Final Exam

Eshan Uniyal eshanuniyal@g.ucla.edu
205172354

Problem 1
D. No exceptions will be thrown (Java doesn't throw exceptions for failing to close an InputStream).

Problem 2
A. Only hi1 (since `catch` blocks are assessed sequentially and at most one `catch` block can be run, and since, in the given example, the first block catches the error)

Problem 3
A. True (using the `setBounds` method)

Problem 4
A. True (while there are other ways to implement ActionListeners - such as using anonymous classes or having the GUI class itself extend ActionListener and have an `actionPerformed` method - private inner classes and lambda expressions are the two clean ways to implement an ActionListener for any component, including JButtons)

Problem 5
A. True (by definition of Autoboxing)

Problem 6
F. `public <T> void Copy(List<? super T> dest, List<? extends T> src)`
(the bounds guarantee the objects in list `src` can be added to the list `dest`)

Problem 7

```java
import java.util.ArrayList;
import java.util.EmptyStackException;

public class MyStack<E> {

    // private field to track stack
    private ArrayList<E> stack;

    // constructor
    public MyStack() { stack = new ArrayList<>(); }

    // size of stack
    public int size() { return stack.size(); }

    // add element to top of stack
    public void push_back(E e) { stack.add(e); }

    // remove element from top of stack
    public E pop_back() throws EmptyStackException {
        // checking if stack is empty
        if (size() == 0)
            throw new EmptyStackException();
        // stack isn't empty -> remove and retrieve last element
        else return stack.remove(size() - 1);
    }

    // access top of stack
    public E peek() {
        // checking if stack is empty
        if (size() == 0)
            return null;
        // stack isn't empty -> return last element
        else return stack.get(size() - 1);
    }
}
```

Problem 8

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

public class GUI {

    // fields aren't strictly necessary for this problem,
    // but good to have in case we were to add more methods
    private JFrame guiFrame;
    private JTextField field1, field2, sumField;
    private JButton addButton;

    // constructor
    public GUI() {
        // initialising frame, text fields, and add button
        guiFrame = new JFrame();
        field1 = new JTextField();
        field2 = new JTextField();
        sumField = new JTextField();
        addButton = new JButton("Add");

        // formatting sum field
        sumField.setEditable(false);
        sumField.setBackground(Color.WHITE);

        // setting layout to GridLayout with 4 columns and 2 rows
        guiFrame.setLayout(new GridLayout(4, 2));
        // adding components in
        guiFrame.add(new JLabel("Number 1:"));
        guiFrame.add(field1);
        guiFrame.add(new JLabel("Number 2:"));
        guiFrame.add(field2);
        guiFrame.add(new JLabel("Sum:"));
        guiFrame.add(sumField);
        guiFrame.add(addButton);

        // adding action listener to the add button
        addButton.addActionListener(e -> {
            // extracting numbers in fields 1 and 2
            int number1 = Integer.parseInt(field1.getText());
            int number2 = Integer.parseInt(field2.getText());
            // adding and setting text for sumField
            sumField.setText(Integer.toString(number1 + number2));
        });

        // setting default close operation, size, et cetera on guiFrame
        guiFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        guiFrame.setSize(400, 200);
        guiFrame.setVisible(true);
    }
}
```

Problem 9

```
// algorithm: find smallest element in array, and make it swap places with the first element
// then find smallest element in subarray (ignoring first element) and make that swap places with the second element
// if length of the array is n, iterate n - 1 times

// creating array list of given characters in order
ArrayList<Character> ordering = new ArrayList<>();
ordering.add('k'); ordering.add('m'); ordering.add('g');
ordering.add('j'); ordering.add('b');

// iterating three (=str_arr.size() - 1) times (as per algorithm above)
for (int i = 0; i < str_arr.size() - 1; i++) {
    // by default, we assume first element in subarray is smallest
    String smallestString = str_arr.get(i); // default smallest string (per given ordering)
    int smallestStringIndex = i;    // index of default smallest string
    // iterating over the rest of the subarray
    for (int j = i + 1; j != str_arr.size(); j++) {
        String currentString = str_arr.get(j);
        // checking if current string is smaller than the currently-known smallest string
        // iterating over current string till it differs from smallest string
        // (or either string runs out)
        int index = 0;
        while (index + 1 <= smallestString.length()
                && index + 1 <= currentString.length()
                && smallestString.charAt(index) == currentString.charAt(index))
            index++;
        // if either string ran out, that means it is a substring of the other
        // the shorter string is therefore smaller
        if (index == currentString.length() || index == smallestString.length()) {
            // checking if current string is shorter
            if (currentString.length() < smallestString.length()) {
                // updating smallestString and smallestStringIndex
                smallestString = currentString;
                smallestStringIndex = j;
            }   // if current string is longer, we don't need to do anything
        }
        // neither string ran out -> compare characters at index
        else {
            // finding indices of differing character in the ordering ArrayList
            // since these characters are different (exiting condition of while loop)
                // one is greater than the other
            int smallestStringCharIndex = ordering.indexOf(smallestString.charAt(index));
            int currentStringCharIndex = ordering.indexOf(currentString.charAt(index));
            // checking if current string is smaller
            if (currentStringCharIndex < smallestStringCharIndex) {
                smallestString = currentString;
                smallestStringIndex = j;
            }
        }
    }
    // swapping element at index i with the smallest string
    String stringToSwap = str_arr.get(i);
    str_arr.set(i, smallestString);
    str_arr.set(smallestStringIndex, stringToSwap);
}
```