1.a). Obviously, no case class or case object can be found in the *LazyList* class. We need to use "*LazyList.xxx*" to call the function. *LazyList* is more like a static singleton class in Java.
b). In *MyLazyList*, the case class *MyLazyList* extends the abstract LazyListLike. To see if a particular element is *MyLazyList*, you need to use the "case" keyword to make a decision (pattern matching). In fact, you can have an instance of ListLike.
2. Define a method that will evaluate lazyTail every time this method is called. To use it in certain methods, and later debug certain test cases, you need to remember the current lazyTail value.
3. Recursive calls:
Line 131: case MyLazyList(h, f) => inner(rs :+ h, f())
Line 383: def continually[X](x: => X): ListLike[X] = MyLazyList(x, () => continually(x))
Line 408: def from(start: Int, step: Int): ListLike[Int] = MyLazyList(start, () => from(start + step, step))
4. There is no mutable variables or mutable collections, because mutable variables are leaded by key word "var". And we need to import "scala.collection.mutable" to use mutable collections, otherwise all collections in Scala are by default immutable.
5. It is used to merge a collection to current collection and result is a collection of pair of tuple elements from both collections. If both of the two ListLike collections are not empty, then combine each element from these two collections respectively to form a new ListLike collection. Each element in this new collection is a tuple of the elements from the two original collections.
6. Creating MyLazyList allows you to dynamically create new elements as needed, so that the list has an unlimited number of elements. The size of MyLazyList is indefinite.