t2 <- t2y
    t3 <- t3y
  } yield f(t1, t2, t3)
} // TO BE IMPLEMENTED

/**
  * You get the idea...
  */
```

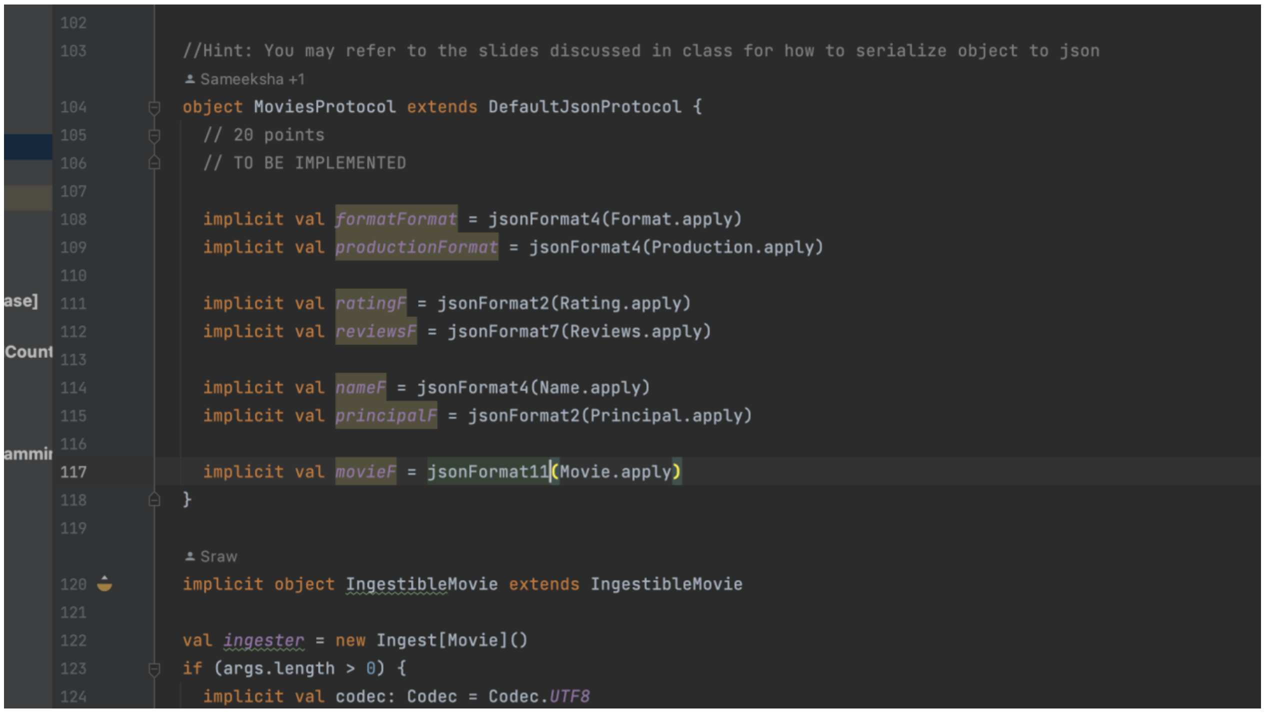


```scala
  * @tparam T2 the type of parameter 2
  * @tparam R  the type of the result of function f
  * @return a value of R, wrapped in Try
  */
Sameeksha
def map2[T1, T2, R](t1y: Try[T1], t2y: Try[T2])(f: (T1, T2) => R): Try[R] = {
  for {
    t1 <- t1y
    t2 <- t2y
  } yield f(t1, t2)
} // TO BE IMPLEMENTED


/**
```

2. Movie



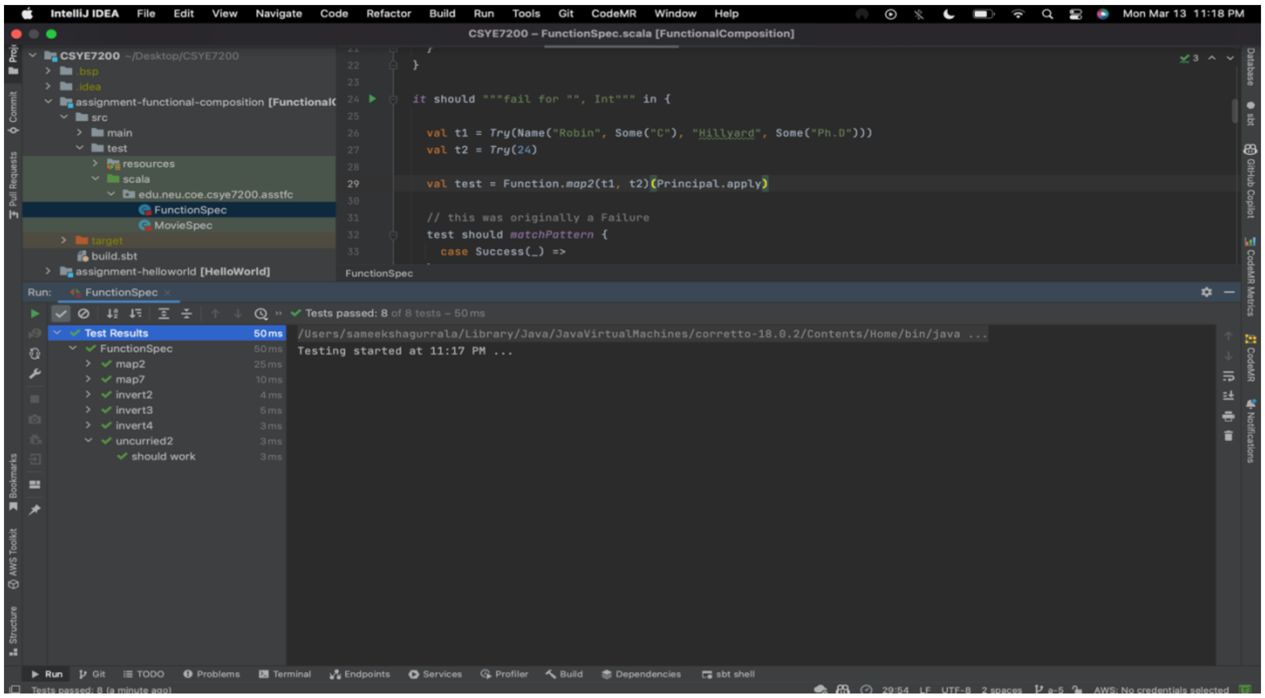
```scala
Sameeksha +1
def testSerializationAndDeserialization(ms: Seq[Movie]): Boolean = {

  // 5 points
  // TO BE IMPLEMENTED
  import MoviesProtocol._
  //Convert the input to Json format
  val json = ms.toJson.toString()
  //Deserialize the Json format back to Object
  val movies = json.parseJson.convertTo[Seq[Movie]]
  movies == ms
}
```



```scala
//Hint: You may refer to the slides discussed in class for how to serialize object to json
Sameeksha +1
object MoviesProtocol extends DefaultJsonProtocol {
  // 20 points
  // TO BE IMPLEMENTED

  implicit val formatFormat = jsonFormat4(Format.apply)
  implicit val productionFormat = jsonFormat4(Production.apply)

  implicit val ratingF = jsonFormat2(Rating.apply)
  implicit val reviewsF = jsonFormat7(Reviews.apply)

  implicit val nameF = jsonFormat4(Name.apply)
  implicit val principalF = jsonFormat2(Principal.apply)

  implicit val movieF = jsonFormat11(Movie.apply)
}

Sraw
implicit object IngestibleMovie extends IngestibleMovie

val ingester = new Ingest[Movie]()
if (args.length > 0) {
  implicit val codec: Codec = Codec.UTF8
```

-Unit tests

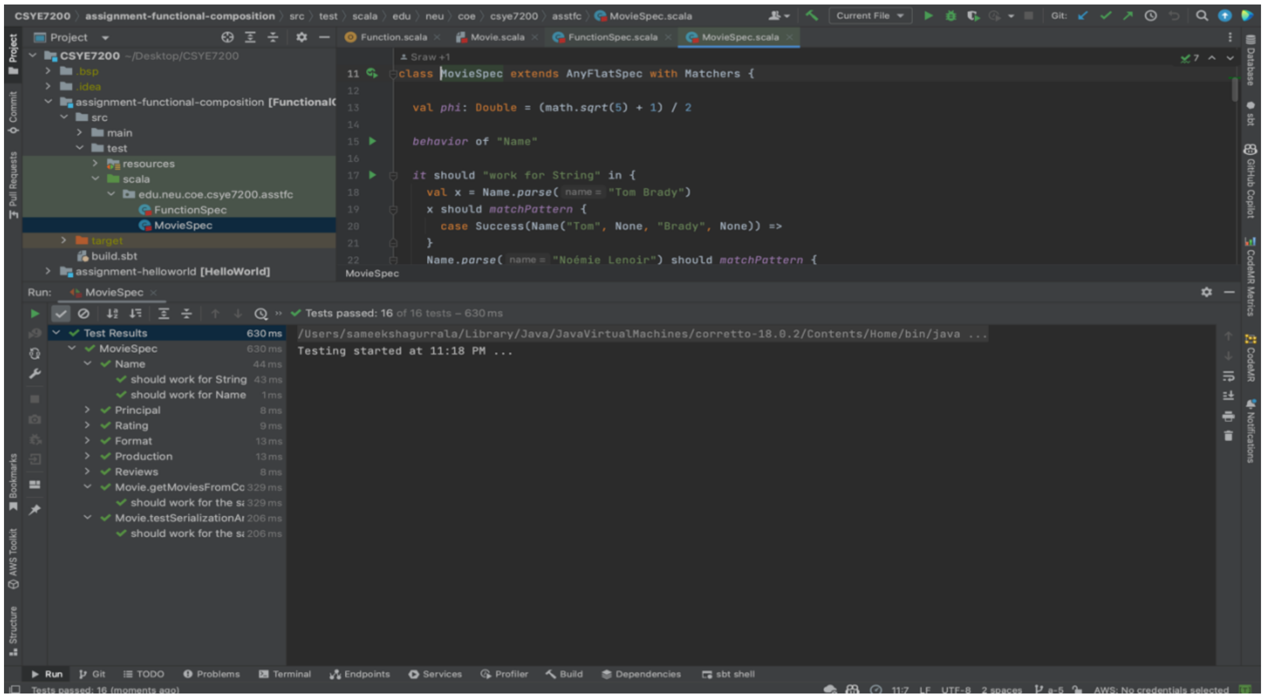
FunctionSpec



```scala
  }

  it should """fail for "", Int""" in {

    val t1 = Try(Name("Robin", Some("C"), "Hillyard", Some("Ph.D")))
    val t2 = Try(24)

    val test = Function.map2(t1, t2)(Principal.apply)

    // this was originally a Failure
    test should matchPattern {
      case Success(_) =>
```

```
Tests passed: 8 of 8 tests – 50 ms

Test Results          50 ms
  FunctionSpec        50 ms
    map2              25 ms
    map7              10 ms
    invert2            4 ms
    invert3            5 ms
    invert4            3 ms
    uncurried2         3 ms
      should work      3 ms

/Users/sameekshagurrala/Library/Java/JavaVirtualMachines/corretto-18.0.2/Contents/Home/bin/java ...
Testing started at 11:17 PM ...
```

MovieSpec



```scala
class MovieSpec extends AnyFlatSpec with Matchers {

  val phi: Double = (math.sqrt(5) + 1) / 2

  behavior of "Name"

  it should "work for String" in {
    val x = Name.parse( name = "Tom Brady")
    x should matchPattern {
      case Success(Name("Tom", None, "Brady", None)) =>
    }
    Name.parse( name = "Noémie Lenoir") should matchPattern {
```

```
Tests passed: 16 of 16 tests – 630 ms

Test Results                          630 ms
  MovieSpec                           630 ms
    Name                               44 ms
      should work for String           43 ms
      should work for Name              1 ms
    Principal                           8 ms
    Rating                              9 ms
    Format                             13 ms
    Production                         13 ms
    Reviews                             8 ms
    Movie.getMoviesFromCc             329 ms
      should work for the sı          329 ms
    Movie.testSerializationAı         206 ms
      should work for the sı          206 ms

/Users/sameekshagurrala/Library/Java/JavaVirtualMachines/corretto-18.0.2/Contents/Home/bin/java ...
Testing started at 11:18 PM ...
```