Xinzhuo Liu – SEC01 (NUID 2197134)

# Big Data System Engineering with Scala
# Spring 2022
# Assignment No. 2

**Task**

- **implement the from method in the companion object of MyLazyList**

- **Submit, along with your unit test screenshots, the expression that you used to implement the from method**

- **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?**

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

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

- **3. List all of the recursive calls that you can find in MyLazyList (give line numbers).**

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

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

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

**Solution/ Unit test (screenshot)**

1.a The chief way of difference between MyLazyList and LazyList is that, in MyLazyList, the functions donot invoke parameter defined by “lazy”, but use LazyList which is defined as a function that returns ListLike[X] as the tail. And from that, I think humbly that MyLazyList invokes a class to have the same function like “lazy” does as we can complement the elements whenever we need them.

1.b By comparing the source code of LazyList to that of MyLazyList, I found that while invoking the tail, MyLazyList just uses a “lazyTail()” to represents the tail when it is “state.tail” in LazyList. And after locating where the “state” is, I notice that in “state”, tail is defined by “lazy”. Then I realized that might be the difference.

2. The “lazyTail()” is defined as a function which, when invoked, yields the tail of the stream. So that the poin. This “lazyTail()” function serves as “lazy” that whenever we

need the tail, then the tail can be created immediately. In another word, we can get things lazily initialized by using “lazyTail()”.

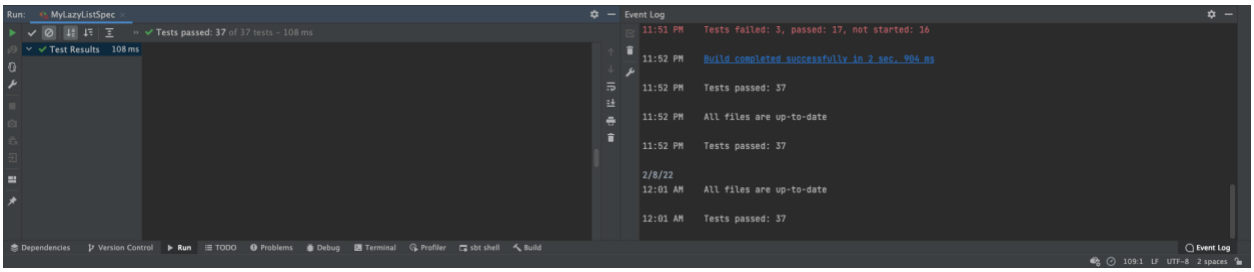
3. recursive calls: 43/ 69/ 82/ 98/ 116/ 131/ 361/ 372/ 383/ 388/ 408

4. mutable variables and mutable collections: none

5. zip method: In order to merge two MyLazyList into a new tuple that has a relationship between elements of each from beginning to end.

6. Because MyLazyList serves as LinkedList in Java, and the basic reason is that MyLazyList is a recursive structure and maintain the size method will add memory overhead to all lists and add slight time overhead creation of all lists.

```
/**
  * Construct a stream of Integers starting with <code>start</code> and with successive elements being
  * 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))
```




**Project Source (github link to specific module relate to assignment)**

Link: https://github.com/ZhongLBuL/Scala/tree/main/Assign2