Harshil Shah – SEC01 (NUID 002780887)

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

- GitHub Repo URL - https://github.com/harshilshahneu/CSYE7200-Harshil-Shah

---

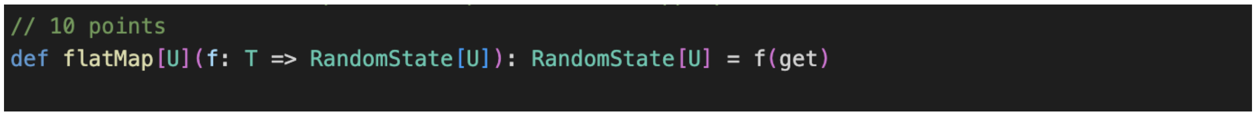
- List of Tasks Implemented

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].

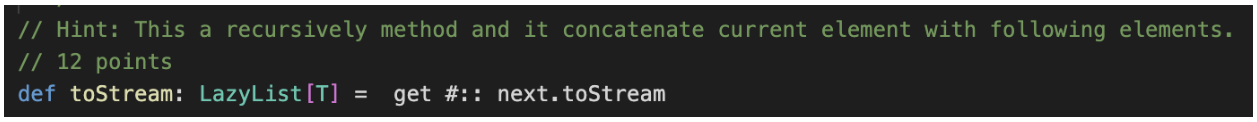
But once we have a RandomState[T], we will want to be able to map it into a RandomState[U] so we'll need to implement map. While we're at it, we might as well implement flatMap too. Technically, this will mean that it's a "monad" but we haven't talked about those yet -- but they are important.

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.

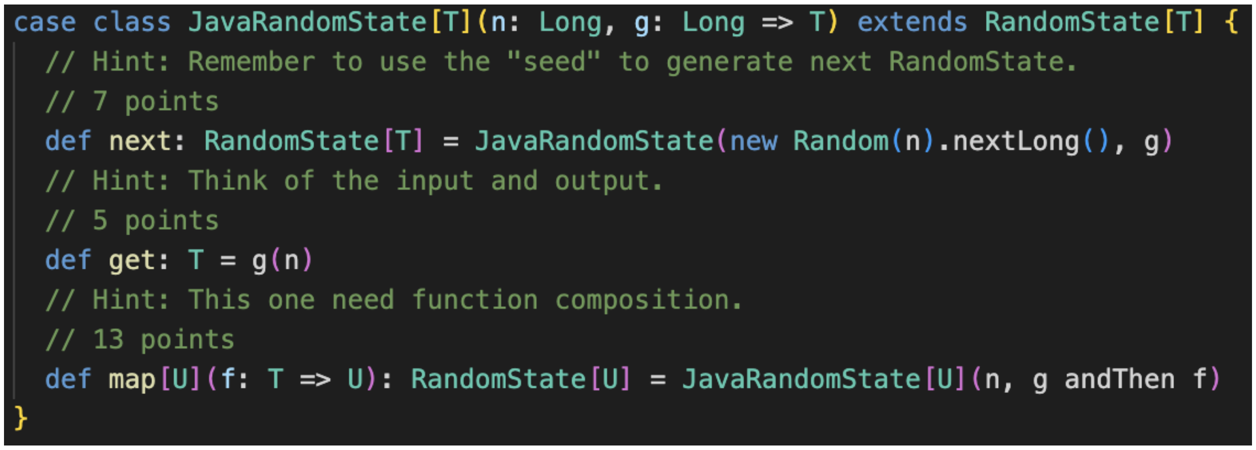
- Code



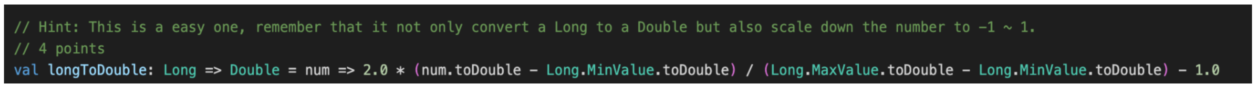
```scala
// 10 points
def flatMap[U](f: T => RandomState[U]): RandomState[U] = f(get)
```



```scala
// Hint: This a recursively method and it concatenate current element with following elements.
// 12 points
def toStream: LazyList[T] =  get #:: next.toStream
```

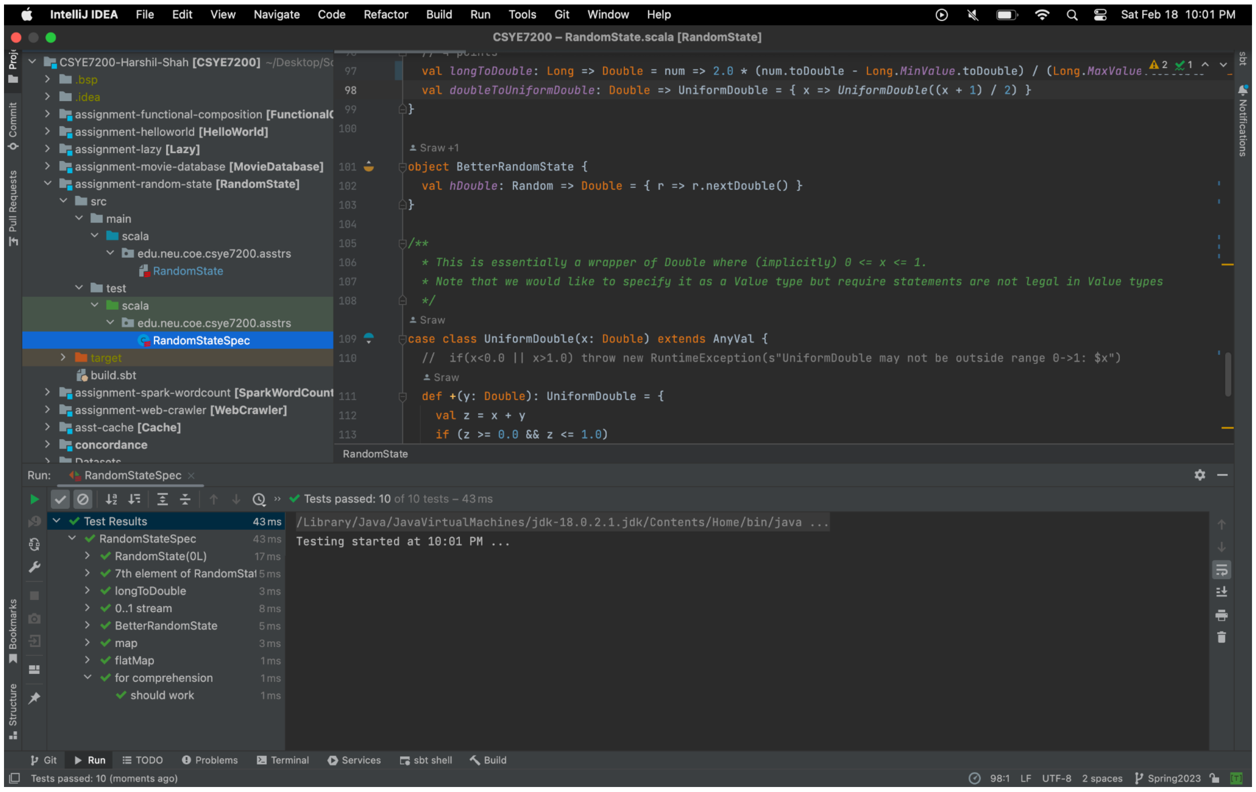


```scala
case class JavaRandomState[T](n: Long, g: Long => T) extends RandomState[T] {
  // Hint: Remember to use the "seed" to generate next RandomState.
  // 7 points
  def next: RandomState[T] = JavaRandomState(new Random(n).nextLong(), g)
  // Hint: Think of the input and output.
  // 5 points
  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)
}
```



```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 = num => 2.0 * (num.toDouble - Long.MinValue.toDouble) / (Long.MaxValue.toDouble - Long.MinValue.toDouble) - 1.0
```

- Unit tests



```scala
  val longToDouble: Long => Double = num => 2.0 * (num.toDouble - Long.MinValue.toDouble) / (Long.MaxValue.
  val doubleToUniformDouble: Double => UniformDouble = { x => UniformDouble((x + 1) / 2) }
}

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

/**
 * This is essentially a wrapper of Double where (implicitly) 0 <= x <= 1.
 * Note that we would like to specify it as a Value type but require statements are not legal in Value types
 */
case class UniformDouble(x: Double) extends AnyVal {
  //  if(x<0.0 || x>1.0) throw new RuntimeException(s"UniformDouble may not be outside range 0->1: $x")
  def +(y: Double): UniformDouble = {
    val z = x + y
    if (z >= 0.0 && z <= 1.0)
```

Tests passed: 10 of 10 tests – 43 ms

```
Test Results                          43 ms
  RandomStateSpec                     43 ms
    RandomState(0L)                   17 ms
    7th element of RandomStat         5 ms
    longToDouble                      3 ms
    0..1 stream                       8 ms
    BetterRandomState                 5 ms
    map                               3 ms
    flatMap                           1 ms
    for comprehension                 1 ms
      should work                     1 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
Testing started at 10:01 PM ...
```

Tests passed: 10 (moments ago)
