CSC 171 LAB 07

Lab Instructions

The goal of this lab is to give you an opportunity to practice again with inheritance and overriding, and a first chance to practice with abstract classes and interfaces.

In your README I would like you to explain what you have learned or observed about interfaces, abstract classes, and classes that extend or implement both.

For this lab, implement all of the code required by the instructions below in a package called "**`csc171.lab07`**". Unless otherwise indicated, you are also required to write a test for each section that demonstrates that your implementation meets the requirements of the section. Your tests should be implemented in a package called "**`csc171.lab07.tests`**".

1. Create a class called "**`Node`**." A **`Node`** has two instance variables: an **`Object`** named "**`item`**," and a **`Node`** named "**`next`**." Create accessors and mutators for both instance variables.

2. Create an interface called "Items." This interface is meant to define the methods for a collection of related **`Objects`**. Remember that the methods in an interface are abstract; they define a signature but do not have a body. Your interface should define at least the following methods:
   a. **`public void add( Object item )`** - adds a single object to the collection.
   b. **`public void addAll( Object[] items )`** - adds all of the elements in the array to the collection.
   c. **`public Object get( int index )`** - returns an item at a specific index.
   d. **`public int size()`** - returns the number of items stored in the collection.
   e. **`public String toString()`** - overrides the **`toString`** method on **`java.lang.Object`** and returns a string that lists all of the items on a single line. For example, if the collection contained the strings "abc", "123", and "cat" the toString method should return something like "abc 123 cat".

3. Define an **`abstract class`** called "**`AbstractItems`**" that **`implements`** the Items interface that you created in section 3. At this point you will not need to provide implementations for any of the methods in the Items interface. **Do not write a test for this section.**

4. In your **`AbstractItems`** class, create an implementation of the "**`addAll`**" method that uses a loop to iterate over the array of objects and calls the "**`add`**" method once for

each object. Do **not** try to implement the "**`add`**" method. **Do not write a test for this section.**

5. In your **`AbstractItems`** class, create an implementation of the "**`toString`**" method that uses a loop to create and return the string described in section 3. You will probably need to use the "**`size`**" method to create your loop, and you will need to use the "**`get`**" method to retrieve each of the items to create your string. Do **not** try to implement the "**`get`**" or "**`size`**" methods. **Do not write a test for this section.**

6. Create a class called "**`ArrayOfItems`**" that extends the **`AbstractItems`** class. Create a constructor that takes an integer parameter called "**`maximumSize`**" and use it to create an object array of that size to hold your items. Use this array to implement the remaining methods of the "**`Items`**" interface: **`add`**, **`size`**, and **`get`**. Note that "**`size`**" should return the number of items that have been added to the array, **not** the maximum size of the array. Your test for this section should test the "**`addAll`**" and "**`toString`**" methods as well as the "**`add`**", "**`get`**", and "**`size`**" methods.

7. Create a class called "**`ListOfItems`**" that extends the **`AbstractItems`** class. To start, create "stubbed" implementations of each of the methods in the **`Items`** interface. A "stub" is a method body that contains the minimum amount of code to get the class to *compile*, but that doesn't really provide a working implementation. For example, a stub for the "**`size`**" method would just return **`0`**, and a stub for the "**`get`**" method would just return **`null`**.

8. Your **`ListOfItems`** class will use the **`Node`** class that you created in section 1. You will need two **`Node`** instance variables. The first should be called "`head`" and holds the first item in your collection. The second is called "**`tail`**" and holds the last item in your collection. Create a parameterless constructor that creates an empty **`Node`** and sets **both** the "**`head`**" and the "**`tail`**" to point to it. You may also want a "**`size`**" variable that starts at **`0`**.

9. In the **`ListOfItems`** class, implement the "**`add`**" method from the **`Items`** interface using **`Nodes`**. The first time an **`Object`** is added to your list, you should add it to the "**`head`**" **`Node`** using the "**`setItem`**" mutator. Each time after that, you should do the following:
    a. Create a new **`Node`**.
    b. Add the item to the new **`Node`**.
    c. Use the "**`setNext`**" mutator on the "**`tail`**" to point to the new `Node`.
    d. Assign the new `Node` to the "**`tail`**" variable.
    e. You may want to increment your "**`size`**" variable if you added one.

    Your test for this section should test the "**`addAll`**" method as well as the "**`add`**" method.

10. Implement the "**`get`**", and "**`size`**" methods on the **`ListOfItems`** class. If you added a "**`size`**" variable, implementing the "**`size`**" method should be trivial. The "**`get`**" method will need a loop to start at the "**`head`**" and use the "**`getNext`**" accessor to move from one **`Node`** to the next, counting as it goes until the specified index is reached. Your test for this section should test the "**`toString`**" method as well as the "**`get`**" and "**`size`**" methods.

HAND IN

Before handing in, create two additional files in your lab directory:

1. Create a README that contains the following:
    a. Your contact information (name, class, lab session), TA name, and assignment number.
    b. A brief (one paragraph at most) description of the assignment.
    c. Instructions explaining how to run your code. While it is possible that some TAs will compile, run, and test your code in Eclipse, it is also possible that they will want to compile and run it from the command line. You should include instructions on how to compile and run each of the executable Java classes.
2. Create a file titled "SampleOutput" showing the results of running your code. (You may copy and paste from the Eclipse console). If appropriate, please include a comment above the output for each section (e.g. "#OUTPUT FOR SECTION 1") and put a few blank lines between each section.

Hand in by uploading the compressed (i.e. "zipped") folder containing your assignment files to Blackboard.