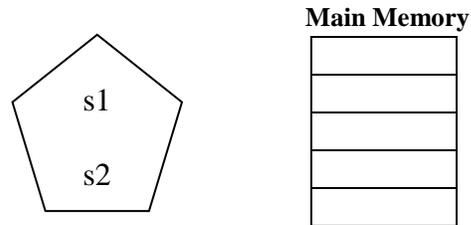


CSCI 201L Midterm – Written SOLUTION
Spring 2015
10% of course grade

1. **Strings** – Do the following two blocks of code produce the same results? Fill in the main memory diagram *and* explain. (0.5% + 0.5%)

```
public boolean equalStrings(String s1, String s2) {
    return s1 == s2;
}

public boolean sameStrings(String s1, String s2) {
    return s1.equals(s2);
}
```

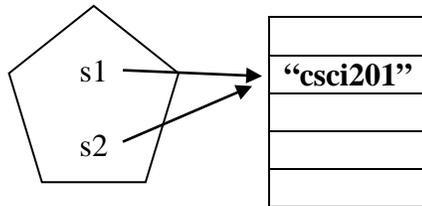


1.0% - if either of the following explanations are provided with the associated main memory diagrams

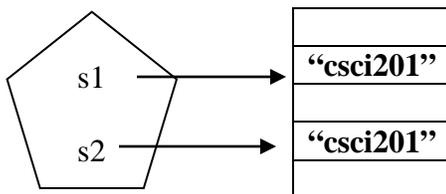
0.5% - if either the diagram or the explanation is provided but not both

0% - if both the explanation and the diagram are not correct

Many times these two methods will return the same result because Strings in Java are immutable, meaning that the same String will be referenced from the same variable. Main memory will look like this typically:



However, the Strings could be stored in different locations in memory, in which case the == would not return the same value as the .equals method.



- 2. Serialization** – We talked about using serialization with file I/O. Give two advantages that serializing an object has over writing code to save all of the variables in a class to a file. **(0.75% + 0.75%)**

0.75% for each reason

- 1. does not require programmer to write very much custom code (just implementing Serializable and creating a serialVersionUID)**
- 2. if the class inherits from other classes, our code would have to save the variables in the parent class as well, which is accomplished by serialization.**
 - a. This may not be possible if the parent class has private variables that do not have accessor methods**
- 3. when reading the file, the class is automatically populated back without having to parse the file**
- 4. <other answers could be acceptable if explained>**

3. **Exception Handling** – Does the following code compile? If it does, what is the output? If it does not, explain the error and provide a solution. (0.5% + 0.5%):

```
1  public class Problem3 {
2      Problem3(int i1, int i2) throws M {
3          try {
4              if (i1 < i2) {
5                  throw new M();
6              }
7          } catch (NumberFormatException nfe) {
8              System.out.println("nfe: " + nfe.getMessage());
9          } finally {
10             System.out.println("leaving constructor");
11         }
12     }
13
14     public static void main(String [] args) {
15         new Problem3(8, 10);
16     }
17
18     private class M extends Exception {
19         public String getMessage() {
20             return "Numbers not formatted correctly";
21         }
22     }
23 }
```

0.5% - The code does not compile because we are not handling the checked exception M.

0.5% - To fix this, we could do one of three things:

- 1. catch M in the Problem3 constructor**
- 2. put a try/catch block in the main method to catch M**
- 3. change class M to inherit from NumberFormatException instead of Exception**

4. **Inheritance** – Provide a code snippet to exhibit the principle of proximity precedence. Explain the principle with respect to your example. (0.5% + 0.5%)

0.5% - Proximity precedence occurs when more than one variable (or method) is named the same within a shared scope. For example, you could have a member variable and a local variable with the same name. Without referring to the member variable with the `this` identifier, the local variable will be used because it is closer in scope.

0.5% for an example of proximity precedence, similar to below

```
1 public class Problem4 {
2     private int num = 2;
3     public Problem4() {
4         int num = 3;
5         // the following line refers to the local variable
6         // num instead of the member variable
7         System.out.println(num);
8         // to refer to member variable, qualify num with this
9         System.out.println(this.num);
10    }
11 }
```

5. **Generics** – Before Java added generics into the language, there were errors that could only be found at runtime. Explain what type of errors generics allow us to find at compile time that would only be caught at runtime in a language that does not have generics. Provide a code snippet to help explain your answer. (0.5% + 1.0%)

0.5% - Generics allow us to specify an object type as a parameter instead of using the Object class as a parameter in a method and having to typecast.

1.0% - here is a code snippet to show the problem. The commented code will compile if uncommented but will cause the program to crash if the Object passed into the method is not a String. The print method on line 2 will not allow the code to compile if a String is not passed into the method.

```
1  public class Problem5 {
2      public static<E extends String> void print(E o) {
3          if (o.length() > 0) {
4              System.out.println("yeah");
5          }
6      }
7      /*
8      public static void print(Object o) {
9          String str = (String)o;
10         if (str.length() > 0) {
11             System.out.println("yeah");
12         }
13     }
14     */
15     public static void main(String [] args) {
16         print("Hello");
17     }
18 }
```

6. **Anonymous Inner Classes** – The following code may appear to some people to be instantiating an interface, which we learned is not possible in Java. If the following code does not compile, explain why and fix it. If the following code does compile, explain what is happening on lines 7-14. **(0.5% + 1.0%)**

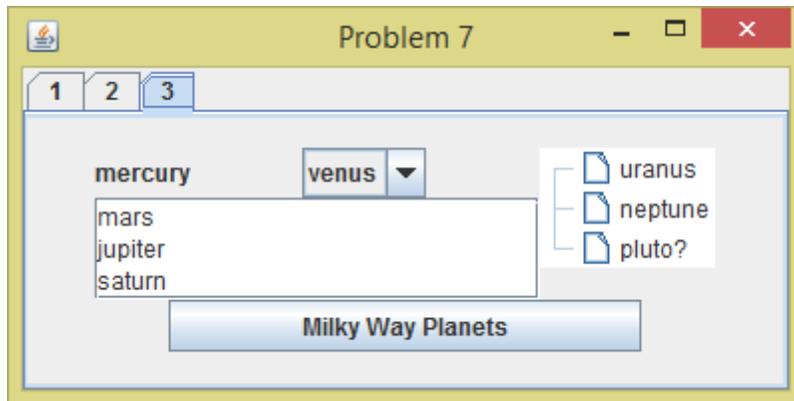
```
1  public class Problem6 {
2      public Problem6(I i) {
3          i.a();
4          i.b();
5      }
6      public static void main(String [] args) {
7          new Problem6(new I() {
8              public void a() {
9                  System.out.println("a");
10             }
11             public void b() {
12                 System.out.println("b");
13             }
14         });
15     }
16 }
17 interface I {
18     void a();
19     void b();
20 }
```

0.5% - The code does compile.

1.0% - We are not instantiating an interface on line 7. We are instantiating an anonymous class (i.e. a class that does not have a name). The class implements the interface I, so we must override both of the methods that are in the interface.

7. Looking at the following code, draw out the GUI that is shown when the application first runs. The code below does compile and run, so do not try to find any errors. (2.5%)

```
1  import java.awt.*;    // note: imported * to save space for exam
2  import javax.swing.*; // not recommended coding practice
3  public class Problem7 extends JFrame {
4      public Problem7() {
5          super("Problem 7");
6          setSize(400, 200);
7          setLocation(100, 100);
8          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9          createGUI();
10     }
11     private void createGUI() {
12         JTabbedPane jtp = new JTabbedPane();
13         JPanel jp1 = new JPanel();
14         jp1.setLayout(new GridLayout(1, 2));
15         jp1.add(new JLabel("world"));
16         JPanel jp2 = new JPanel();
17         JPanel jp3 = new JPanel();
18         jp3.setLayout(new GridBagLayout());
19         GridBagConstraints gbc = new GridBagConstraints();
20         gbc.gridx = 0;
21         gbc.gridy = 0;
22         jp3.add(new JLabel("mercury"), gbc);
23         gbc.gridx = 1;
24         jp3.add(new JComboBox(new String[]{"venus", "earth"}), gbc);
25         gbc.gridx = 0;
26         gbc.gridy = 1;
27         gbc.gridwidth = 2;
28         jp3.add(new JScrollPane(new JTextArea("mars\njupiter\nsaturn", 3,
29                                             20)), gbc);
30         gbc.gridx = 2;
31         gbc.gridy = 0;
32         gbc.gridwidth = 1;
33         gbc.gridheight = 3;
34         gbc.anchor = gbc.FIRST_LINE_START;
35         jp3.add(new JTree(new String[]{"uranus", "neptune", "pluto?"}), gbc);
36         gbc.gridx = 0;
37         gbc.gridy = 2;
38         gbc.gridwidth = 3;
39         gbc.gridheight = 1;
40         gbc.ipadx = 100;
41         gbc.anchor = gbc.CENTER;
42         jp3.add(new JButton("Milky Way Planets"), gbc);
43         jtp.add("1", jp1);
44         jtp.add("2", jp2);
45         jtp.add("3", jp3);
46         jtp.setSelectedComponent(jp3);
47         add(jtp, BorderLayout.CENTER);
48     }
49     public static void main(String [] args) {
50         Problem7 p7 = new Problem7();
51         p7.setVisible(true);
52     } // end class Problem7
```



0.5% - Tabs look correct. Note that Mac will have tabs centered, which is fine

0.2% - Title of window is "Problem 7"

0.5% - overall layout looks correct. This one can have partial credit given

0.3% - "mercury" is shown as a JLabel

0.3% - drop down list only shows "venus"

0.5% - textarea shows "mars", "jupiter", and "saturn" on different lines

0.2% - tree shows "uranus", "neptune", and "pluto?" as shown