NAME: ______________________________________________

CU ID: _______________________________________________

Section TA/time: _______________________________________

You have one and a half hours to do this exam.
All programs in this exam must be written in Java.
The exam consists of 15 pages. Make sure you have all of them.

**************************************************************************

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

**************************************************************************
Problem 1 (20 points)

The great conductor Seiji Ojava has set up a rather tangled Java program with the classes and interfaces shown in Figure 1.

![UML diagram of class and interface relationships](image)

A skeleton code for interface C is shown below:

```java
interface C extends A, B { ... }
```

Answer the following questions about this program.

1. (2 points) Write down a skeleton code for class F.
2. (2 points) Write down a skeleton code for class E.
3. (2 points per statement) Which of the following statements is legal? Explain each answer briefly. No credit will be given unless your explanation makes sense. Each statement should be considered independently of the others.
   i. `A r = new A();`
   ii. `A r = new F();`
   iii. `A r = (A)(new F());`
   iv. `A r = new G();`
   v. `F r = new G();`
   vi. `G r = new F();`
   vii. `G r = (G)(new F());`
   viii. `B r = new I();`


Problem 2 (20 points)

Java Nagila is learning about complex numbers. She has written a class named Complex for representing complex numbers, shown on the next page. She forgot to include methods for updating the real and imaginary parts of complex numbers.

Fortunately, she remembered that she could use inheritance to extend the behavior of classes, so she wrote a new class called BComplex, declared to be a subclass of Complex, that has methods for updating the real and imaginary parts of complex number. Finally, she wrote a client class that tests these classes out. These classes are shown on the next page.

Unfortunately, when she went to compile and run this program, the Java compiler gave her many errors. Fortunately, you are her 211 consultant, so she immediately came to see you for help.

Explain to Java Nagila the errors she made, specifying the line number of each line that needs to be fixed, explaining briefly what the error is, and showing her how to fix it. For example,

Line 6 Constructor must have same name as class name.
    Java is case-sensitive. So change complex to Complex.

Be sure to read the comments to figure out what Java Nagila had intended to write.
1. //class for Complex numbers
2. class Complex {
3.   private float x;
4.   private float y;
5.   //constructor for class Complex
6.   public Complex complex(float r, float i) {
7.     //initialize instance variable x
8.     float x = r;
9.     //initialize instance variable y
10.    float y = i;
11.   }
12.   //return real part of complex number
13.   public int Re() {
14.     return x;
15.   }
16.   //return imaginary part of complex number
17.   public int Im() {
18.     return y;
19.   }
20. }
21. //subclass of Complex that includes methods for
22. //writing to real and imaginary parts of complex numbers
23. class BComplex implements Complex{
24.   //method for assigning to real part of complex number
25.   public void setRe(float r){
26.     x = r;
27.     return null;
28.   }
29.   //method for assigning to imaginary part of complex number
30.   public void setIm(float i){
31.     y = i;
32.     return;
33.   }
34. }
35. //client class
36. class testComplex{
37.   public void main(String arg) {
38.     Complex c = new BComplex(1.0f,2.0f);
39.     c.setRe(3.0);
40.     System.out.println(c.x);
41.     System.out.println(c instanceof BComplex);
42.   }
43. }
Problem 3 (35 points)

The program below computes the frequencies of values (≥ 0) in an array of array of int (AAI) (sometimes called multi-dimensional arrays), and prints a histogram.

The AAI is initialized to the following values:

1st row: 2
2nd row: 5, 4
3rd row: 4, 5, 4

A frequency table is computed for the AAI, i.e. how many times a particular value occurs in the AAI. In our case the frequency table computed would look like this:

<table>
<thead>
<tr>
<th>Value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

The histogram based on these frequencies would look like this:

```
*  
** 
**
```

012345

i.e. each "bar" in the histogram has number of "stars" equal to the frequency of the value indicated below the bar.

The next few pages show skeleton code for a class called ArrayOfArrays. You need to complete this skeleton code, without modifying any code given to you, by answering 7 short questions numbered (a) through (g).
public class ArrayOfArrays {

    public static void main(String args[]) {
        // Creates an AAI which has 3 rows.
        int[][] values = new int[3][];

        // (a) (4 points)
        // Write code which creates the following structure for the AAI:
        // 1st row of length 1, 2nd of length 2 and 3rd of length 3.

        // (b) (3 points)
        // Write code to initialize the rows of the AAI as follows:
        // 1st row: 2
        // 2nd row: 5, 4
        // 3rd row: 4, 2, 4

        // Computes the frequencies and prints the histogram
        int[] frequencies = computeFrequencies(values);
        printHistogram(frequencies);

        // (c) (3 points)
        values[0] = values[2] = values[1];
        // What will the following statement print?
        // The code for method printValues is on the next page.
        printValues(values);

    } // end of main method
}
// Prints an AAI.
public static void printValues(int[][] anAAI) {
    for (int i = 0; i < anAAI.length; i++) {
        for (int j = 0; j < anAAI[i].length; j++)
            System.out.print(anAAI[i][j]);
        System.out.println();
    }
}

// (d) (3 points)
// Write a method to compute the max value in a non-empty array of int.
public static int findMax(int[] anIntArray) {
    // end method

// (e) (4 points)
// Write a method to compute the max value in an AAI.
public static int findMax(int[][] anAAI) {
    // end method
// (f) (8 points)
// Write a method to compute the frequencies of values in an AAI.
// Assume all values in the AAI are greater or equal to 0.
// The method should return an array of int containing the frequencies.
public static int[] computeFrequencies(int[][] anAAI) {

} // end method

// (g) (10 points)
// Write a method to print the histogram, assuming that the values
// in the AAI are between 0 and 9. For example,
// given the present data, it should print the following:
//   *
//   **
// 012345
public static void printHistogram(int[] anIntArray) {

} // end method
} // end of class ArrayOfArrays
This page intentionally left blank.
Problem 4 (25 points)

In this small GUI-based application, we have 3 lights represented by three objects of class Label, corresponding to the green, yellow and red lights of a traffic light signal. We have three buttons for switching on the corresponding light. Only one light can be on at a time, i.e. the lights behave like “radio buttons”. A light is switched on and off by changing the background color of the corresponding Label using the setBackground() method. A light is off when its color is gray (Color.gray). The colors green, yellow and red are represented by the constants Color.green, Color.yellow and Color.red respectively. Initially the green light is on. Note that these lights do not behave like true traffic lights as they are controlled by individual button.

The GUI is shown in Figure 4a and the containment hierarchy is shown in Figure 4b. The skeleton code is provided below. Write the method createGUI() which creates the GUI according to the containment hierarchy in Figure 4b, and the method addListeners() which creates and adds the relevant listeners to the relevant sources. The user should be able to terminate the program when the “close-box” of the top-level window is clicked.

```java
import java.io.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Pseudo Traffic Lights
 */
public class PseudoTrafficLights extends Frame {
    private Panel lightsPanel;
    private Label redLightLabel;
    private Label yellowLightLabel;
    private Label greenLightLabel;
    private Panel buttonsPanel;
    private Button redLightBtn;
    private Button yellowLightBtn;
    private Button greenLightBtn;

    Frame (FlowLayout)
    Panel (GridLayout)
    Label
    Label
    Label

    Panel (GridLayout)
    Button Button Button

    Figure 4
```

```java
    public class PseudoTrafficLights extends Frame {
        private Panel lightsPanel;
        private Label redLightLabel;
        private Label yellowLightLabel;
        private Label greenLightLabel;
        private Panel buttonsPanel;
        private Button redLightBtn;
        private Button yellowLightBtn;
        private Button greenLightBtn;
```
// a. (13 points) Write method createGUI to set up the GUI.
private void createGUI() {
}

// b. (12 points) Write method addListeners to add listeners to the sources
private void addListeners() {
}
// Method to terminate the program
private void terminate() {
    dispose();
    System.exit(0);
}

public PseudoTrafficLights() {
    super("Traffic Lights");

    // Create the GUI
    createGUI();

    // Add the listeners to the sources
    addListeners();

    // Pack and show the window
    pack();
    setVisible(true);
}

// The main method
public static void main (String[] args) {
    new PseudoTrafficLights();
}