CSCI 201L Written Exam #1 - SOLUTION
Fall 2016
10% of course grade

The exam is closed book, closed note, but one 8.5"x11" double-sided paper of hand-written notes is allowed. One hour and 50 minutes will be allowed.

1. **Polymorphism** – Does the following code compile? If so, what is the output? If not, correct the error(s) so that it does compile. (NOTE: You are not allowed to comment or remove any lines.) **(0.5% + 0.5%)**

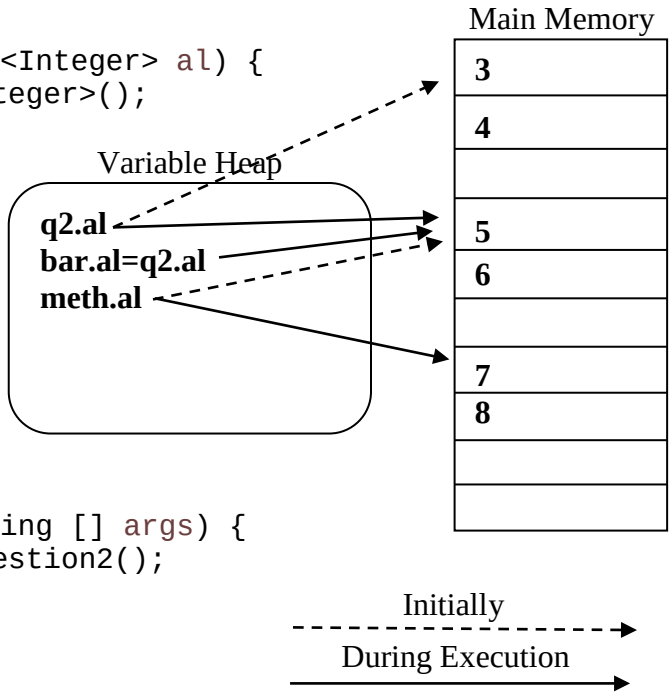
```
1  public class Question1 extends A implements B, C {
2     public void bar() {
3           System.out.println("Q1");
4     }
5     public void foo() {
6           System.out.println("Q2");
7     }
8     public static void main(String [] args) {
9           A a = new Question1();
10          a.bar();
11          a.foo();
12          A a1 = new A();
13          a1.bar();
14          a1.foo();
15          B b = new Question1();
16          b.bar();
17          b.foo();
18          C c = new A();
19          c.foo();
20    }
21 }
22 class A implements B {
23    public void bar() {
24          System.out.println("A1");
25    }
26    public void foo() {
27          System.out.println("A2");
28    }
29 }
30 interface B extends C {
31    public void bar();
32 }
33 interface C {
34    public void foo();
35 }
```

0.5% - Yes, the code compiles
0.5% - for correct output

```
Q1
Q2
A1
A2
Q1
Q2
A2
```

2. Java Basics – What is the output of the following code? Explain your answer by filling in the main memory diagram below. **(0.5% + 0.5%)**

```
1  import java.util.ArrayList;
2  public class Question2 {
3     private ArrayList<Integer> al;
4     public Question2() {
5           al = new ArrayList<Integer>();
6     }
7     private void printArrayList() {
8           for(int i=0; i < al.size(); i++) {
9                 System.out.print(al.get(i) + " ");
10          }
11          System.out.println();
12    }
13    private void bar() {
14          al = new ArrayList<Integer>();
15          al.add(5);
16          al.add(6);
17    }
18    private void meth(ArrayList<Integer> al) {
19          al = new ArrayList<Integer>();
20          al.add(7);
21          al.add(8);
22    }
23    private void foo() {
24          al.add(3);
25          al.add(4);
26          printArrayList();
27          bar();
28          printArrayList();
29          meth(al);
30          printArrayList();
31    }
32    public static void main(String [] args) {
33          Question2 q2 = new Question2();
34          q2.foo();
35    }
36 }
```



0.5% - filling in main memory diagrams similar to as shown above
0.5% - for correct output

```
3 4
5 6
5 6
```

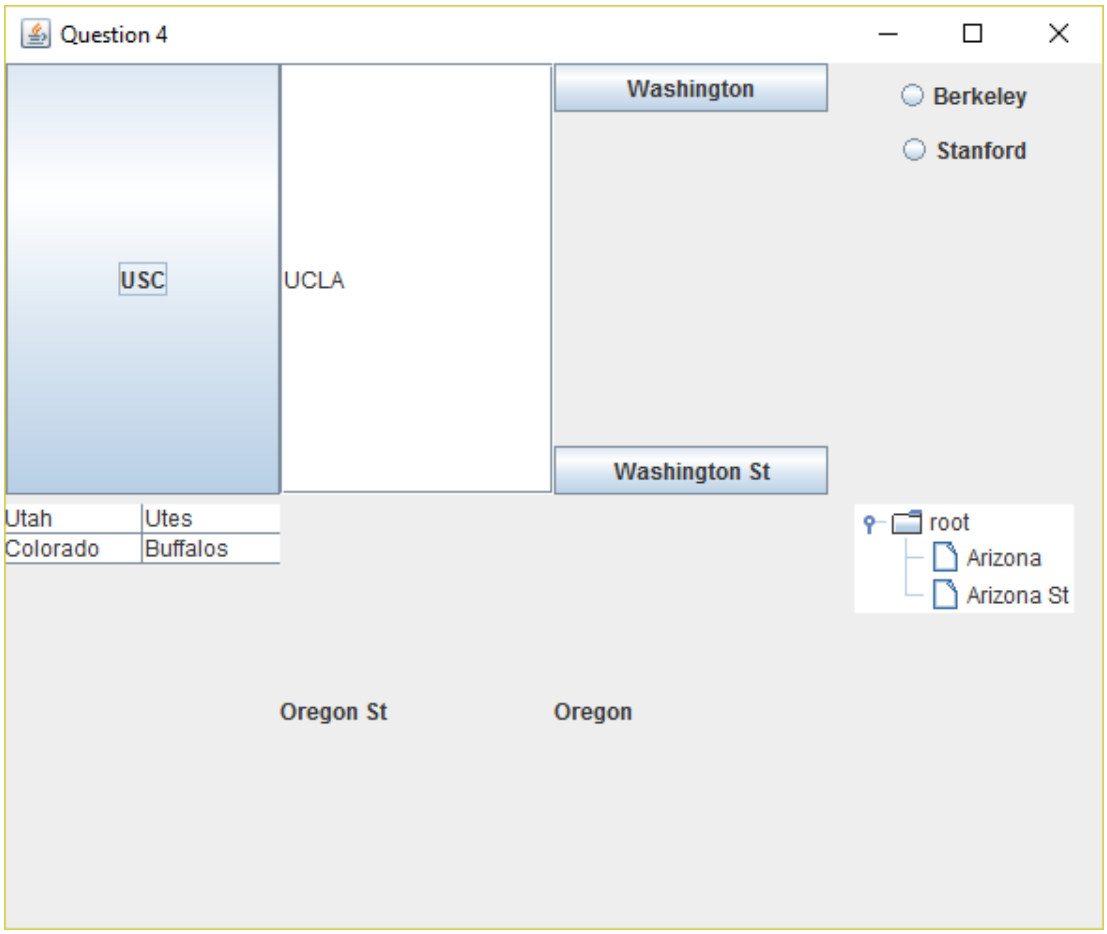
3. **Garbage Collection** – On what line does the memory location pointed to by the variable `arr` declared on line 11 become a candidate for garbage collection? Explain your answer. **(0.5% + 0.5%)**

```
1  public class Question3 {
2     static int [] arr1;
3     public static void foo(int arr[]) {
4           arr1 = arr;
5     }
6     public static void bar(int arr[]) {
7           arr = new int[5];
8     }
9     public static void main(String [] args) {
10          if (args.length == 2) {
11                int[] arr = new int[10];
12                foo(arr);
13                bar(arr);
14          }
15    }
16 }
```

0.5% - `arr` becomes a candidate for garbage collection on line 16
0.5% - Since the value of arr is assigned to the static variable `arr1` on line 4, the memory location pointed to by `arr` will not become a candidate for garbage collection until the variable `arr1` goes out of scope, which never happens for static variables.

4. GUI Programming - Draw the GUI that is generated by the following code. **(2.0%)**

```
1   import java.awt.*;
2   import javax.swing.*;
3
4   public class Question4 extends JFrame {
5    public static final long serialVersionUID = 1;
6    public Question4() {
7     super("Question 4");
8     setSize(600, 500);
9     setLayout(new GridLayout(2, 4));
10    add(new JButton("USC"));
11    add(new JTextField("UCLA"));
12    JPanel jp = new JPanel();
13    jp.setLayout(new BorderLayout());
14    jp.add(new JButton("Washington"), BorderLayout.NORTH);
15    jp.add(new JButton("Washington St"), BorderLayout.SOUTH);
16    add(jp);
17    JPanel jp1 = new JPanel();
18    jp1.add(new JRadioButton("Berkeley"));
19    jp1.add(new JRadioButton("Stanford"));
20    add(jp1);
21    String [] columnNames = {"University", "Mascot"};
22    Object [][] data = {
          {"Utah", "Utes"},
          {"Colorado", "Buffalos"}};
23    JTable jt = new JTable(data, columnNames);
24    JPanel jp2 = new JPanel();
25    jp2.add(jt);
26    add(jp2);
27    add(new JLabel("Oregon St"));
28    add(new JLabel("Oregon"));
29    String [] treedata = {"Arizona", "Arizona St"};
30    JTree tree = new JTree(treedata);
31    tree.setRootVisible(true);
32    JPanel jp3 = new JPanel();
33    jp3.add(tree);
34    add(jp3);
35  }
36  public static void main(String [] args) {
37    Question4 q4 = new Question4();
38    q4.setVisible(true);
39  }
   40 }
```



0.4% - GridLayout arranged with 4 columns and 2 rows
1.6% - 0.2% for each of the 8 cells formatted correctly.
Partial credit can be assigned.

5 **Serialization** – When reading a serialized object, a downcast is typically required. Since downcasting is a potentially dangerous operation, why is it considered acceptable to downcast when using serialization? **(1.0%)**

1.0% When deserializing from a stream, the programmer should know the protocol that is being used, which will include the type of objects. If the protocol is completely unknown, there would be no reason to cast since the programmer wouldn't know anything about the type of data being returned.

Partial credit can be assigned.

6. **Generics** – Explain how generics have caused fewer downcast operations to be required. **(1.0%)**

1.0% - With generics, methods can take a generic type instead of taking an Object as a parameter. When specific methods are called, the generic will determine the compile-time type of the object, so methods in the compile-time type of the object can be called without needing a cast.

Partial credit can be assigned.

7. **Software Engineering –** Give two reasons why managers typically would prefer plan-based methodologies to agile-based methodologies. **(0.5% + 0.5%)**

0.5% - Managers are not able to determine how much of a project is completed as well with an agile methodology compared to a plan-based methodology.

0.5% - With agile development, projects are typically billed as time-and-material rather than fixed-cost since requirements can change. With fixed-cost projects, there is the potential of making more of a profit.

Other answers may be acceptable as well.

8. **Anonymous Inner Classes –** The following code does not compile. Explain the compilation error(s) and fix the code so that it will compile. (NOTE: You are not allowed to comment or remove any lines.) **(1.0%)**

```
1  public class Question8 {
2     public static void foo(C1 c) {
3           c.foo();
4           c.bar();
5     }
6     public static void main(String [] args) {
7           Question8.foo(new C1() {
8                 void foo() {
9                       System.out.println("a");
10                }
11          });
12    }
13 }
14 class C1 {
15    protected int i;
16    public C1(int num) {
17          i = num;
18    }
19    void foo() {
20          System.out.println(i);
21    }
22    void bar() {
23          System.out.println(20);
24    }
25 }
```

0.5% - Line 7 is calling the default constructor in the anonymous inner class, which is calling the default constructor in the inherited class, which is class C1. There is no default constructor in C1, so the code will not compile.

0.5% - Add a constructor that takes no parameters into class C1.
OR
You could add a number into the call on line 7 since anonymous inner classes automatically inherit all of the constructors from parent classes

9. **Exception Handling –** Runtime errors and unchecked exceptions do not need to be handled by a programmer. Unchecked exceptions can be circumvented through good programming practice. Explain why runtime errors cannot be handled by a programmer at runtime. **(1.0%)**

1.0% - Runtime errors include `OutOfMemoryError` and `NoClassDefFoundError`. These are errors that cannot be fixed programmatically at runtime. Instead, a programmer would have to figure out the problem, modify the code, then re-run the program to fix the problem.

Partial credit can be assigned.