Harshil Shah – SEC01 (NUID 002780887)

Big Data System Engineering with Scala
Spring 2023
Assignment No. 2 (Lazy)

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

---

- List of Tasks Implemented

1. Implemented the *form* method of MyLazyList.scala
2. Find answers to the following questions:

- (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?

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

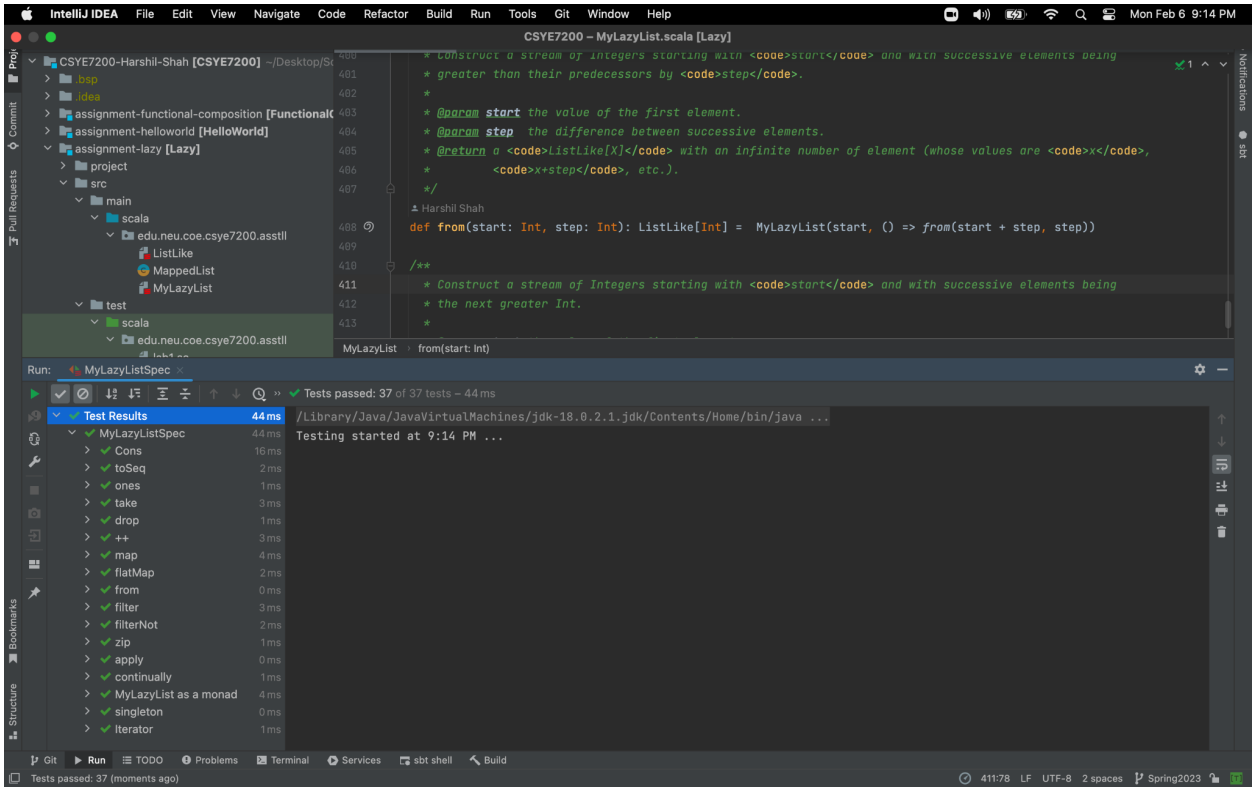
  def tail = lazyTail()

- List all of the recursive calls that you can find in MyLazyList (give line numbers).
- List all of the mutable variables and mutable collections that you can find in MyLazyList (give line numbers).
- What is the purpose of the zip method?
- Why is there no length (or size) method for MyLazyList?

- Code

```
/**
  * 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))
```

- Unit tests



- Result

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.
    - LazyList provides a private, implementation-defined mechanism for lazy evaluation of a list, while MyLazyList provides a concrete implementation of a lazy list that is subclassable. Additionally, LazyList is designed to be used as a base class, while MyLazyList is designed to be used as a concrete implementation.

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

- LazyList can be defined using stream class in scala which implements the LazyList as a base class, whereas in case of MyLazyList, you can define it directly by calling the apply method of MyLazyList

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

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

- This line defines a method named "tail" that returns a "ListLike[X]" object which is the result of the lazy evaluation of the "lazyTail()" method.
- By using the function lazyTail instead of a concrete list object, the evaluation of the tail of the list can be deferred until it's actually needed. This allows for the creation of infinite or very large lists that can be processed lazily, one element at a time, rather than having to evaluate the entire list all at once.

3. List all of the recursive calls that you can find in *MyLazyList* (give line numbers).
   - Line number 26: `def ++[Y >: X](ys: ListLike[Y]): ListLike[Y] = MyLazyList[Y](x, () => lazyTail() ++ ys)`
   - Line number 43: `MyLazyList(y.head, () => y.tail ++ lazyTail().flatMap(f))`
   - Line 60: `def +:[Y >: X](y: Y): ListLike[Y] = MyLazyList(y, () => this)`
   - Line 70: `if (p(x)) MyLazyList(x, tailFunc) else tailFunc()`
   - Line 82: `case MyLazyList(y, g) => MyLazyList((x, y), () => lazyTail() zip g())`
   - Line 98: `case MyLazyList(h, f) => MyLazyList(h, () => f() take n - 1)`
   - Line 116: `case MyLazyList(_, f) => f().drop(n - 1)`
   - Line 131: `case MyLazyList(h, f) => inner(rs :+ h, f())`
   - Line 164: `case h :: t => MyLazyList(h, t)`
   - Line 170: `case MyLazyList(x, f) => Some(x -> f())`
   - Line 298: `case h :: t => MyLazyList(h, t)`
   - Line 361: `case h :: t => MyLazyList(h, () => apply(t))`
   - Line 372: `def apply[X](x: X, xs: Seq[X]): ListLike[X] = MyLazyList(x, () => apply(xs))`
   - Line 383: `def continually[X](x: => X): ListLike[X] = MyLazyList(x, () => continually(x))`
   - Line 388: `lazy val ones: ListLike[Int] = MyLazyList(1, () => ones)`
   - Line 408: `def from(start: Int, step: Int): ListLike[Int] = MyLazyList(start, () => from(start+step, step))`
   - Line 418: `def from(start: Int): ListLike[Int] = from(start, 1)`

4. List all of the mutable variables and mutable collections that you can find in *MyLazyList* (give line numbers).
   - Since no var is used or no mutable collections are imported, there are no mutable variables or collections in MyLazyList

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

- The purpose of the zip method is to allow for the elements of multiple lists to be processed together in a parallel and efficient manner, instead of iterating over each list individually. This is particularly useful when processing data from multiple collections, as it provides a convenient and readable way to pair corresponding elements from each collection. Additionally, the lazy evaluation of the resulting list allows for efficient processing, as the elements are only generated as needed, rather than being loaded into memory all at once.

6. Why is there no *length* (or *size*) method for *MyLazyList*?
    - MyLazyList is implemented as a lazy, infinite sequence. A length or size method would not be appropriate in this context, as it is not possible to determine the length of an infinite sequence