Hrishikesh Sanjay Pawar – SEC01 (NUID 002707307)

Big Data System Engineering with Scala
Spring 2023
Assignment No. #4

-List of Tasks Implemented

1. Implemented flatMap() to map a RandomState into another RandomState.

2. Implemented toStream() to return a stream of T values

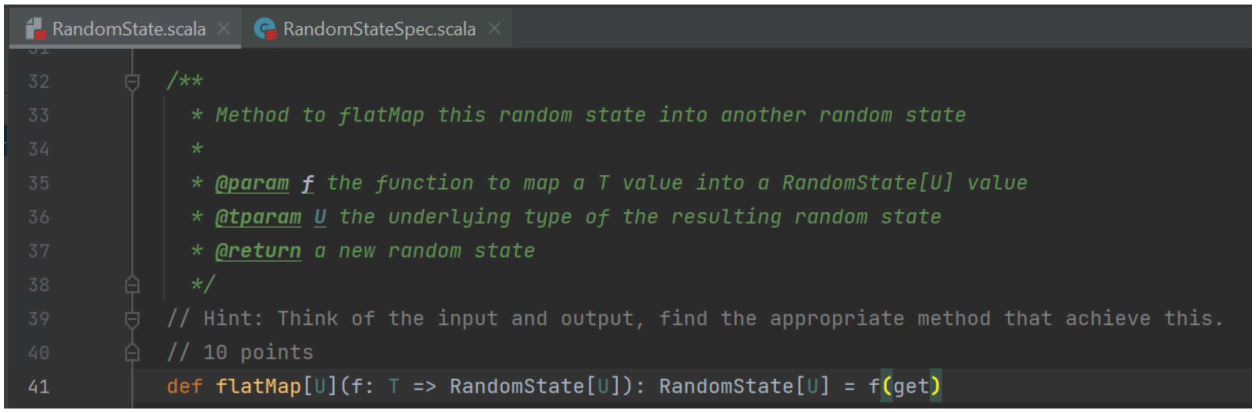
3. Implemented next() method to generate next RandomState

4. Implemented get() method

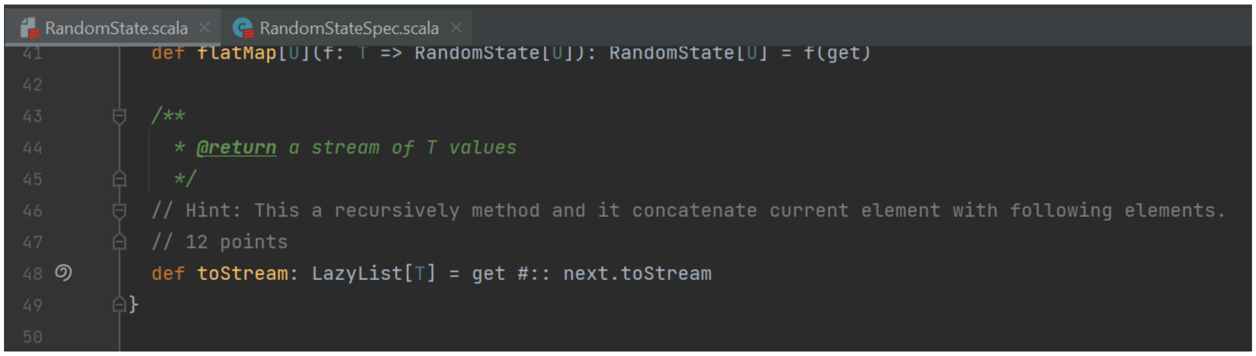
5. Implemented map() method using function composition

6. Implemented longToDouble

-Code



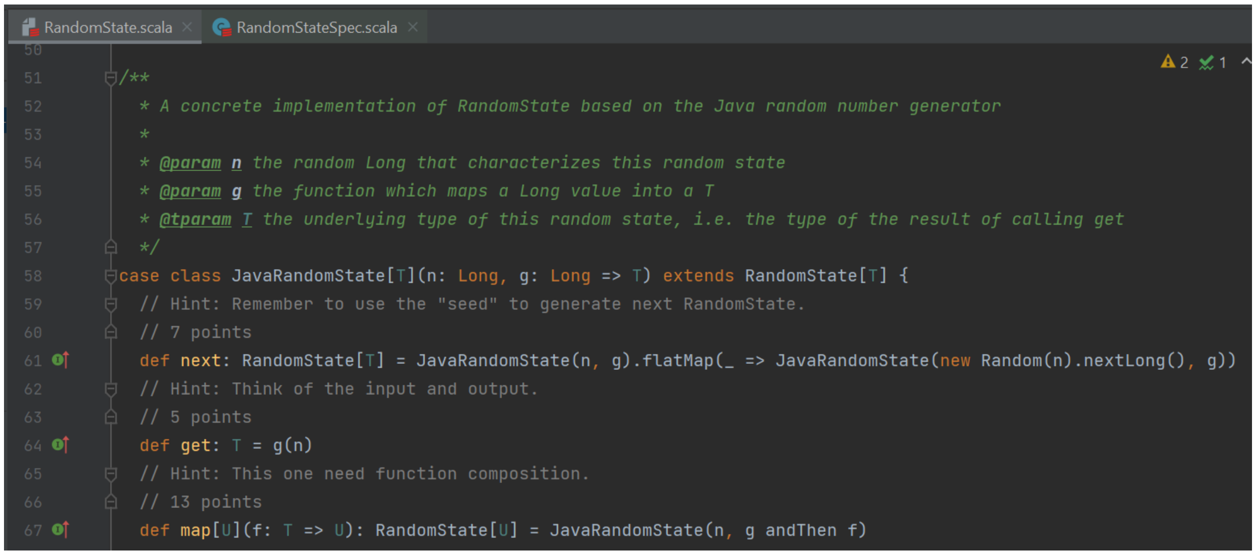
```
  /**
    * Method to flatMap this random state into another random state
    *
    * @param f the function to map a T value into a RandomState[U] value
    * @tparam U the underlying type of the resulting random state
    * @return a new random state
    */
  // Hint: Think of the input and output, find the appropriate method that achieve this.
  // 10 points
  def flatMap[U](f: T => RandomState[U]): RandomState[U] = f(get)
```



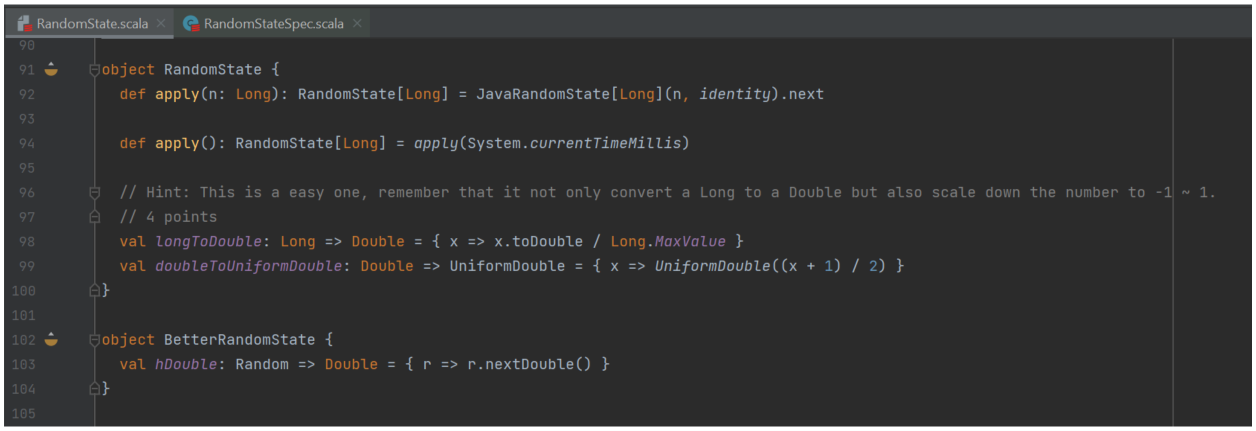
```
  def flatMap[U](f: T => RandomState[U]): RandomState[U] = f(get)

  /**
    * @return a stream of T values
    */
  // Hint: This a recursively method and it concatenate current element with following elements.
  // 12 points
  def toStream: LazyList[T] = get #:: next.toStream
}

```


RandomState.scala | RandomStateSpec.scala

```scala
/**
  * A concrete implementation of RandomState based on the Java random number generator
  *
  * @param n the random Long that characterizes this random state
  * @param g the function which maps a Long value into a T
  * @tparam T the underlying type of this random state, i.e. the type of the result of calling get
  */
case class JavaRandomState[T](n: Long, g: Long => T) extends RandomState[T] {
  // Hint: Remember to use the "seed" to generate next RandomState.
  // 7 points
  def next: RandomState[T] = JavaRandomState(n, g).flatMap(_ => JavaRandomState(new Random(n).nextLong(), g))
  // Hint: Think of the input and output.
  // 5 points
  def get: T = g(n)
  // Hint: This one need function composition.
  // 13 points
  def map[U](f: T => U): RandomState[U] = JavaRandomState(n, g andThen f)
```


RandomState.scala | RandomStateSpec.scala

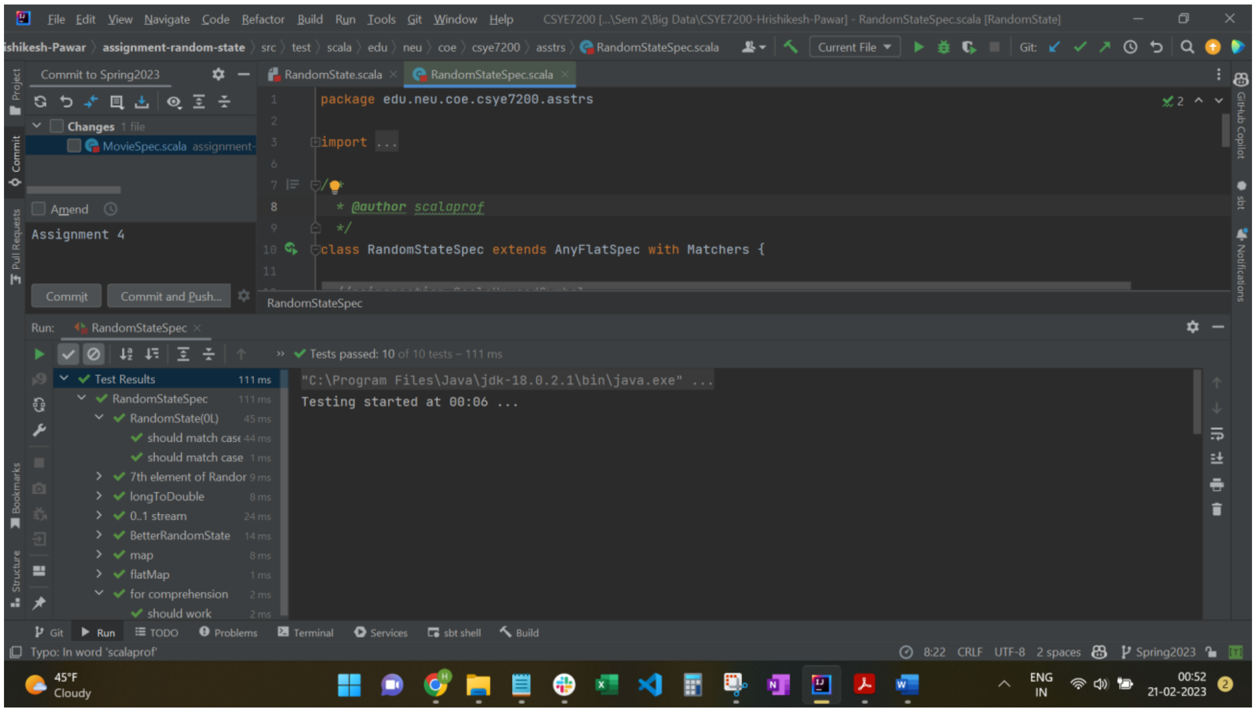
```scala
object RandomState {
  def apply(n: Long): RandomState[Long] = JavaRandomState[Long](n, identity).next

  def apply(): RandomState[Long] = apply(System.currentTimeMillis)

  // Hint: This is a easy one, remember that it not only convert a Long to a Double but also scale down the number to -1 ~ 1.
  // 4 points
  val longToDouble: Long => Double = { x => x.toDouble / Long.MaxValue }
  val doubleToUniformDouble: Double => UniformDouble = { x => UniformDouble((x + 1) / 2) }
}

object BetterRandomState {
  val hDouble: Random => Double = { r => r.nextDouble() }
}
```

-Unit tests



```
package edu.neu.coe.csye7200.asstrs

import ...

/*
 * @author scalaprof
 */
class RandomStateSpec extends AnyFlatSpec with Matchers {
```

```
Tests passed: 10 of 10 tests – 111 ms

Test Results                     111 ms
  RandomStateSpec                111 ms
    RandomState(0L)               45 ms
      should match case           44 ms
      should match case            1 ms
    7th element of Randor          9 ms
    longToDouble                   8 ms
    0..1 stream                   24 ms
    BetterRandomState             14 ms
    map                            8 ms
    flatMap                        1 ms
    for comprehension              2 ms
      should work                  2 ms
```

```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Testing started at 00:06 ...
```

- Result

Module for RandomState was implemented successfully.