CSCI 201L Midterm – Written SOLUTION
Spring 2016
10% of course grade

1. **Abstract Classes and Inheritance** – Answer the following questions about inheritance.
   a. Does the following code compile? **(0.5%)**

No

   b. If the code compiles, what is the output? If the code does not compile, explain the error. **(0.5%)**

Line 8 doesn't compile because it is trying to explicitly call m1() in class AD, but the method is abstract.

```
1  public class Problem1 extends AD {
2     public Problem1() {}
3     public Problem1(AC ac) {
4           ac.m1();
5           ac.m2();
6           this.m1();
7           this.m2();
8           super.m1();
9           super.m2();
10    }
11    public void m1() {
12          System.out.println("1");
13    }
14    public static void main(String [] args) {
15          AC ac = new Problem1();
16          new Problem1(ac);
17    }
18 }
19
20 abstract class AD extends AC {
21    public void m2() {
22          System.out.println("2");
23    }
24 }
25
26 abstract class AC {
27    public abstract void m1();
28    public void m2() {
29          System.out.println("3");
30    }
31 }
```

2. **Serialization** – If an object does not implement the `Serializable` interface, a `NotSerializableException` will be thrown when trying to serialize that object. If an object *does* implement the `Serializable` interface, there are still two types of variables in the class that will not be serialized. Explain the two types of variables that will not be serialized and explain why. **(0.5% + 0.5%)**

Transient variables are specifically defined so that they will not be serialized.

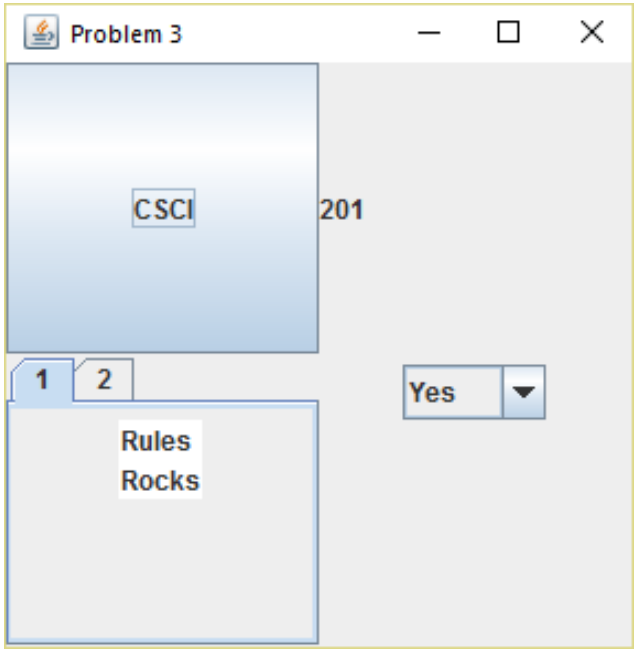
Static variables are not serialized because classes are not serialized but rather object are serialized. Static variables are not associated with an object but are associated with a class.

3. **File I/O and Exception Handling –** Complete the method below to read two integers from the input file, compare them, and write true or false into the output file based on whether the values are the same or not. *You do not need to write all of the include statements, but you do need to write the exceptions that can be thrown.* **(1.0%)**

```
public void compareValues(String inputFile, String outputFile) {
	Scanner scan = null;
	PrintWriter pw = null;
	try {
		scan = new Scanner(new FileReader(inputFile));
		int num1 = scan.nextInt();
		int num2 = scan.nextInt();
		pw = new PrintWriter(new FileWriter(outputFile));
		if (num1 == num2) {
			pw.println("true");
		}
		else {
			pw.println("false");
		}
	} catch (FileNotFoundException fnfe) {
		System.out.println("fnfe: " + fnfe.getMessage());
	} catch (IOException ioe) {
		System.out.println("ioe: " + ioe.getMessage());
	} finally {
		if (pw != null) {
			pw.close();
		}
		if (scan != null) {
			scan.close();
		}
	}
}
```

4. **GUIs and Layout Managers –** Draw the GUI that is generated from the following code. **(2.0%)**

```
1   import java.awt.GridLayout;
2   import javax.swing.*; // importing * to save space
3
4   public class Problem4 extends JFrame {
5     public static final long serialVersionUID = 1;
6     public Problem4() {
7       super("Problem 4");
8       setLayout(new GridLayout(2, 2));
9       add(new JButton("CSCI"));
10      add(new JLabel("201"));
11      JTabbedPane jtp = new JTabbedPane();
12      JPanel jp1 = new JPanel();
13      jp1.add(new JList<String>(new String[]{"Rules", "Rocks"}));
14      JPanel jp2 = new JPanel();
15      jtp.add("1", jp1);
16      jtp.add("2", jp2);
17      add(jtp);
18      JPanel jp3 = new JPanel();
19      jp3.add(new JComboBox<String>(new String[]{"Yes", "No"}));
20      add(jp3);
21      setSize(300, 300);
22      setVisible(true);
23    }
24    public static void main(String [] args) {
25      new Problem4();
26    }
27  }
```



5. **Inner Classes** – `KeyListener` is an interface that contains three methods – `keyPressed`, `keyTyped`, and `keyReleased`, all three of which take a `KeyEvent` as a parameter. Write the code using an anonymous inner class to add a `KeyListener` to a `JTextField` that prints out "Hello" to the command line when a user releases the 'h' key on the keyboard. **(1.0%)**

```
JTextField jtf = new JTextField();
jtf.addKeyListener(new KeyAdapter() {
      public void keyReleased(KeyEvent ke) {
            if (ke.getKeyCode() == KeyEvent.VK_H) {
                  System.out.println("hello");
            }
      }
});
```

6. **Software Engineering –** Agile methodologies were developed long after plan-based methodologies. Give two reasons why agile methodologies, such as Scrum and XP, came into existence. **(0.5% + 0.5%)**

Rapid application deployment became more important during the dot.com days when web development became more prevalent.

Mobile application development demands more frequent updates of code.

Changing customer requirements need to be reflected in development sooner rather than later.

<u>Agile Principles</u> – these are acceptable answers as well.
We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more

Other answers may be acceptable.

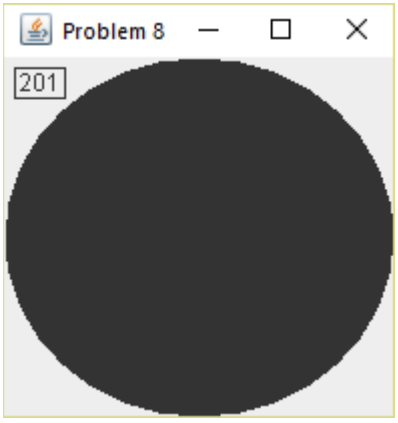
7. **GUI Components –** The `JComboBox` class can fire two different events – `ActionEvent` and `ItemEvent`. Explain when each is fired. **(0.5% + 0.5%)**

   ActionEvent is fired anytime any element is clicked in a JComboBox.

   ItemEvent is fired twice for any selection – once for deselecting an item and once for selecting an item. If the same item is selected, no ItemEvent will be fired.

8. Graphics – Draw the GUI rendered by the following code. **(1.0%)**

```
1   import java.awt.Graphics;
2   import javax.swing.JFrame;
3   import javax.swing.JPanel;
4
5   public class Problem8 extends JFrame {
6     public static final long serialVersionUID = 1;
7     public Problem8() {
8       super("Problem 8");
9       setSize(200, 200);
10      add(new MyPanel());
11      setVisible(true);
12    }
13    public static void main(String [] args) {
14      new Problem8();
15    }
16  }
17  class MyPanel extends JPanel {
18    public static final long serialVersionUID = 1;
19    protected void paintComponent(Graphics g) {
20      super.paintComponent(g);
21      g.drawString("201", 7, 17);
22      g.drawPolygon(new int[]{5, 5, 30, 30},
23                    new int[]{5, 20, 20, 5}, 4);
24      g.fillOval(0, 0, getWidth(), getHeight());
25    }
26  }
```

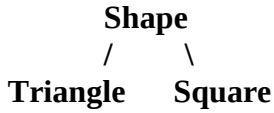


9. **Inheritance –** There are two types of typecasting – upcasting and downcasting. Explain what each one is and why one is potentially dangerous. **(0.5% + 0.5%)**

Downcasting is when you typecast to a type that is lower than the compile-time type of an object.

Upcasting is when you typecast to a type that is higher than the compile-time type of an object.

Downcasting is dangerous because the run-time type may along a different branch of the hierarchy. For example,



If we have a Shape as the compile-time type and typecast it to be a Triangle, it may actually have been a Square instead. This will cause an exception to be thrown.