CSCI 201L Midterm – Written SOLUTION
Fall 2015
10% of course grade

1. Inheritance – Answer the following questions about inheritance.

a. Does Java allow overloading, overriding, and redefining of methods? **(0.5%)**

0.5% - Only overloading and overriding.
0% - if they said all three or anything different.

b. Explain what is meant by **overloading**. Provide a code snippet as an example. **(0.5%)**

0.25% - Overloading is when there are two methods with the same name and different parameter lists
0.25% - Code snippet along the lines of the following:

```
void foo() {

}
void foo(int num) {

}
```

c. Explain what is meant by **overriding**. Provide a code snippet as an example. **(0.5%)**

0.25% - Overriding is when you have the exact same method implemented in a child class that is also implemented in a parent class.
0.25% - Code snippet along the lines of the following. I was expecting Java code, but C++ code would be acceptable as well. If the keyword `virtual` is omitted from C++ code, no credit should be given.

Java

```
class Parent {
      void foo() {
            // some code
      }
}
class Child extends Parent {
      void foo() {
            // some code
      }
}
```

C++

```
class Parent {
      virtual void foo() {
            // some code
      }
}
class Child : public Parent {
      void foo() {
            // some code
      }
}
```

d. Explain what is meant by **redefining**. Provide a code snippet as an example. **(0.5%)**

0.25% - Redefining is when a child class implements the same function that is in the parent class, but the parent class has not declared the function `virtual`.
0.25% - same as C++ code in part c but without the keyword virtual. If virtual is provided, no credit will be given. If Java code is provided, no credit will be given.

2. **Exception Handling** – Java, unlike C++, has both checked and unchecked exceptions. Explain why unchecked exceptions in Java do not need to be handled by a corresponding `try` block. **(0.5%)**

0.25% - Unchecked exceptions can be handled by good programming practices instead of exception handling.
0.25% - Exception handling is a very expensive operation (takes about 10x as long to execute as good programming practice), so handling unchecked exceptions through exception handling should be avoided.

3. **User Interface Design –** Explain two differences between JDialog and JFrame. **(0.5%+0.5%)**

0.5% for each answer, up to two. If more than two answers are provided, subtract 0.25% for each incorrect answer.

1. JDialog can be modal or not modal, meaning that it can force a user to perform some action before being able to return to another window.
2. JDialog does not have the minimize and maximize buttons on the window by default.
3. Other differences may be acceptable.

4. Generics – Answer the following questions based on the generic code below.

```
1   public class Problem4 {
2     public static<T, T1 extends T> void meth(T1 [] list) {
3           T num = list[0];
4           System.out.println(num);
5     }
6
7     public static void main(String[] args) {
8           Integer[] integers = {1, 2, 3, 4, 5};
9           String[] strings = {"London", "Paris", "New York"};
10
11          Problem4.<Number, Integer>meth(integers);
12          Problem4.<Object, Object>meth(strings);
13    }
14  }
```

a. Does the code throw a compile-time error, have a warning, throw a runtime error, or compile and run? **(0.5%)**

0.5% - The code compiles with no warnings and runs.

Answer either question (b) or (c) based on your answer to (a) above. If you answer both questions (b) and (c), you will not get credit for either one.

b. If the code throws a compile-time error, has a warning, or throws a runtime error, explain the problem and provide a solution to it in the above code. **(0.5%)**

0% if this question is answered.

c. If the code compiles and runs, explain what is happening on lines 11, 12, 2, and 3. **(0.5%)**

0.5% for an explanation similar to the following.

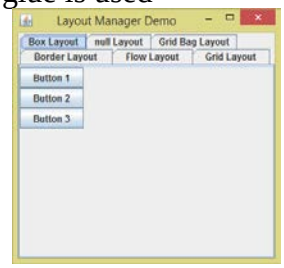
Line 11 is calling the static method meth in the Problem3 class. Since meth is a parameterized method, line 11 is assigning the type Number to T and the type Integer to T1, which must be a subclass of T (as shown on line 2).

Line 3 is then assigning the first object of the array list, which is an Integer, to the variable num, which is a Number (which is fine).

The same thing is happening on line 12, but both T and T1 are of type Object.

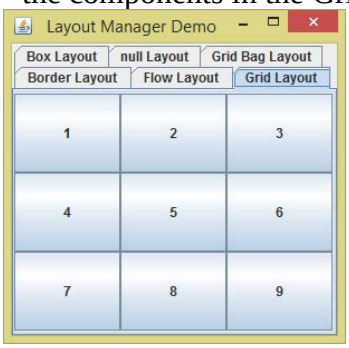
5. **Layout Managers –** For each layout manager below, explain its functionality. Include whether the preferred height and width of a component is acknowledged, how components are laid out when added, and what happens to the components when the frame is resized. Draw (no code needed) a typical GUI that uses each layout manager. **(0.5% + 0.5% + 0.5%)**
    a. **BoxLayout (Y_AXIS)**

0.1% - yes, the preferred height and width of components are acknowledged
0.1% - components are added from top to bottom
0.1% - the components will stay at the top of the BoxLayout when the frame is resized unless glue is used

0.2% 

    b. **GridLayout**

0.1% - no, the preferred height and width of components are not acknowledged
0.1% – components are added from left to right, top to bottom based on how many cells are in the grid
0.1% - the components in the GridLayout will adjust along with the frame

0.2% 

    c. **GroupLayout**

0.1% - yes, the preferred height and width of components are acknowledged
0.1% - components are added to a row and a column, and can be added in any order
0.1% - the components in the GroupLayout will move with the resizing but will not resize themselves
0.2% - any window at all will be correct here

6. **Inner Classes** – Your friend from UCLA sees some of your GUI code and tells you that it won't compile because you have instantiated an abstract class. You respond by saying that this is an anonymous inner class. Explain to your less-educated UCLA friend what is actually happening with an anonymous inner class. Use the following code as an example. **(1.0%)**

```
1  public class Problem6 {
2    public Problem6(B b) {
3      b.b();
4    }
5    public static void main(String [] args) {
6      new Problem6(new B() {
7        public void b() {
8          System.out.println("b");
9        }
10     });
11   }
12 }
13 abstract class B {
14   abstract void b();
15 }
```

1.0% - Answers for this problem will be worded differently, but the answers must have the following statements.

0.4% - The abstract class is not being instantiated.

0.2% - The class that is being instantiated does not have a name (it is anonymous).

0.2% - The class is extending the abstract class B on line 6 and providing the implementation of itself in line on lines 6-10. Class B has one abstract method, which is defined on line 14 and implemented on lines 7-9.

0.2% - An instance of that anonymous class is being passed into the constructor.

7. **Option Panes and Dialogs –** Give two differences and two similarities between option panes and dialog boxes in Java. **(0.25% + 0.25% + 0.25% + 0.25%)**

Similarities

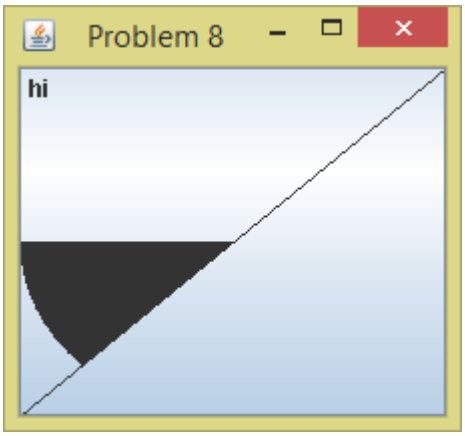
1. JOptionPane and JDialog inherit from Window, Container, and Component.
2. Anything you can do in an option pane, you can do in a dialog box.
3. Other similarities could be acceptable.

Differences

1. Dialog boxes can be customized whereas option panes have limited customization available.
2. Option panes have built in functionality whereas dialog boxes must have all of the code written.
3. Other differences could be acceptable.

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

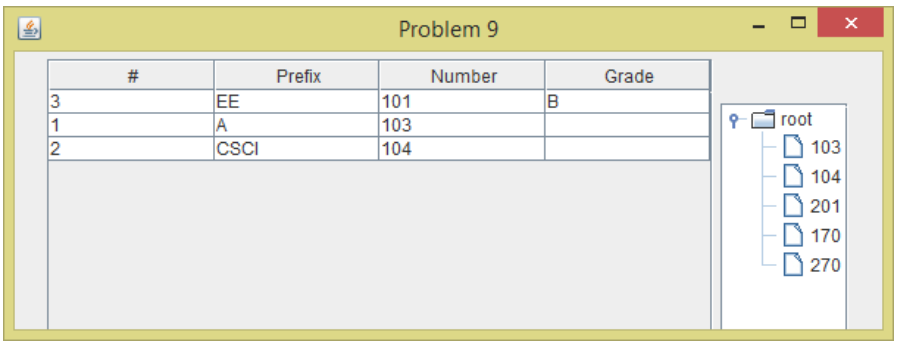
```
1   import java.awt.Graphics;
2   import javax.swing.JButton;
3   import javax.swing.JFrame;
4
5   public class Problem8 extends JFrame {
6      public static final long serialVersionUID = 1;
7      public Problem8() {
8        super("Problem 8");
9        setSize(200, 200);
10       add(new MyButton());
11       setVisible(true);
12     }
13     public static void main(String [] args) {
14       new Problem8();
15     }
16  }
17  class MyButton extends JButton {
18     public static final long serialVersionUID = 1;
19     protected void paintComponent(Graphics g) {
20       super.paintComponent(g);
21       g.drawString("hi", 5, 15);
22       g.drawLine(0, getHeight(), getWidth(), 0);
23       g.fillArc(0, 0, getWidth(), getHeight(), -180, 45);
24     }
25  }
```



0.2% - “hi” is in the top left corner of the window.
0.1% - “Problem 8” is in the title bar.
0.3% - Line drawn from bottom left to top right.
0.3% - Semi-circle drawn from 180 degrees to 205 degrees
0.1% - Semi-circle is filled in

9. Tables and Trees – Draw the GUI that is generated by the following code. **(1.0%)**

```
1   import java.awt.FlowLayout;
2   import javax.swing.JFrame;
3   import javax.swing.JScrollPane;
4   import javax.swing.JTable;
5   import javax.swing.JTree;
6   import javax.swing.table.DefaultTableModel;
7   public class Problem9 extends JFrame {
8      public static final long serialVersionUID = 1;
9      public Problem9() {
10       super("Problem 9");
11       setSize(300, 300);
12       setLayout(new FlowLayout());
13       Object [][] data = new Object[][] {
14         {1, "CSCI", 103},
15         {2, "CSCI", 104},
16         {3, "EE", 101}
17       };
18       Object [] names = new Object[] {"#", "Prefix", "Number"};
19       DefaultTableModel model = new DefaultTableModel(data, names);
20       model.moveRow(2, 2, 0);
21       model.addColumn("Grade");
22       model.setValueAt("B", 0, 3);
23       model.setValueAt("A", 1, 1);
24       JTable jt = new JTable(model);
25       JScrollPane jsp = new JScrollPane(jt);
26       add(jsp);
27       String [] classes = {"103", "104", "201", "170", "270"};
28       JTree tree = new JTree(classes);
29       tree.setRootVisible(true);
30       JScrollPane jsp1 = new JScrollPane(tree);
31       add(jsp1);
32       setVisible(true);
33    }
34    public static void main(String [] args) {
35       new Problem9();
36    }
37  }
```



0.1% - Table drawn with four columns
0.1% - Values in first row are correct
0.1% - Values in second row are correct
0.1% - Values in third row are correct

0.1% - Tree displayed on right/bottom side
0.1% - all numbers at same level in tree
0.1% - Icon is provided next to each number
0.1% - "root" is displayed at the top

0.2% - overall look of the GUI seems correct