CS1102S Data Structures and Algorithms

Assignment 02:
Lists and Iteration

For this assignment, use the following submission procedure:

- Solve your assignment.
- Copy the files

  ```
  MySchemeList.java
  SchemeListUtil.java
  MyLinkedList.java
  Generator.java
  ```

  to your account on `sunfire` to a folder (let's say `cs1102s/assignment2`).
- Go to that folder and execute the `submit` program:

  ```
  % cd cs1102s/assignment2
  % ls *.java
  MySchemeList.java
  SchemeListUtil.java
  MyLinkedList.java
  Generator.java
  % /home/course/cs1102s/bin/submit
  ```

  You will get a message that tells you whether the file has been submitted successfully. The script `submit` can be run several times, and you can remove submitted files later. For details on how to use the submit script, type

  ```
  /home/course/cs1102s/bin/submit -h
  ```

Please note all file names are case sensitive and have to conform to the assignment questions.
Submission is activated 5 days before the submission deadline, which is on 9/2, 6:00PM and will be de-activated at that time.

1. (5 marks) Download the assignment project from `http://www.comp.nus.edu.sg/~cs1102s/java/assignment_02.zip`.

Extact the zip file and create a Java project in Eclipse, as usual.

Look at the package playingWithLists. The interface SchemeList contains functions that should remind you of what can be done to lists in Scheme. You are given a class MySchemeList that implements SchemeList.

Complete this class (body of many methods is missing), and test it using MySchemeListTest.java.

Make sure that car, cdr, setCar, and setCdr throw a java. util .NoSuchElementException exception if applied to an empty list.

Submit the completed class MySchemeList.java.

2. (5 marks) Implement the functions given in class SchemeListUtil.java.

   Test the functions using class SchemeListUtilTest.java. Note that the types of arguments and the return types of the functions use the interface type SchemeList whereas the functions are called in SchemeListUtilTest.java with arguments of type MySchemeList.

   Submit the completed class SchemeListUtil.java.

3. (6 marks) The given interface

```
package playingWithLists;

public interface List<Any> extends Iterable<Any>{

    // get returns the element at position idx
    // throws IndexOutOfBoundsException if
    // idx < 0 || idx >= size of list
    Any get(int idx);

    // set returns the current value at position idx,
    // and replaces it by a newVal.
    // throws IndexOutOfBoundsException if
    // idx < 0 || idx >= size of list
    Any set(int idx, Any newVal);

    // add inserts an element x at a given position idx.
    // all subsequent elements' index increases by 1.
    // throws IndexOutOfBoundsException if
    // idx < 0 || idx > size of list
    void add(int idx, Any x);

    // remove removes the element at a given position idx.
    // all subsequent elements' index decreases by 1.
    // throws IndexOutOfBoundsException if
    // idx < 0 || idx >= size of list
    void remove(int idx);
```

```
}
```

follows the textbook closely. Implement a class MyLinkedList, which extends the class MySchemeList and implements this interface List. Make sure that the List functions throw the exception IndexOutOfBoundsException as indicated in List.java, that the iterator function next() throws the exception java.util.NoSuchElementException, and that the iterator function remove() throws the exception IllegalStateException as described in the textbook. (You may ignore exceptions arising from concurrent modification java.util.ConcurrentModificationException.)

Submit the completed class MyLinkedList.java.

4. (4 marks) Let us say that you are a huge fan of enhanced for loops in Java 5. You would like to write:

```
for (Integer i : First100Integers) {
    System.out.println(i);
}
```

Unfortunately, the enhanced for loop needs an Iterable object after the ":". So let us make one.

First, we need an iterator:

```
class MyIterator implements java.util.Iterator<Integer> {
    Integer current = 0;
    public boolean hasNext() {
        return current.intValue() <= 100;
    }
    public Integer next() {
        return current++;
    }
    public void remove() {
        throw new java.util.NoSuchElementException();
    }
};
```

Now we can use this iterator in an Iterable class:

```
class MyIterable implements Iterable<Integer> {
    public java.util.Iterator<Integer> iterator() {
        return new MyIterator();
    }
}
```

This is how to use it:

```
Iterable<Integer> First100Integers = new MyIterable();
```

```
for (Integer i : First100Integers)
   System.out.println(i);
```

In Java, you can create classes "inline", i.e. whenever you need the class. For example, you can write the class directly after **new**. This is called an "anonymous inner class", because you don't even need to give the class a name. You only need to specify what class it extends:

```
Iterable<Integer> First200Integers
= new Iterable<Integer>(){
        public java.util.Iterator<Integer> iterator() {
           return
              new java.util.Iterator<Integer>() {
                    Integer current = 0;
                    public boolean hasNext() {
                       return current <= 200;
                    }
                    public Integer next() {
                       return current++;
                    }
                    public void remove() {
                       throw new java.util.NoSuchElementException();
                    }
                 };
           }
        };

for (Integer i : First200Integers)
   System.out.println(i);
```

Now this still is quite lengthy. Instead of specifying the next and hasNext methods, we could specify how to get from one value to the next using an op method, and when we are done. These two operations are passed to a Generator object (which is of course Iterable) in form of anonymous inner classes, along with an initial value (here 0):

```
Generator<Integer> First300Integers
= new Generator<Integer>(
        0,
        new Operator<Integer>() { public Integer op(Integer i) {
                                     return i+1; } },
        new Terminator<Integer>() {public boolean done(Integer i) {
                                     return i>300; } });

for (Integer i : First300Integers)
   System.out.println(i);
```

Your job is to complete the class Generator.java such that iteration works as expected.

Here are a couple of other uses of our fancy iterators. First, we iterate through all **long** powers of 2

```
Generator<Long> AllLongPowersOf2
= new Generator<Long>(
        new Long(1),
        new Operator<Long>() { Long op(Long i) {
                                    return i*2; } },
        new Terminator<Long>() { boolean done(Long i) {
                                      return i<0; } });

for (Long i : AllLongPowersOf2)
   System.out.println(i);
```

Here is a method to generate larger and larger strings out of "ab".

```
Generator<String> SomeStrings
= new Generator<String>(
        "",
        new Operator<String>() { String op(String s) {
                                    return s+"ab"; } },
        new Terminator<String>() { boolean done(String s) {
                                      return s.length()>100; } } );

for (String s : SomeStrings)
   System.out.println(s);
```

Submit the completed class Generator.java.