Due: 5:00 PM Friday 10 February 2012

Overview: For this project, you will write a Java class that implements a given interface. This class must use a doubly linked list data structure to provide support for an (already written) image notebook class that will be provided to you.

Project Description: In class we discussed singly linked lists, i.e., each node in the list has a single reference to the subsequent node (if one exists) in the list. For this project, you must implement a doubly linked list data structure, i.e., each node has two references: a reference to the previous node (if one exists) in the list, and a reference to the subsequent node (if one exists) in the list.

On the course web page, you will find the file ObjectListInterface.java. This interface uses the Java Generics mechanism to specify the operations that a list should provide. You must write a class called ObjectList that implements ObjectListInterface<T>. Your ObjectList class must use a doubly linked list which can hold references to any type of object. You may not use any of Java’s standard linked list classes — your doubly linked list must be completely original and self-contained.

Following is a list of the methods contained in ObjectListInterface.java with a brief description of what each should do.

- **getLength()**: Return the number of items \( n \) in your list.
- **getCurrent()**: Returns the current value from the list.
- **getCurrentPosition()**: You must maintain a “pointer” to the current position in the list. This method will return the index (between 0 and \( n - 1 \)) of that pointer (0 if the list is empty).
- **getNext()**: Move to the next position in the list, and return the associated value. If you are currently at the end of the list or if the list is empty, this should be a no-op (i.e., return the last item in the list if you are already at the end of the list, or return null if the list is empty).
- **getPrevious()**: Move to the previous position in the list, and return the associated value. If you are currently at the beginning of the list or if the list is empty, this should be a no-op (i.e., return the first item in the list if you were already at the beginning, or if the list is empty, return null).
- **getFirst()**: Move to the beginning of the list, and return the associated value (if any).
- **getLast()**: Move to the end of the list, and return the associated value (if any).
- **append()**: Place the provided value at the end of the list.
- **insert()**: Place the provided value into the list before the current position.
- `remove()`: Remove the current value. By default, the previous value should then become the current value.

- `replace()`: Replace the current value with the provided value.

- `clear()`: Remove all values from the list.

Note that your class `ObjectList` will work for any object type — you should test your implementation on simple objects (e.g., `Integer`) before testing in the context of the image notebook.

Also on the web page, you will find the files `ImageNotebook.java` and `CanvasPanel.java`. These files provide source code necessary to execute a graphical image notebook in which you can browse, add, and remove GIF and JPEG images. `ImageNotebook.java` presumes the existence of a complete and robust implementation of your class `ObjectList<T>`. Once you have a completed `ObjectList` class that is fully tested, you can use your implementation to support the image notebook by compiling and then executing `ImageNotebook`.

You may not modify the source code for `ImageNotebook` or `CanvasPanel` for this project — I will test your `ObjectList` class using unmodified `ImageNotebook` and `CanvasPanel` classes. All of the methods listed in `ObjectListInterface` are invoked by `ImageNotebook` in the context of the method descriptions provided above.

For your convenience, sample images are provided on the web page. (Note: these images were gathered from public sources — some or all may be subject to copyright.) Be sure to fully stress test your class using the image notebook — remember to check boundary conditions, etc.

**Write-up and Submission:**

- Your program must be properly documented so that I can execute `javadoc` and obtain a complete, meaningful Java documentation HTML page (see Chapter 1 of the text).

- You must supply a separate well-written, logical discussion of your program with sufficient detail so that — by reading this discussion but not looking at your program — another student in the class could construct a correct program and understand the theory on which it is based. At a minimum, the write-up must include discussion of the algorithm(s) you used in your implementation, and of the means by which you tested your implementation.

- Submit a zip archive containing only your `ObjectList.java` file and your write-up to my netfiles inbox cs221 folder by 5:00 PM Friday 10 February 2012.

Your work on this and all programming projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.