1.00

Introduction to Computers and Engineering Problem Solving

Quiz 2 / November 5, 2004

Name:	
Email Address:	
TA:	
Section:	

You have 90 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary files have already been imported.

Good luck.

Question	***Points***
Question 1	**/ 10**
Question 2	**/ 30**
Question 3	**/ 30**
Question 4	**/ 30**
Extra Credit	**/ 10**
Total	**/ 100**

Question 1. True / False (10 points)

1. An event source in `Swing` can have more than one event listener registered with it.

TRUE FALSE

2. An event listener in `Swing` can respond to events from more than one event source.

TRUE FALSE

3. A Java class can inherit from more than one class.

TRUE **FALSE**

4. A Java class can implement multiple interfaces.

TRUE FALSE

5. Consider the following classes defined in separate Java source files:

```
public class Animal
{
      private int age;
}

public class Lion extends Animal
{
      public Lion(int a)
      {
            age = a;
      }
}
```

The above code would compile.

TRUE **FALSE**

Question 2. Swing Components and Events (30 points)

Read the following code carefully and answer questions.
Please assume that all the necessary packages are imported.

```
public class MyApplication     Part 1     {

      private JLabel message;
      private Font myFont = new Font("SansSerif", Font.BOLD, 18);

      public MyApplication() {

            JButton yesButton = new JButton("Yes");
            JButton noButton = new JButton("No");
            message = new JLabel("IS LEARNING JAVA A FUN
                                  EXPERIENCE??");
            message.setFont(myFont);

            JPanel panel = new JPanel();
            panel.setLayout(new BorderLayout());
            JPanel buttonPanel = new JPanel();

            buttonPanel.add(yesButton);
            buttonPanel.add(noButton);
            panel.add(message, BorderLayout.CENTER);
            panel.add(buttonPanel, BorderLayout.SOUTH);

            Container con = this.getContentPane();
            con.add(panel);

            yesButton.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        //
                        // Implementation hidden
                        //
                  }
            });

            noButton.addActionListener(
                  // Part 3
            );

      }

      public static void main(String[] args) {
            MyApplication app = new MyApplication();
            app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            app.pack();
            app.setVisible(true);
      }
}
```

In the above piece of code, new Java learners are asked a question as shown in Figure 1. Depending on the answer, the application either displays the message of Figure 2 on a green background or that of Figure 3 on a yellow background. Look at the code carefully and answer the questions given below:

Figure 1

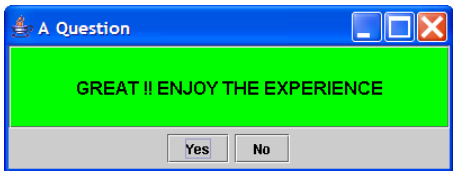


Figure 2

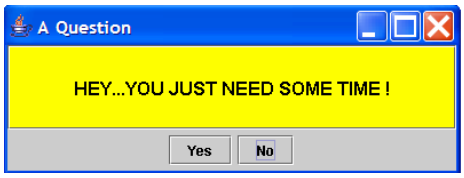


Figure 3

Part 1. The `main()` method invokes the object of the class `MyApplication` and displays it on screen. What should be the correct declaration of the class?

```
public class MyApplication ___extends JFrame_____________
```

Part 2. Assuming the class declaration is correct, identify the top-level window, container/containers, and component/components from above figures. Please do this by inserting the appropriate class and variable names from the code on the previous page in the following table.

Top-level window: **instance of `MyApplication.java`**

Container/containers: **`JPanel panel, buttonPanel`**

Component/Components: **`JLabel message`**
`JButton yesButton, noButton`

Part 3. In the space given below, write an anonymous inner class to handle the events for the 'No' button, so the Figure 3 shown above is correctly displayed.

```
noButton.addActionListener( new ActionListener() {

      public void actionPerformed(ActionEvent e) {

            message.setText("HEY…YOU JUST NEED SOME TIME!");
            message.setBackground(Color.yellow);
            message.setOpaque(true);









);
```

Question 3: Inheritance & Abstract Classes (30 points)

Read the following code carefully and answer the questions. Assume that the code compiles and all the classes `Figure.java`, `Triangle.java`, `Rectangle.java`, and `GetAreas.java` are defined in separate Java source code files within the same package.

```
public class Figure {

      public double height;
      public double width;

      public Figure(double h, double w) {
            height = h;
            width = w;
      }

      public double area() {
            System.out.println("Undefined Area");
            return 0.0;
      }
}
```

```
public class Triangle extends Figure {

      public Triangle(double h, double w) {
            super(h, w);
      }

      public double area() {
            return 0.5 * height * width;
      }
}
```

```
public class Rectangle extends Figure {

      public Rectangle(double h, double w) {
            super(h, w);
      }

      public double area() {
            return height * width;
      }
}
```

```
public class GetAreas {

      public static void main(String[] args) {
            Figure f = new Figure(5, 4);
            System.out.println(f.area());

            Rectangle r = new Rectangle(12.5, 6);
            System.out.println(r.area());

            Triangle t = new Triangle(7, 8);
            System.out.println(t.area());
      }
}
```

Part 1. Show the output when the `main()` method is invoked.

```
Undefined Area
0.0
75.0
28.0
```

Assume the Java source files `Triangle.java` and `Rectangle.java` are UNCHANGED. However, the source file `Figure.java` has been changed as follows.

```
public abstract class Figure {

      public double height;
      public double width;

      public Figure(double h, double w) {
            height = h;
            width = w;
      }

      public abstract double area();
}
```

Similarly, the source file `GetAreas.java` has also been changed to the following.

```
public class GetAreas {

      public static void main(String[] args) {
            Figure f = new Figure(5, 4);
            System.out.println(f.area());

            Rectangle r = new Rectangle(14, 8);
            System.out.println(r.area());

            Triangle t = new Triangle(3, 4);
            System.out.println(t.area());
      }
}
```

Part 2. When you try to compile `GetAreas.java`, do you expect success? If so, show the output below. If you do NOT expect the file `GetAreas.java` to compile, explain why. DO NOT modify `Figure.java`, `Triangle.java`, or `Rectangle.java`. If you do NOT expect the file `GetAreas.java` to compile, make necessary changes by adding, deleting, or modifying the code in `GetAreas.java` to make it compile successfully while still creating at least two geometric shapes and outputting their respective areas. Show the output of your revised code below.

`GetAreas.java` doesn't compile because creating an instance of an abstract class is not allowed in Java.

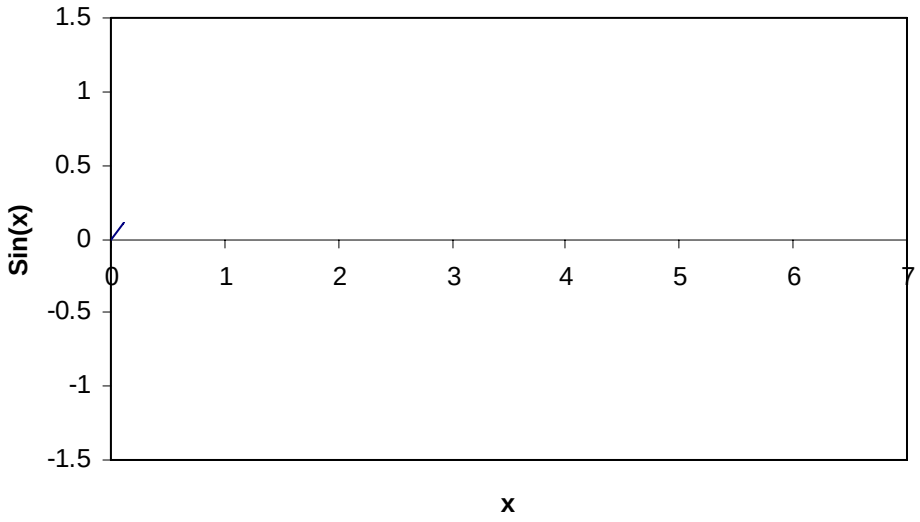
One of the simple ways of fixing the problem is to delete the first two lines in the `main()` method.

`GetAreas.java` will then compile and the corresponding output will be:

```
112.0
6.0
```

Question 4. Finding Minima (30 points)

In this question, you are to write a method for finding the minima of a continuous one dimensional function. Recall that *x* is a local minimum of a function if the values of the function near *x* are greater than the value of the function at *x*. For example, consider the Sine function sketched below. The function has a local minimum approximately at $x = 4.7$ (or more precisely, exactly at $x = 3\pi/2$).



Our method for finding the minimum needs a starting triplet (*a*, *b*, *c*) such that

$a < b < c$ and $f(a) > f(b)$ and $f(c) > f(b)$.

For example, (2, 5, 6) would be a valid triplet for the Sine function sketched above since, by inspection, $f(2) > f(5)$ and $f(6) > f(5)$. Once we have a triplet (*a*, *b*, *c*), an iteration consists of the following.

We have two intervals (*a*, *b*) and (*b*, *c*). Our method guesses the minima to be at the midpoint of the bigger interval. Call this point *z*. For example, if the triplet is (2, 5, 6), two intervals are (2, 5) and (5, 6). Since (2, 5) is the bigger interval, our method guesses the minima to be at (2+5)/2 and hence *z* is 3.5. The function value is computed at *z*.

Suppose $a < z < b$ and $f(z) > f(b)$, our new triplet is (*z*, *b*, *c*). If $a < z < b$ and $f(z) < f(b)$, our new triplet is (*a*, *z*, *b*).

Suppose $b < z < c$ and $f(z) > f(b)$, our new triplet is (*a*, *b*, *z*). If $b < z < c$ and $f(z) < f(b)$, our new triplet is (*b*, *z*, *c*).

In the above example, $a = 2$, $b = 5$, $c = 6$, $z = 3.5$ and hence $a < z < b$. By inspection $f(3.5) > f(5)$ and hence our new triplet is (3.5, 5, 6).

Now that we have a new triplet, we can perform another iteration. We continue to perform iterations until (*c* - *a*) is less than some tolerance provided as an argument to the method. Based on the above description, complete the method for finding a local minimum.

```
public static double minimum(MathFunction test, double a,
                             double b, double c, double tol) {

      double interval1;  // interval1 = b - a
      double interval2;  // interval2 = c - b

      double z;   // mid point of the bigger interval
      double fz;  // function value at z

      while((c - a) > tol) {


            interval1 = b - a;
            interval2 = c - b;

            if (interval1 > interval2) {

                  // Your code here

                  z = (a + b) / 2;
                  fz = test.f(z);
                  if (fz > test.f(b)) {
                        a = z;
                  }
                  else {
                        c = b;
                        b = z;
                  }
            }
            else  {

                  // Your Code here

                  z = (b + c) / 2;
                  fz = test.f(z);
                  if (fz > test.f(b)) {
                        c = z;
                  }
                  else {
                        a = b;
                        b = z;
                  }
            }
      }
      return b;  //b always represents the local minimum
}
```

Extra Credit. Inheritance (10 points)

NOTE: You're strongly advised not to spend any time on this question until you have completed the regular questions.

While we, the 1.00 teaching staff, were creating the exam, we debated the answer to the following question and could not agree with each other. Please give us your answer and explain the reason behind it. These classes are in different Java source code files but in the same package.

```
public class Super {
	int index = 5;

	public void printVal() {
		System.out.println("Super");
	}
}

public class Sub extends Super {
	int index = 2;

	public void printVal() {
		System.out.println("Sub");
	}
}

public class Runner {
	public static void main(String[] args) {
		Super sup = new Sub();
		System.out.print(sup.index + ", ");
		sup.printVal();
	}
}
```

What will be printed to standard output? Circle one.

a) The code will not compile.
b) The code compiles and "5, Super" is printed to standard output.
c) The code compiles and "5, Sub" is printed to standard output.
d) The code compiles and "2, Super" is printed to standard output.
e) The code compiles and "2, Sub" is printed to standard output.

Why?

The answer is (c), "5, Sub", where "5" is the `Super` index value and "Sub" is the `String` printed by the `Sub` class's `printVal()` method. The tricky issue is that the method call to `printVal()` is applied to the class of the object referred to by `sup` (which is class `Sub`, because that is the class to which the "new" operator is applied) but the variable access (to the value of `index`) applies to the declared type of the object referred to by `sup`, which in this case is the type `Super`.

The Java Virtual Machine determines which version of the `printVal()` method to call at runtime depending on whether `sup` refers to an instance of the `Super` or `Sub` class. But when the `main()` method goes to retrieve the value of `sup.index`, it doesn't bother to check whether `sup` is really a reference to an instance of `Super` or `Sub`. The compiler uses the type of the variable `sup` (a reference to an object of type `Super`) AT COMPILE TIME to decide that it should retrieve the `index` value from the base class.