Anirudha Joshi – SEC01 (NUID 002991365)

Big Data System Engineering with Scala
Spring 2023
Assignment No. 4 (Random State Assignment)

GitHub - https://github.com/anirudhajoshi2808/Anirudha_Joshi_CSYE7200

- Tasks:

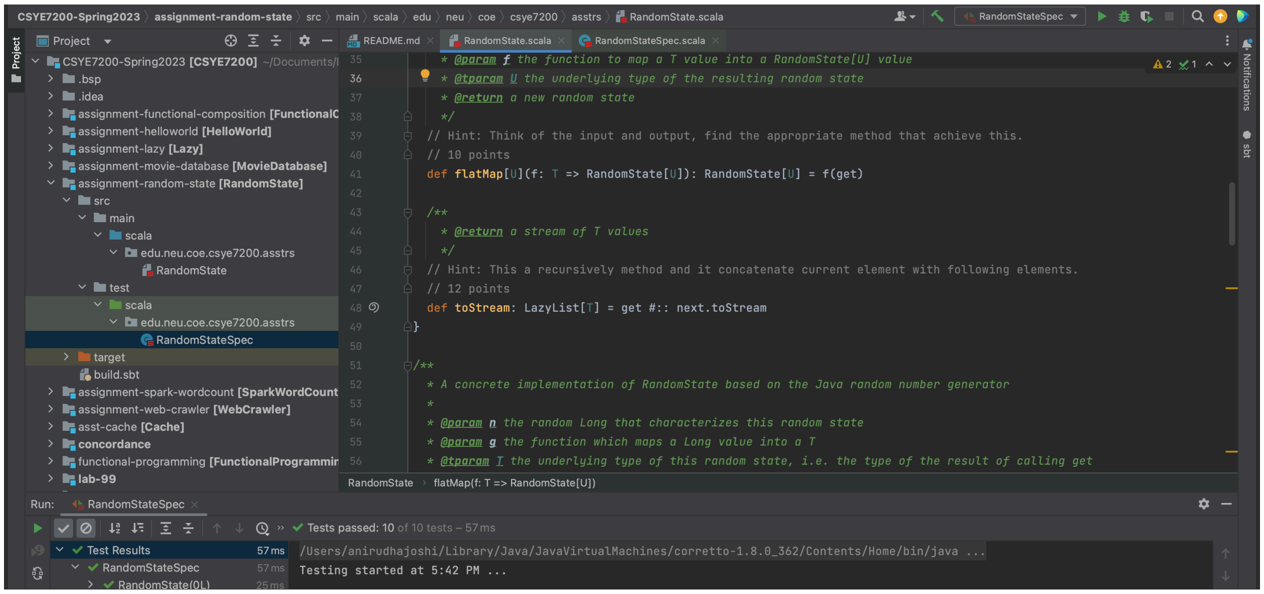
We need to create a trait called *RandomState* which will have two obvious methods: *next* and *get*. Of course, we don't really know what the type of the result of *get* will be, so let's make it parametric, thus: *RandomState[T]*.

But once we have a *RandomState[T]*, we will want to be able to map it into a *RandomState[U]* so we'll need to implement *map*. While we're at it, we might as well implement *flatMap* too. Technically, this will mean that it's a "monad" but we haven't talked about those yet -- but they are important.

There's one other convenience method that we should probably implement and that is *toStream* which will return a *LazyList[T]*. As usual, I have provided the basic framework and a specification for your work: *src/main/scala/edu/neu/coe/csye7200/asstrs/RandomState.scala* and the corresponding *RandomStateSpec* in the *test* directory. All you have to do is to implement the 6 *TO BE IMPLEMENTED* and run the tests. When it's all green, you're done. You can get these from the class repo (see *Course Material/Resources/Class Repository*), the module name for this assignment is *assignment-random-state*.

- Code: RandomState.scala

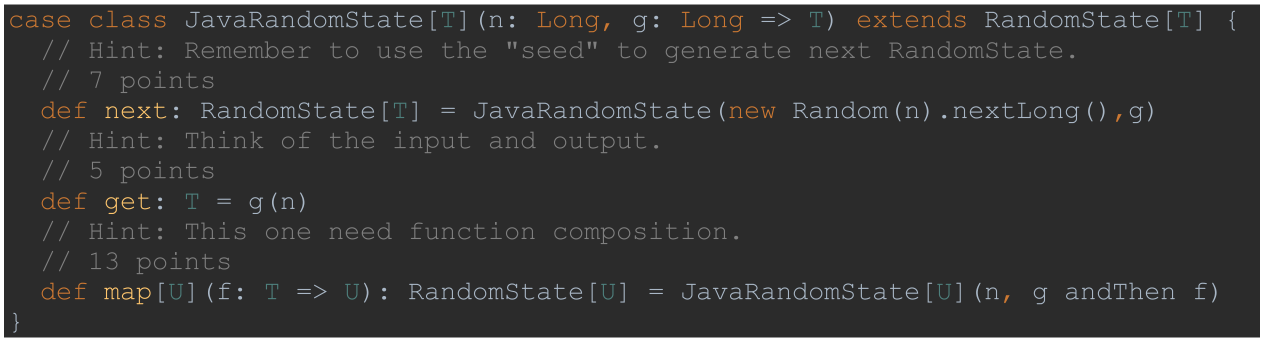
```
// 10 points
def flatMap[U](f: T => RandomState[U]): RandomState[U] = f(get)
// 12 points
def toStream: LazyList[T] = get #:: next.toStream
```


CSYE7200-Spring2023 › assignment-random-state › src › main › scala › edu › neu › coe › csye7200 › asstrs › RandomState.scala
RandomStateSpec
Project
README.md
RandomState.scala
RandomStateSpec.scala
CSYE7200-Spring2023 [CSYE7200] ~/Documents/
.bsp
.idea
assignment-functional-composition [FunctionalC
assignment-helloworld [HelloWorld]
assignment-lazy [Lazy]
assignment-movie-database [MovieDatabase]
assignment-random-state [RandomState]
src
main
scala
edu.neu.coe.csye7200.asstrs
RandomState
test
scala
edu.neu.coe.csye7200.asstrs
RandomStateSpec
target
build.sbt
assignment-spark-wordcount [SparkWordCount
assignment-web-crawler [WebCrawler]
asst-cache [Cache]
concordance
functional-programming [FunctionalProgrammin
lab-99
RandomState › flatMap(f: T => RandomState[U])
Run: RandomStateSpec
Test Results 57 ms
RandomStateSpec 57 ms
RandomState(0L) 25 ms

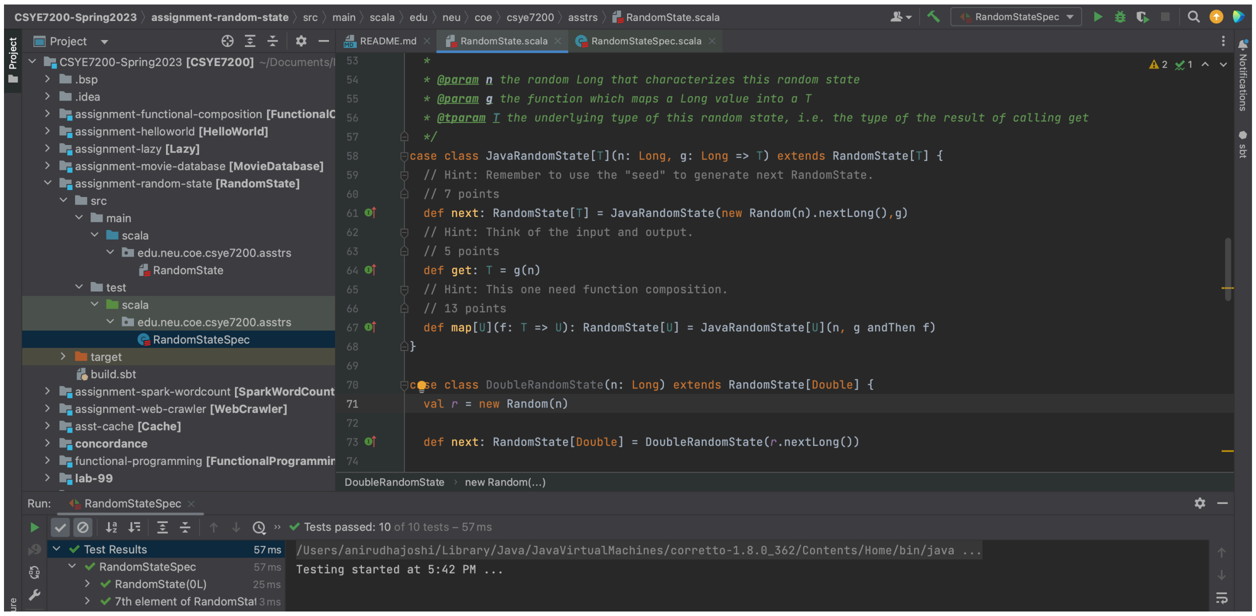
```
   * @param f the function to map a T value into a RandomState[U] value
   * @tparam U the underlying type of the resulting random state
   * @return a new random state
   */
  // Hint: Think of the input and output, find the appropriate method that achieve this.
  // 10 points
  def flatMap[U](f: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

  /**
   * @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
}

/**
 * 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
```

Tests passed: 10 of 10 tests – 57 ms

```
/Users/anirudhajoshi/Library/Java/JavaVirtualMachines/corretto-1.8.0_362/Contents/Home/bin/java ...
Testing started at 5:42 PM ...
```



```
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(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[U](n, g andThen f)
}
```

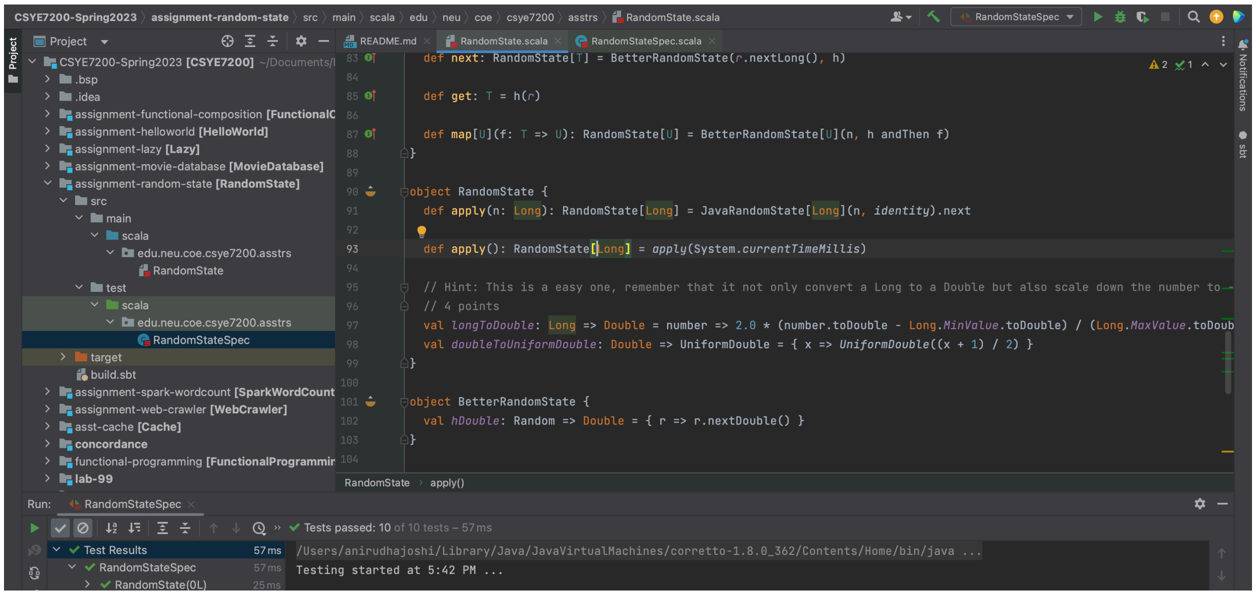

CSYE7200-Spring2023 › assignment-random-state › src › main › scala › edu › neu › coe › csye7200 › asstrs › RandomState.scala
RandomStateSpec
Project
README.md
RandomState.scala
RandomStateSpec.scala
CSYE7200-Spring2023 [CSYE7200] ~/Documents/
.bsp
.idea
assignment-functional-composition [FunctionalC
assignment-helloworld [HelloWorld]
assignment-lazy [Lazy]
assignment-movie-database [MovieDatabase]
assignment-random-state [RandomState]
src
main
scala
edu.neu.coe.csye7200.asstrs
RandomState
test
scala
edu.neu.coe.csye7200.asstrs
RandomStateSpec
target
build.sbt
assignment-spark-wordcount [SparkWordCount
assignment-web-crawler [WebCrawler]
asst-cache [Cache]
concordance
functional-programming [FunctionalProgrammin
lab-99
DoubleRandomState › new Random(...)
Run: RandomStateSpec
Test Results 57 ms
RandomStateSpec 57 ms
RandomState(0L) 25 ms
7th element of RandomStat 3 ms

```
 *
 * @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(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[U](n, g andThen f)
}

case class DoubleRandomState(n: Long) extends RandomState[Double] {
  val r = new Random(n)

  def next: RandomState[Double] = DoubleRandomState(r.nextLong())

```

Tests passed: 10 of 10 tests – 57 ms

```
/Users/anirudhajoshi/Library/Java/JavaVirtualMachines/corretto-1.8.0_362/Contents/Home/bin/java ...
Testing started at 5:42 PM ...
```

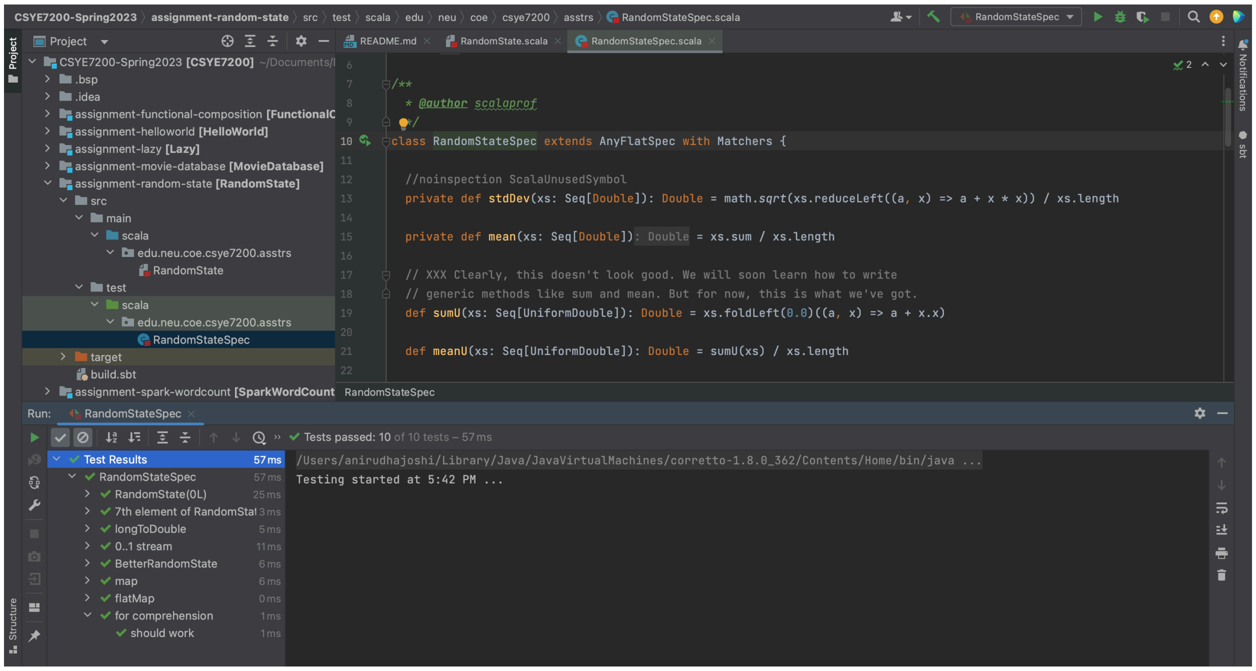
```scala
// Hint: This is a easy one, remember that it not only convert a Long to a
Double but also scale down the number to -1 ~ 1.
// 4 points
val longToDouble: Long => Double = number => 2.0 * (number.toDouble -
Long.MinValue.toDouble) / (Long.MaxValue.toDouble - Long.MinValue.toDouble) -
1.0
```



```scala
  def next: RandomState[T] = BetterRandomState(r.nextLong(), h)

  def get: T = h(r)

  def map[U](f: T => U): RandomState[U] = BetterRandomState[U](n, h andThen f)
}

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
  // 4 points
  val longToDouble: Long => Double = number => 2.0 * (number.toDouble - Long.MinValue.toDouble) / (Long.MaxValue.toDoub
  val doubleToUniformDouble: Double => UniformDouble = { x => UniformDouble((x + 1) / 2) }
}

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

```
Tests passed: 10 of 10 tests – 57 ms
/Users/anirudhajoshi/Library/Java/JavaVirtualMachines/corretto-1.8.0_362/Contents/Home/bin/java ...
Testing started at 5:42 PM ...
```

- Unit tests (RandomStateSpec.scala)



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

  //noinspection ScalaUnusedSymbol
  private def stdDev(xs: Seq[Double]): Double = math.sqrt(xs.reduceLeft((a, x) => a + x * x)) / xs.length

  private def mean(xs: Seq[Double]): Double = xs.sum / xs.length

  // XXX Clearly, this doesn't look good. We will soon learn how to write
  // generic methods like sum and mean. But for now, this is what we've got.
  def sumU(xs: Seq[UniformDouble]): Double = xs.foldLeft(0.0)((a, x) => a + x.x)

  def meanU(xs: Seq[UniformDouble]): Double = sumU(xs) / xs.length
```

```
Tests passed: 10 of 10 tests – 57 ms
Test Results                         57 ms
  RandomStateSpec                    57 ms
    RandomState(0L)                  25 ms
    7th element of RandomStat         3 ms
    longToDouble                      5 ms
    0..1 stream                      11 ms
    BetterRandomState                 6 ms
    map                               6 ms
    flatMap                           0 ms
    for comprehension                 1 ms
      should work                     1 ms
/Users/anirudhajoshi/Library/Java/JavaVirtualMachines/corretto-1.8.0_362/Contents/Home/bin/java ...
Testing started at 5:42 PM ...
```