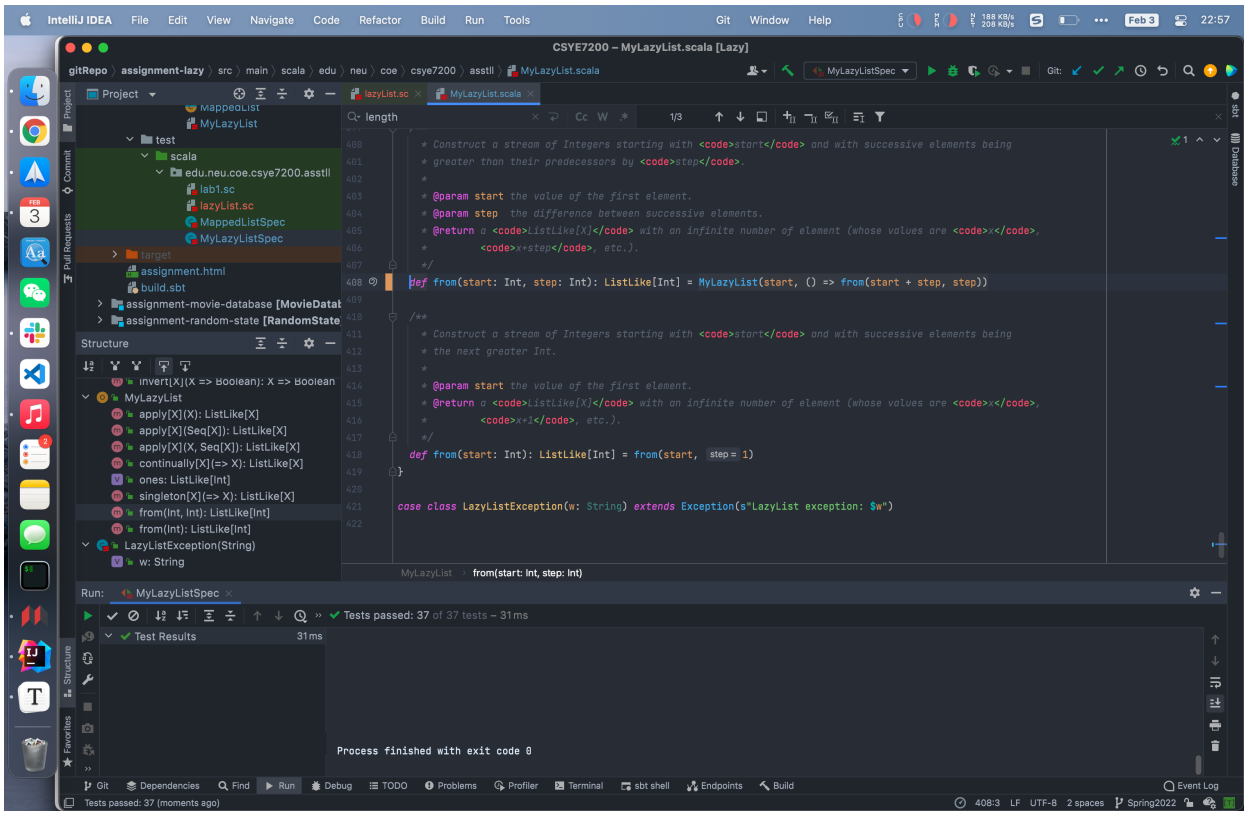
Assignment 2 (Lazy)

> Name: Zhilue Wang
>
> NUID: 001522973

Unit test screenshot



Q1

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.

The `MyLazyList` implementation has only tail lazily evaluated, whereas `LazyList` from Scala library has both head and tail lazily evaluated.

A simple code to show the difference:

```scala
import edu.neu.coe.csye7200.asstll.MyLazyList

def headValue: Int = {
  println("Generating head value")
  1
}

val systemLL = (headValue #:: LazyList.empty).take(0)

val customLL = MyLazyList.apply(headValue).take(0)
```

```
import edu.neu.coe.csye7200.asstll.MyLazyList

def headValue: Int

val systemLL: scala.collection.immutable.LazyList[Int] = LazyList()

Generating head value
edu.neu.coe.csye7200.asstll.LazyListException: LazyList exception: empty
  at edu.neu.coe.csye7200.asstll.EmptyList$.head(MyLazyList.scala:186)
  at edu.neu.coe.csye7200.asstll.EmptyList$.head(MyLazyList.scala:177)
  at edu.neu.coe.csye7200.asstll.LazyListLike.toString(MyLazyList.scala:333)
  at scala.runtime.ScalaRunTime$.inner$1(ScalaRunTime.scala:261)
  at scala.runtime.ScalaRunTime$.stringOf(ScalaRunTime.scala:266)
  at scala.runtime.ScalaRunTime$.replStringOf(ScalaRunTime.scala:274)
  at .lzycompute(<synthetic>:8)
  ... 36 elided
```

The `take` function in `MyLazyList` will return `EmptyList` if the argument is 0, so if its head is lazy then it should not print out the output string defined in `headValue`

(b) Why do you think there is this difference?

Arguments in the constructor of system `LazyList` is defined as call-by-name

https://github.com/scala/scala/blob/v2.13.8/src/library/scala/collection/immutable/LazyList.scala#L1138

The head argument in `MyLazyList` is defined as call-by-value

https://github.com/rchillyard/CSYE7200/blob/Spring2022/assignment-lazy/src/main/scala/edu/neu/coe/csye7200/asstll/MyLazyList.scala#L16

Q2

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

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

The argument `lazyTail` is a `Function0` function which is similar to a lambda function. So here we defined a `tail` function that retrieve the evaluated result from `lazyTail` function by adding a pair of bracket in the end. (Basically it evaluates the tail part)

It also provides an API that can be used to access the tail part of Lazylist outside the class.

Q3

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

26, 43, 68, 82, 98, 116, 131, 361, 383, 388, 408

Q4

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

None

Q5

What is the purpose of the *zip* method?

It combines elements from two lists into tuples, stored in a new LazyList.

```
[1, 2, 3] zip [2, 3, 4] == [(1, 2), (2, 3), (3, 4)]
```

It will terminate whenver it reaches the end of any list.

```
[1, 2, 3, 4] zip [2, 3, 4] == [(1, 2), (2, 3), (3, 4)]
[1, 2, 3] zip [2, 3, 4, 5] == [(1, 2), (2, 3), (3, 4)]
```

Q6

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

We can still implement the length function, an implementation could be something similar to code below:

```
def length() = 1 + lazyTail().length()

// in EmptyList object
def length() = 0
```

Though there are two issues:

- It cannot deal with infinite list
- It will try to evaluate all tails to retrieve all elements in the list and thus lose the advantages of laziness