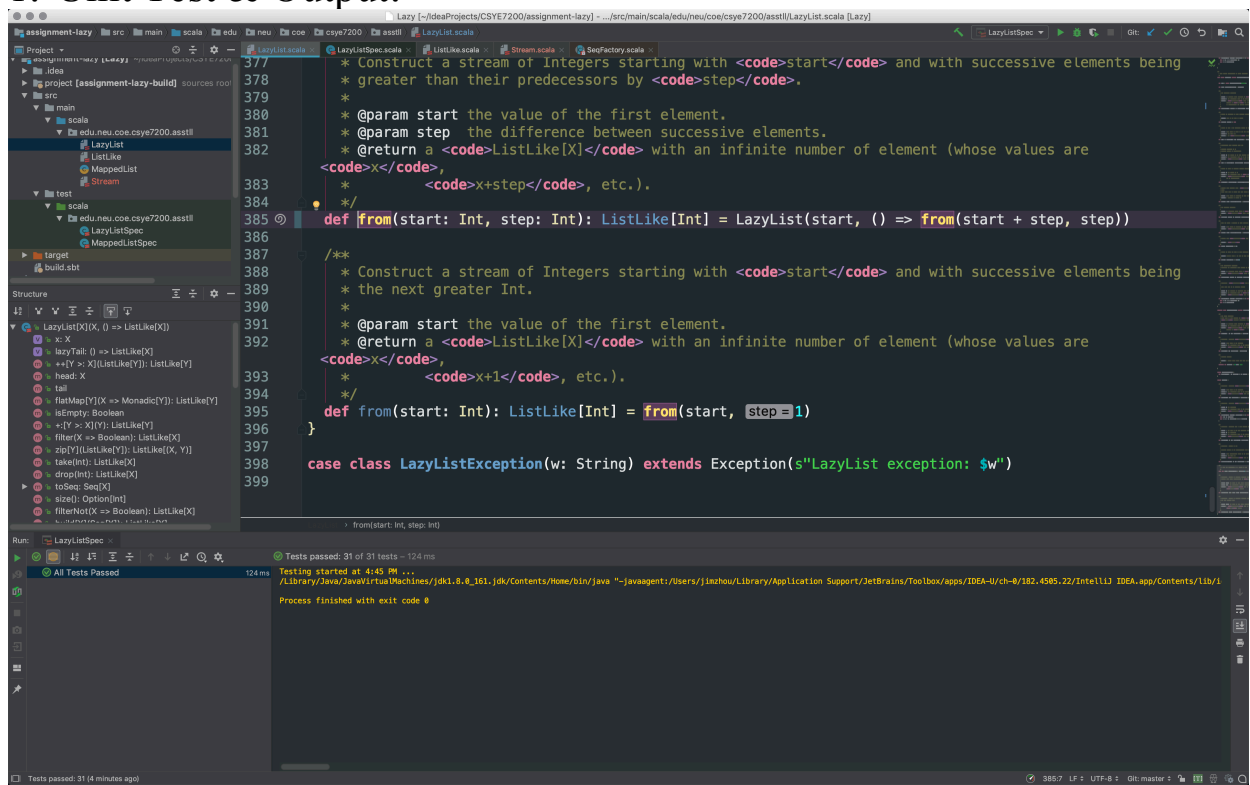


1. Unit Test & Output:



```
377 // Construct a stream of integers starting with <code>start</code> and with successive elements being
378 // greater than their predecessors by <code>step</code>.
379 //
380 // @param start the value of the first element.
381 // @param step the difference between successive elements.
382 // @return a <code>ListLike[X]</code> with an infinite number of element (whose values are
383 // <code>x</code>,
384 // <code>x+step</code>, etc.).
385 def from(start: Int, step: Int): ListLike[Int] = LazyList(start, () => from(start + step, step))
386
387 /**
388 // Construct a stream of Integers starting with <code>start</code> and with successive elements being
389 // the next greater Int.
390 //
391 // @param start the value of the first element.
392 // @return a <code>ListLike[X]</code> with an infinite number of element (whose values are
393 // <code>x</code>,
394 // <code>x+1</code>, etc.).
395 //
396 def from(start: Int): ListLike[Int] = from(start, step = 1)
397 }
398
399 case class LazyListException(w: String) extends Exception(s"LazyList exception: $w")
```

Run: LazyListSpec

Tests passed: 31 of 31 tests - 124 ms

Testing started at 4:45 PM ...

Process finished with exit code 0

2. Questions:

- a. (a) what is the chief way by which LazyList differs from Stream (the built-in Scala class that does the same thing). Don't mention the methods that LazyList does or doesn't implement--I want to know what the structural difference is.

First of all, the LazyList class is a case class, which Stream is not.

Second, the LazyList class has constructor

Third, LazyList class is same as the cons class in Stream, the Stream class is a wrap class of cons. Which makes Stream become a lazy list of Cons, and for each Cons cell is lazy for tail too.

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

First, the case class will optimize the LazyList class when it in pattern match.

Second, the constructor can be used explicit create a instance, but it also been covered in object with apply method, where the Stream class do the same thing.

Third, Since the Stream is build with Cons, which means it can easier to find tail by using less memory, however, for our LazyList, if we want to

reach the tail, we should evaluate each one until we find the last one. That will cause Stack over flow.

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

def tail = lazyTail()

It makes the LazyList class become a lazy evaluate. The head is eager evaluated, but the tail is a call by name function. When tail is called, it will evaluate and return a ListLike object which contain the rest list behind the head. And it is also a lazylist, only the head is evaluated. If we don't have this method, the whole list will be evaluated.

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

128, 369, 385.

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

40, 67, 374.

- f. What is the purpose of the zip method?

Make two separate Stream or list, become one. And each nodes becomes a Tuple2. In my opinion, the purpose is to process two Stream at the same time. Since the whole Stream is lazy. We can zip another Stream into the current one in any time, any position. Which is beneficial to big amount of data processing.

- g. Why is there no length (or size) method for LazyList?

Because the LazyList is build with the actual list, only different is it evaluate the tail lazily, and eager evaluate for head. However, in order to get the size of the whole list, we must evaluate every element in the list. When the list is infinite, the stack or memory will overflow. Different for the Stream, the stream is build upon cons. Therefore, it is a lazy list of con cells. When we want to find the tail, we just need to evaluate each cell, when we done with one cell, we can pop it from stack, and evaluate the next cell. In this way, we will not (barely not) get overflow. Therefore, the Stream can have length method, but LazyList can not.