## 187 Programming With Data Structures

Assignment 02 – 70 Points

**Overview**

This assignment focuses on the topics covered in Chapters 3 in *Object-Oriented Data Structures Using Java 3rd Edition*. The exercises include both written and programming problems. You must provide a solution for each exercise completely to receive credit. The written exercises must be submitted as a PDF document and the programming tasks must be submitted as an exported Eclipse project[[1]](#footnote-1). You must submit both before the specified due date to receive full credit.

**Submission Instructions**

You are to submit your solutions to Blackboard before the specified due date. It is your responsibility to ensure that you have submitted your assignment properly. To do this you should upload your solutions to Blackboard and check to make sure that your submission has been received. In addition, if you are not confident that you uploaded your solutions properly it is helpful to download your submission to make sure that what you submitted is what you think you submitted. We will not allow extensions to any assignment, including assignments that were not submitted properly, so it is important that you verify that Blackboard has received your submission correctly before the assigned due date. It is extremely unlikely that Blackboard will be in error so make sure your submission is received.

**Written Exercises**

**Question 1 (5 Points)**: Explain the difference between a programmer-defined exception that extends the Java Exception class and one that extends the Java RuntimeException class.

**Question 2 (5 Points):** Following the style of Figure 3.2 on page 162 of the book, show the effects of the following stack operations, assuming you begin with an empty stack:

Push block5

Push block7

Pop

Pop

Push block2

Push block1

**Question 3 (5 Points):** Describe inheritance of interfaces and explain why it was used in Section 3.4 of the book.

**Question 4 (10 Points):** The Java programming language is useful not only as a language, but also because it comes with an extensive library containing many useful data structures out of the box. An important piece of this library is called the *Java Collections Framework*. For this problem you need to consult the Java API documentation[[2]](#footnote-2) to answer the following questions.

* **(2 Points)** Describe the major differences between the Java library’s java.util.Vector and java.util.ArrayList classes.
* (2 Points) Explain the difference between the books Stack ADT and Java’s java.util.Stack ADT in terms of the pop() method. What methods would you use in java.util.Stack to do:  
    
  Element t = stack.top();  
  stack.pop();
* **(3 Points)** Explain how iterators in the Java Collections Framework are used.
* **(3 Points)** Which classes of the Java library implement the Collection interface?

**Question 5 (5 Points):** What are the main difference, in terms of memory allocation, between using an array-based stack and using a reference-based stack?

**Programming Exercise**

**(40 Points)** The goal of this programming exercise is to solidify your understanding of stacks, exceptions, and other material covered in Chapter 3 of the book. This problem will have you implement a program that could be used as the basis of a web browser’s history mechanism. An important component of a web browser includes the ability to press the back and forward button to visit websites that you have previously clicked links to. In Google Chrome it looks like this:

Back

Forward



For example, imagine you visit the following websites in the order shown:

w1, w2, w3, w4, w5, …, wi, …, wn-1, wn

Now imagine you want to go back to website wi, to do this you would click the back button *n-i* times. Once you get back to website wi, perhaps you then want to go forward to the most recent website you had been looking at. Now you need to go forward *n-i* times until you reach website wn – the most recent website you had been viewing. This forward and backward through a browsers history can be easily implemented using a Stack ADT. The website that is “last in” the history is the website that is “first out” when clicking on the back button – thus, you have a LIFO behavior which is what stacks are designed for!

To implement the web browser history mechanism you will need two stacks B and F. The B stack is used to keep track of websites that you have visited and the F stack is used when the back button is clicked. That is, when the back button is clicked you pop the “site” from B and push it on F. When you click on the forward button you pop from F and push on B. Thus, using two stacks allows you to maintain the proper LIFO ordering required by the history mechanism.

**Instructions**

You are to implement the history mechanism of a web browser as described above using the Java programming language. In particular, you are to extend a BrowserHistory class to implement the backward and forward behavior of the browser’s history. The Visit class represents each site that is visited and is used as the type parameter to the DualStackInterface class. The DualStack class implements the DualStackInterface defined by the following Java code:

public interface DualStackInterface<T> extends UnboundedStackInterface<T> {

public void unpop() throws StackUnderflowExeption;

public void clear();

}

The DualStackInterface operates like a regular stack that is extended with the unpop and clear methods. The unpop method will undo a pop operation and the clear method will clear the stack. The DualStack class must implement the DualStackInterface and implement all the methods defined in DualStackInterface and UnboundedStackInterface. Your implementation *must* use LinkedLists to maintain both stacks – this will further exercise your understanding of linked lists and references in Java. The F stack is used to store a Visit object for each new site we visit and the B stack is used to store Visit objects each time we move back in the browser history – that is, we pop a Visit object from the F stack and push it on the B stack. Here is a brief description of each method in the DualStack class:

boolean isEmpty()

This method returns true if both stacks are empty.

void clear()

This method will remove all elements from the stack.

void push(Visit element)

This method will push an element onto the F stack. If there are any elements on the B stack those elements should be cleared (i.e., B should point to null).

void pop() throws StackUnderflowException

This method will pop an element off the F stack and push the element on the B stack. This corresponds to pushing the back button on the browser to view previously viewed visits (websites). It throws a StackUnderflowException when you try to pop from an empty stack.

Visit top() throws StackUnderflowException

This method returns the element on the top of the F stack. It throws a StackUnderflowException when you try to top from an empty stack.

void unpop()

This method pops a Visit object off the B stack and pushes it on the F stack. This corresponds to the user pressing the forward button on the browser.

The BrowserHistory class needs to be extended to use a DualStackInterface object for keeping track of the browser history. For each method in the BrowserHistory class that is labeled with a TODO you should implement by manipulating the DualStackInterface object. If you are confused just remember how a browser works and mimic that behavior.

**Startup Code**

We provide you with some startup code to get you going. The first thing you should do is import the zip archive of the code into your Eclipse environment. Then, you should spend some time reviewing the code to understand how it is pieced together. We have provided templates for all the classes we mentioned above. You will find TODO comments in the code that show you where you need to add code to implement the BrowserHistory and DualStack classes. We include a BrowserTest class that allows you to test your BrowserHistory and DualStack implementations. You may run the BrowserTest as an interactive test or have it read in a file. It accepts the following commands:

w – simulates a visit to a website

b – simulates a push on the back button

f – simulates a push on the forward button

We provide a single test file (00Test.txt) as an example of running a test from a file. You should construct several additional convincing test files that demonstrate the functioning of your program. For example, test that it throws an exception when you call the forward method on the BrowserHistory with an empty stack.

**Rubric**

You will be graded on the following criteria:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 3 | 5 | 8 | 10 | Total |
| Program Compiled | Did not compile lots of errors – does not count if not implemented | Did not compile, several errors. | Did not compile, some errors. | Did not compile 1 or 2 errors. | Compiled | 10 |
| DualStack | Not implemented | Incorrect implemention with significant problems | Incorrect implementation with some problems. | Implementation is close. Minor problems. | Correct implementation | 10 |
| BrowserHistory | Not implemented | Incorrect implemention with significant problems | Incorrect implementation with some problems. | Implementation is close. Minor problems. | Correct implementation | 10 |
| Test Cases | Not done | Insufficient test cases | Test cases did not cover important edge cases. | Good number of test cases. | Well tested. | 10 |
| **Total Points** | | | | | | 40 |

1. Please see the video on the course website describing how to export Eclipse projects for submission. [↑](#footnote-ref-1)
2. <http://docs.oracle.com/javase/6/docs/api> [↑](#footnote-ref-2)