

1.00/1.001 Introduction to Computers and Engineering Problem Solving

Final Examination - December 15, 2003

Name:	
E-mail Address:	
TA:	
Section:	

You have 3 hours to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary files have already been imported.

Good luck!

<i>Question</i>	<i>Points</i>
Question 1	/ 10
Question 2	/ 10
Question 3	/ 20
Question 4	/ 60
Total	/ 100

Question 1. True or False (10 points)

Answer the following questions about Java by circling TRUE or FALSE as appropriate.

1) If a and b are two references of the same type, then a.equals(b) will return true if and only if a and b refer to the same object.

TRUE

FALSE

2) Collisions will never occur in a hash table that uses chaining.

TRUE

FALSE

3) Java streams are Last In, First Out stacks.

TRUE

FALSE

4) It is not possible to read and write one file using the same stream.

TRUE

FALSE

5) An inorder traversal will visit the nodes in a binary search tree in increasing order.

TRUE

FALSE

Question 2. Conceptual Question (10 points)

There are situations or specific sets of data that can make an efficient algorithm or data structure give atypically inefficient or incorrect performance. Please describe at least two examples and strategies to remedy the problem in both cases.

Question 3. Word Counting (20 points)

Ever hungry in your quest to learn more about Java, you spend some time using Google to track down an electronic version of the seminal Java text: *The Java Programming Language, Third Edition* by Arnold, Gosling, and Holmes. As you settle down for a restful IAP, you notice that the word "java" appears 37 times on the first two pages of this book.

This large number of occurrences leads you to wonder how many times "java" appears throughout the entire book. Thankfully, when you have the electronic format, you can write a program to parse the book and count up all of the occurrences. Remembering your Stream lectures, you are quickly able to write a piece of the program that reads in all of the words as `Strings` and stores them in a `String[]`. You decide that your next course of action will be to build a `Map` that maps the words to the number of times they occur.¹ Specifically, you will map `Strings` to `Integers`.

In the box below, write code that will create a `Map` and fill it so that the keys of the `Map` are the words in the book and the values are the number of times each word appears. The only variable you know about is the `String[] words`. This `String` array contains all of the words as they appear in they book but with each `String` changed to lowercase. For example, the first four words seen in the book are the title: "The Java Programming Language" so,

```
words[0] = "the"
words[1] = "java"
words[2] = "programming"
words[3] = "language"
```

¹Realistically, you'd probably build the `Map` as you read in the electronic format of the book, skipping the process of building the intermediate array of `Strings`. This change is made for the sake of simplicity.

Now that you've built a `Map` as described above, write code in the box below to print out the number of times that the word "java" occurred. Just for kicks, also print out the number of times that "MIT" occurred. Remember that the `words[]` array changes all text words to lower case. For both of these words, you should either print out that the word occurred 0 times or that it occurred N times, where N is positive. If you weren't able to complete the code in Part 1, assume that a `Map`, named `wordMap`, exists and contains the appropriate contents. If you were able to complete the code above, use the name of the `Map` that you created.

Part 2

```
// continuing in the main() method from before

} // end main() method
} // end class WordCount
```

Question 4. Swing Application for Stack

In this question, you will complete 2 java classes, **MyStackModel** and **MyStackView** in order to develop a graphical user interface for a Stack data structure. **MyStackModel** uses an instance of the **SLinkedList** from the lecture (#26) on singly linked lists to implement the stack. If you prefer, you can use the **LinkedList** class from the **java.util** package. Follow the directions in the next two parts to complete the java classes and be sure to take a careful look at the given methods before completing each part.

Part 1. MyStackModel.java (30 points)

In this part, you will complete the **MyStackModel** class, which uses an instance of **SLinkedList** to implement a Stack data structure. If you prefer, you may use the **LinkedList** class from the **java.util** package. If you choose to do this, you will have to change the code below. For each change you make, circle the old code that you are replacing and write your new code nearby. You may assume that all appropriate import statements have been made. The **MyStackModel** class will add or remove objects at the beginning of the linked list, `elements`, to implement the stack.

MyStackModel.java:

```
public class MyStackModel {

    private SLinkedList elements;

    public MyStackModel() {
        elements = new SLinkedList();
    }

    // Remove all the elements
    public void clear() {
        elements.clear();
    }

    // Push object into Stack
    public void push(Object obj) {

        // Problem 1
    }

    // Pop object out of Stack
    public Object pop() throws EmptyStackException {

        // Problem 2
    }
}
```

```
// Get all the elements in Stack
public Object[] getAllElements() {

    // Problem 3
}
}
```

1) Problem 1

Complete the `push()` method, which takes an `Object` as an argument and adds it to the `Stack`

```
public void push(Object obj) {

}
}
```

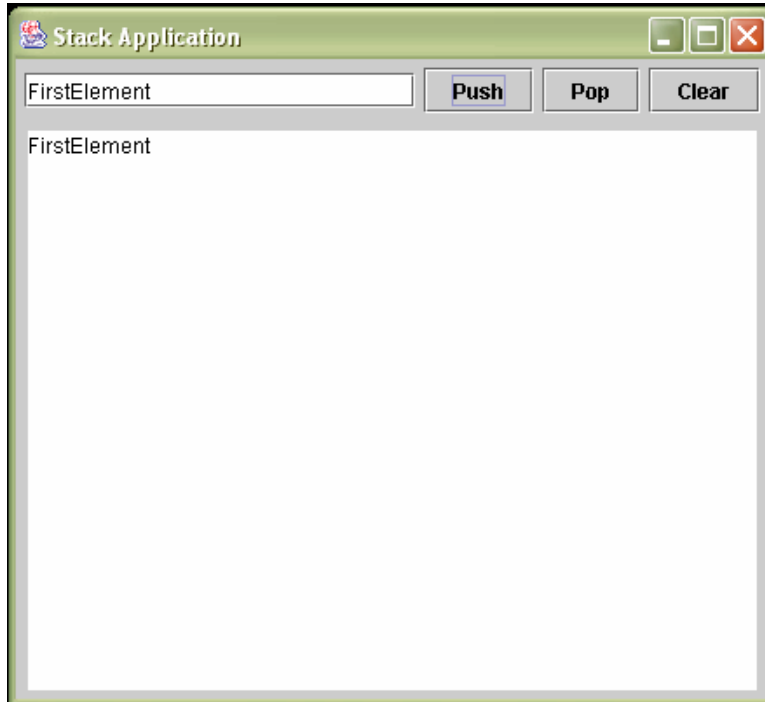
2) Problem 2

Complete the `pop()` method, which pops the object that was last added to your `Stack`. You need to check first whether your `Stack` is empty or not. If your `Stack` is empty, throw an `EmptyStackException` (note the method signature). Otherwise, remove the object from the `Stack` and return it.

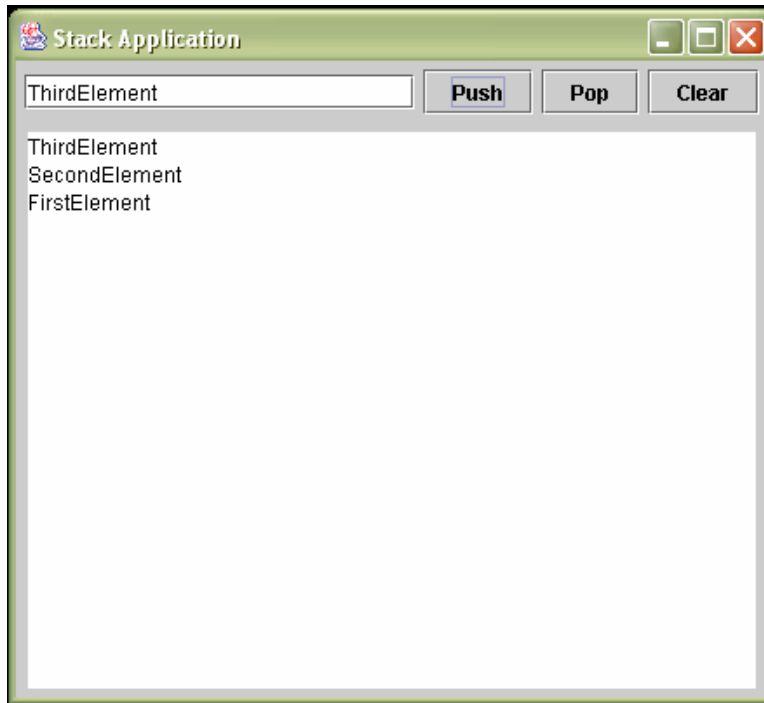
```
public Object pop() throws EmptyStackException {

}
}
```


When user clicks the 'Push' button, your program will get the text from `JTextField` (provided that user has put some text in `JTextField`) and push it onto your Stack (In this problem, all the objects in your Stack are `Strings`.) Also, the `JTextArea` will display all the elements in your Stack with the latest element on top. The `setText()` method of `JTextArea` takes a `String` as an argument and sets the text for `JTextArea`. Take a look at the `clearButton.addActionListener()` in the code on the bottom of p.12 to understand how the `setText()` method works. After adding one element, your application will look like this:

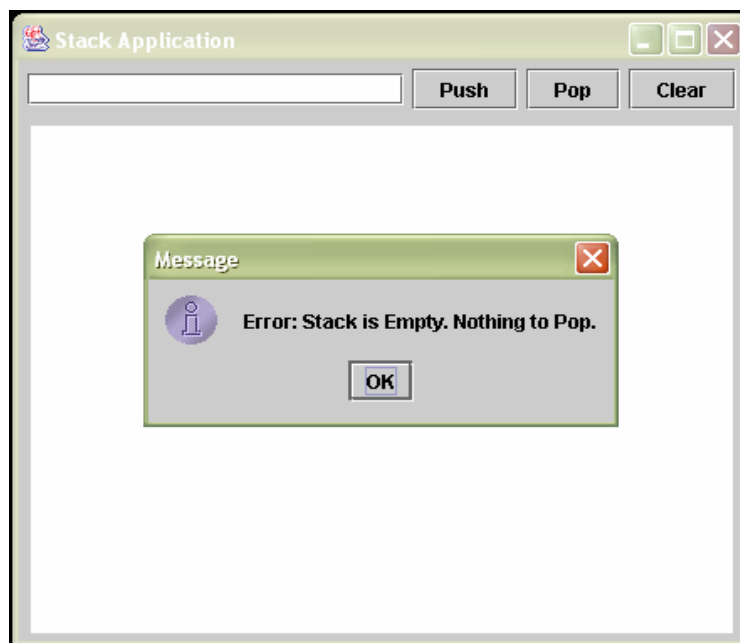


After adding second and third elements, your application will look like this:



Hint: The String "\n" can be appended to a String to start a new line.

When the user presses the “Pop” button, the program should pop the topmost `String` off the stack and throw it away. Please remember that the `pop()` method in `MyStackModel` class throws an `EmptyStackException`. Therefore, when you try to pop an element from an empty `Stack`, your application should generate an error message in a message dialog box:



Here is the code for the `MyStackView` class. Please make sure you understand how all the graphical components are implemented.

MyStackView.java:

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class MyStackView extends JFrame {

    private MyStackModel stack;
    private JPanel controlPanel, stackPanel;
    private JButton pushButton, popButton, clearButton;
    private JTextField inputField;
    private JTextArea stackArea;

    public MyStackView() {

        stack = new MyStackModel();

        controlPanel = new JPanel();
        stackPanel = new JPanel();

        pushButton = new JButton("Push");
        popButton = new JButton("Pop");
        clearButton = new JButton("Clear");

        inputField = new JTextField(20);
        stackArea = new JTextArea(20, 38);

        controlPanel.add(inputField);
        controlPanel.add(pushButton);
        controlPanel.add(popButton);
        controlPanel.add(clearButton);
        stackPanel.add(stackArea);

        getContentPane().add(controlPanel, "North");
        getContentPane().add(stackPanel, "Center");

        pack();
    }
}
```

```

// Add action listener for 'Clear' button
clearButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stack.clear();
        stackArea.setText(updateList());
    }
});

// Add action listener for 'Push' button
pushButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        // Problem 1
    }
});

// Add action listener for 'Pop' button
popButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        // Problem 2
    }
});

}

public String updateList() {

    // Problem 3
}
}

```

1) Problem 1

Complete the `actionPerformed` method that will be called by the 'Push' button. First, get the text from `JTextField` and push it onto the Stack. Then, update the `JTextArea` using the `updateList()` method. (Hint: Take a look at the `actionPerformed` method for the 'Clear' button)

```

pushButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

}});

```


3) Problem 3

Complete the `updateList()` method. This method gets all the elements from `MyStackModel` and creates a `String` that lists all the `String` objects in `Stack`. Make sure that `JTextArea` displays **one String in each line** (Refer to the figures on previous pages).

```
public String updateList() {
```

```
}
```