Bowen Jiang (NUID: 001582174)

CSYE 7200

Big-Data Sys Engr Using Scala

Sprint 2021

Assignment 2

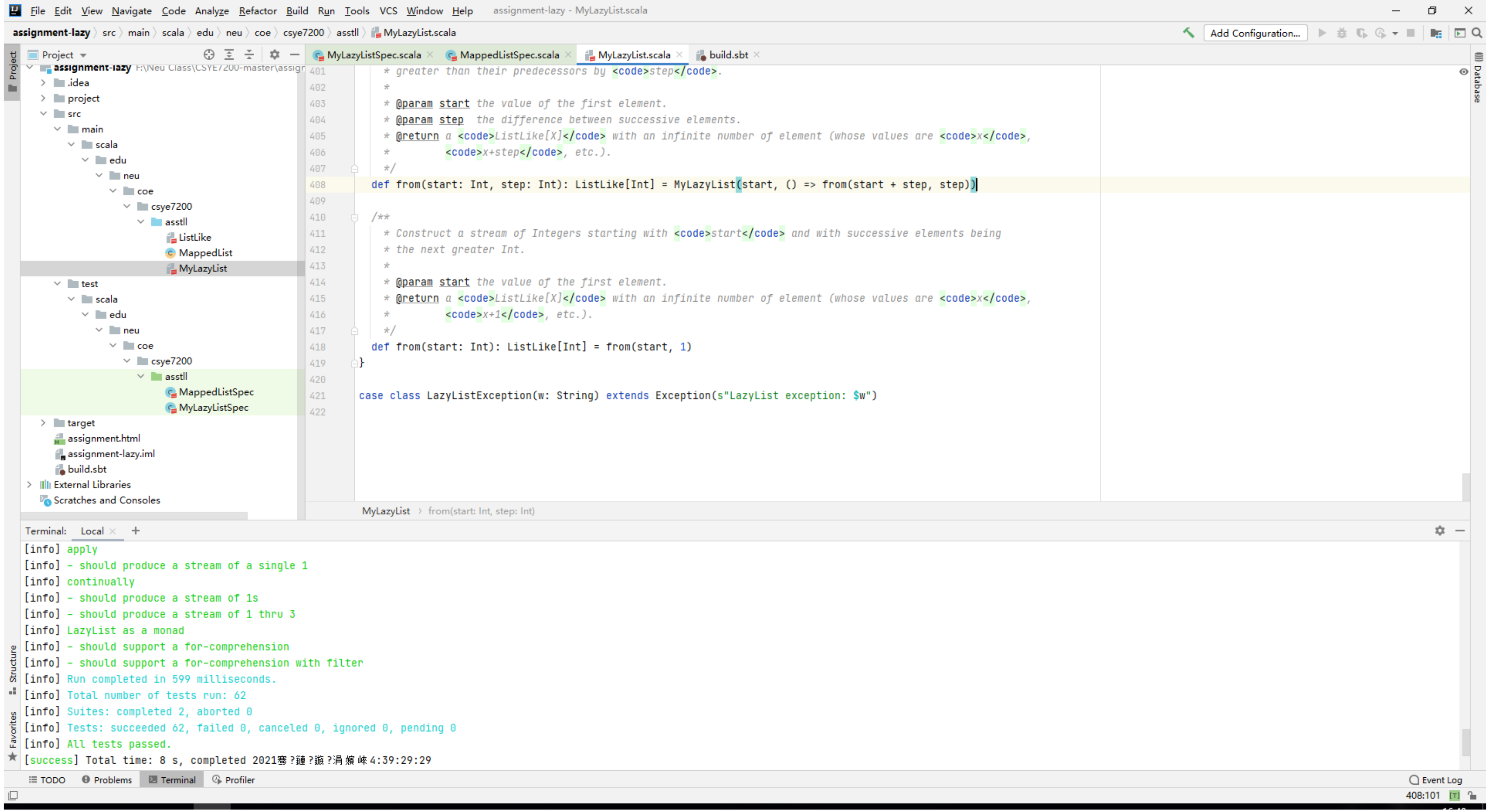
Task:

Assignment 2 is designed to test your understanding of the second week of CSYE7200 lectures, in particular, Scala's mechanism for dealing with lazy values. Furthermore, working through the assignment, you will understand that features like *LazyList* in Scala are not "magic." We can create our own, simply using a function for the lazily-evaluated tail (*MyLazyList*).

You can find the code under the *assignment-lazy* directory in the REPO. Make sure you do a pull first.

You are to submit evidence of all unit tests (in *MyLazyListSpec*) passing (use a screenshot). In order to get the tests to succeed, you must implement the *from* method in the companion object of *MyLazyList*. Replace the *??? //...* with your own working code. There are plenty of other examples of similar functionality in the *MyLazyList* module. Submit, along with your unit test screenshots, the expression that you used to implement the *from* method.

Unit Test Screenshot



```scala
   * greater than their predecessors by <code>step</code>.
   *
   * @param start the value of the first element.
   * @param step  the difference between successive elements.
   * @return a <code>ListLike[X]</code> with an infinite number of element (whose values are <code>x</code>,
   *         <code>x+step</code>, etc.).
   */
  def from(start: Int, step: Int): ListLike[Int] = MyLazyList(start, () => from(start + step, step))

  /**
   * Construct a stream of Integers starting with <code>start</code> and with successive elements being
   * the next greater Int.
   *
   * @param start the value of the first element.
   * @return a <code>ListLike[X]</code> with an infinite number of element (whose values are <code>x</code>,
   *         <code>x+1</code>, etc.).
   */
  def from(start: Int): ListLike[Int] = from(start, 1)
}

case class LazyListException(w: String) extends Exception(s"LazyList exception: $w")
```

```
[info] apply
[info] - should produce a stream of a single 1
[info] continually
[info] - should produce a stream of 1s
[info] - should produce a stream of 1 thru 3
[info] LazyList as a monad
[info] - should support a for-comprehension
[info] - should support a for-comprehension with filter
[info] Run completed in 599 milliseconds.
[info] Total number of tests run: 62
[info] Suites: completed 2, aborted 0
[info] Tests: succeeded 62, failed 0, canceled 0, ignored 0, pending 0
[info] All tests passed.
[success] Total time: 8 s, completed 2021骞?鏈?鍦?涓嬪崍4:39:29:29
```

Questions

1. (a) what is the chief way by which *MyLazyList* differs from *LazyList* (the built-in Scala class that does the same thing). Don't mention the methods that *MyLazyList* does or doesn't implement--I want to know what is the *structural* difference.
   (b) Why do you think there is this difference?
   Answer:
   a: MyLazyList class has constructor and MylazyList is a case class, but LazyList is not.
   MylazyList class is same as the cons class in Stream, the Stream class is a wrap class of cons.

   b:The constructor can create an instance and covered in object with apply method, LazyList can do the same work.
   LazyList is build with cons, it can find tail by using less memory. But for MylazyList we have to evaluate each one util we reach the last one to reach the tail.
   The case class can optimize MyLazyList if the pattern match.

2. Explain what the following code actually does and why is it needed?

```
        def tail = lazyTail()
```

Answer:
It help to make MyLazyList class to a lazy evaluate. It's a lazylist, only the head be evaluated. It defined the tail called by the function. It will evaluate and return a ListLike object contains the rest of the list behind the head.

3. List all of the recursive calls that you can find in *MyLazyList* (give line numbers).
   98, 116, 130, 361, 383, 408

4. List all of the mutable variables and mutable collections that you can find in *MyLazyList* (give line numbers).
   42, 69, 388

5. What is the purpose of the *zip* method?

Combine two different list together. The purpose of the zip method can process two list at the same time, you can zip another list into the current one.

6. Why is there no *length* (or *size*) method for *MyLazyList*?

MyLazyList is build with an actual list, we need to evaluate each element in the list in order to get the length of the list.

If the list is infinite, the stack will be overflow. MyLazyList can not have length method.