P07 Study Playlist

Programming II (CS300) Fall 2019 Department of Computer Sciences University of Wisconsin-Madison	**Due: 9:59PM on October 30, 2019** Pair Programming NOT Allowed for P07

Overview and Learning Objectives:

This assignment involves first creating a collection of songs. Then, to play through that collection of songs, you will create different kinds of iterators that step through those songs in different orders.

Through the process of completing this assignment, you will get practice utilizing the Iterator and Iterable interfaces from the Java API. You will also get practice implementing a doubly linked list, and doing some algorithm analyses. Although this assignment does not require that you write any test methods, doing so is highly encouraged to help you gain confidence that your code works as intended.

Grading Rubric:

5 points	**Pre-Assignment Quiz:** You should not have access to this write-up without first completing this pre-assignment quiz through Canvas. If you have not already, please complete this quiz as soon as you have access to this course on Canvas.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is correct. To become more confident in this, you must write and run additional tests of your own.
10 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope. For this assignment, we will also review the algorithm analyses that you perform and document in the comments of SongCollection.java.
20 points	**Supplemental Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but check your code against different requirements within this specification.
50 points	**TOTAL**

1. Songs and Nodes

Create a new Java Project in Eclipse (using Java 8) that is called: P07 Study Playlist. Add two new classes to this project named Song and DoublyLinkedNode. Implement each of the constructors and instance methods that are described in these JavaDocs. Notice that the DoublyLinkedNode class makes use of a generic type parameter. These classes can make use of whatever fields you find helpful, but the ONLY public methods in these classes should be those listed in the JavaDocs.

2. Collection of Songs

Next, create a class called SongCollection. This class may only include the following three fields (you are NOT allowed to add any additional fields to this class):

```
private DoublyLinkedNode<Song> head;
private DoublyLinkedNode<Song> tail;
private boolean playDirectionForward;
```

Implement a no-argument constructor, which initializes head and tail to null and forwardPlaylist to true. Then in this same class, implement the following methods with the use of a doubly linked list:

```
public void add(Song song) // adds song to the end/tail of this doubly linked list
                           // when song is null, throws a NullPointerException
public Song remove() // removes and returns song from the front/head of this list
                     // when list is empty, throws a NoSuchElementException
```

3. Playlist and ReversePlaylist Iterators

Create two new classes that implement the java.util.Iterator<Song> interface, named Playlist and ReversePlaylist. The constructors for each of these two classes should take a DoublyLinkedNode<Song> as input. The node that is passed to the Playlist constructor is expected to be the head node of a doubly linked list, and the node that is passed to the ReversePlaylist constructor is expected to be the tail node of a doubly linked list.

Each of these iterators should return a different Song each time their next method is called, and these songs should be returned in the order they were added. The Playlist should return them in front/head to back/tail order, and the ReversePlaylist should return them in back/tail to front/head order. If there are no songs left to be returned by an iterator, it's next() method should instead throw a NoSuchElementException. The hasNext() method for each of these iterators should return true when there are more songs left for it to return, and false otherwise.

4. Collection of Songs

Now it is time to make our SongCollection class iterable, by having it implement the java.lang.Iterable<Song> interface. But the kind of iterator that a SongCollection should return will depend on its forwardPlaylist field. So implement the following mutator method within SongCollection:

```
public void setPlayDirection(boolean isForward)
```

When playDirectionForward is true, a SongCollection's iterator() method should return a Playlist object, otherwise it should return a ReversePlaylist object.

5. Algorithm Analysis

Copy the following comment block into the bottom of your SongCollection.java file:

```
///////////////////////////////////////////////////////////////////////////////////
// For each of the following big-O time complexity analyses, consider the problem
// size to be the number of Songs that are stored within the argument songs, when
// the method is first called.
//
// For analysisMethodA, the big-O time complexity is ____________.
//
// For analysisMethodB, the big-O time complexity is ____________.
//
// For analysisMethodC, the big-O time complexity is ____________.
//
///////////////////////////////////////////////////////////////////////////////////
```

The replace the three blanks with the big-O time complexity for each of the following methods. Note that you do not need to copy these methods into your code, but they should run if you do.

```java
public static void analysisMethodA(SongCollection songs) {
  songs.add(new Song("C is for Cookie.", "Cookie Monster"));
  songs.add(new Song("Rubber Duckie.", "Ernie"));
  songs.add(new Song("Elmo's Song.", "Elmo"));
}

public static void analysisMethodB(SongCollection songs) {
  SongCollection copy = new SongCollection();
  for(Song song: songs)
    copy.add(song);
  for(Song song: copy)
    System.out.println(song);
}

public static void analysisMethodC(SongCollection songs) {
  Iterator<Song> playlist = songs.iterator();
  for(int i=0;i<1000;i++)
    if(playlist.hasNext())
      System.out.println( playlist.next() );
}
```

6. Assignment Submission

Congratulations on completing this assignment! Although you are not required to write tests for this assignment, we highly recommend doing so to gain confidence in the correctness of your solutions. Please also be sure to review the requirements in the course style guide to ensure your submission is in compliance with them. The only five files that you should submit to gradescope for this assignment include: Song.java, DoublyLinkedNode.java, SongCollection.java, Playlist.java, and ReversePlaylist.java. Your final grading and scoring for this assignment is based on your submission that is marked “active” within gradescope, and is submitted prior to this assignment’s hard deadline.