1.00 Introduction to Computers and Engineering Problem Solving

Quiz 2: April 16, 2004

Name:	
Email Address:	
TA:	
Section:	

You have 80 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all the necessary files have already been imported. Good luck.

Question	***Points***
Question 1	**/ 10**
Question 2	**/ 20**
Question 3	**/ 20**
Question 4	**/ 30**
Question 5	**/ 20**
Total	**/ 100**

Problem 1: Abstract Classes & Interfaces (10 points)

Consider an abstract class `Animal`, a concrete class `Bird` derived from `Animal`, a concrete class `Cuckoo` derived from `Bird` and an interface `IColor` implemented by the `Bird` class.

```
public class AnimalTest
{
      public static void main(String[] args)
      {
            Animal a1;                          //Statement 1
            Animal a2 = new Animal();           //Statement 2
            Animal a3 = new Bird();             //Statement 3
           Animal a4 = new Cuckoo();            //Statement 4
            Bird b1 = new Bird();               //Statement 5
            Bird b2 = new Animal();             //Statement 6
            Bird b3 = new Cuckoo();             //Statement 7
            Cuckoo c1 = new Bird();             //Statement 8
            IColor myColor;                     //Statement 9
            IColor myColor2 = new IColor();     //Statement 10
      }
}
```

Given the above statements, please circle the one(s) that would issue a compilation error.

1 2 3 4 5 6 7 8 9 10

Problem 2: Inner and Nested Classes (20 points)

Class `Employee` contains data and methods describing the hard-working staff at MIT. Please help complete the methods inside this class as requested in the two boxes below.

```
public class Employee {

    public static String  officeLoc = "77 Massachusetts Av";
    private String name  = "";
    private double age = 0.0;
    public static String  phoneNo = "234-567-0000";

    public Employee( String n, double a ) {
        name = n;
        age = a;
    }

    public class Record{

      //  Complete printRecord(), a method to output employee
      // data using all of the Employee variable(s) that can be
      // accessed by this method.  Use System.out.println() for
      // output.

         public void printRecord(){

             }
      } // end of class Record


    public static class Information{

      // Complete printData(), a print method to output employee
      // data using all of the Employee variable(s) that can be
      // accessed by this method. Use System.out.println().

         public void printData(){

            }
      }  // end of class Information

}   //end of class Employee
```

Problem 3: Stacks (20 points)

You are given the following implementation of a `StringStack`:

```
public class StringStack {

    private Vector v;

    public StringStack() {
        v = new Vector();
    }

    public boolean isEmpty(){
        return v.size()== 0;
    }

    public void push(String o){
        v.add(o);
    }

    public String pop(){
        Object temp = v.elementAt(v.size()-1);
        v.removeElementAt(v.size()-1);
        return (String)temp;
    }
}
```

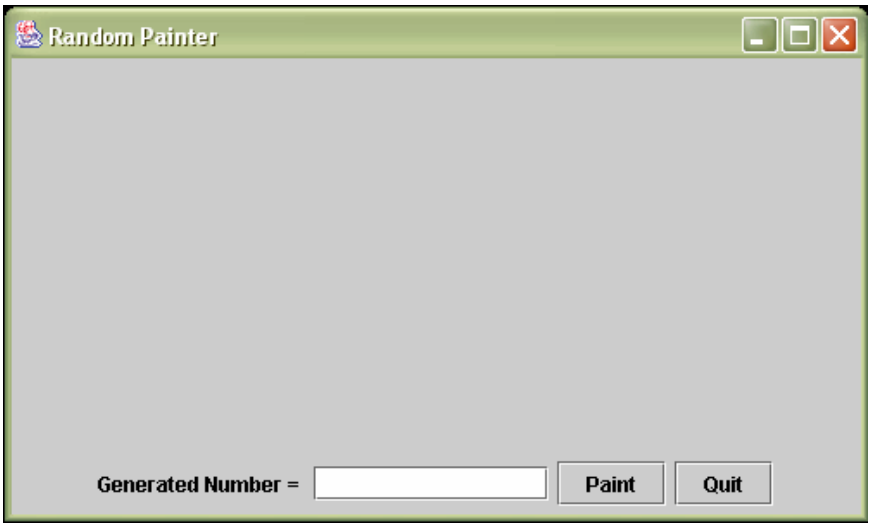
A palindrome (such as 'madam' or 'level') is a word that is read the same way forward or backward. Complete the method below, **which must use a `StringStack`** to check whether a word is a palindrome or not. *Hint:* to process the word character by character, you may find the String method `substring(beginIndex, endIndex)` helpful; it returns the substring from `beginIndex` to `endIndex-1`. All letters are lower-case.

```
public static boolean palindrome(String word) {
```

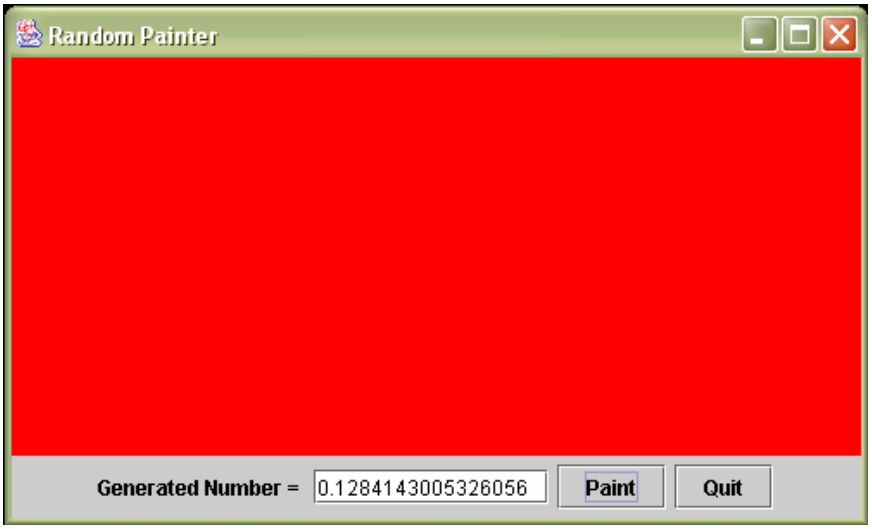
}

Problem 4: Swing Application (30 points)

In this question, you will complete a simple Swing application that generates a random number and paints the window based on that number.



Initial window



Window painted red

When the user clicks the "Paint" button, a random `double` number between 0.0 and 1.0 is generated using the `random()` method of Java's `Math` class. Then, the `JTextField` of the application displays this number and the window gets painted based on this number: the window will be painted RED if the number is **less than or equal to 0.5** and will be painted BLUE if the number is **greater than 0.5** (please refer to the screen-shots above). Also, when the user clicks the "Quit" button, the application should terminate. Assume that all necessary files have been imported and that all code compiles successfully.

```
public class RandomPainter extends JFrame {

	private JPanel paintPanel, controlPanel;
	private JLabel numLabel;
	private JTextField numField;
	private JButton paintButton, quitButton;
	private Random r;
	private double number;

	public RandomPainter()
	{
		this.setTitle("Random Painter");
		this.setSize(500, 300);

		// Initialize components
		paintPanel = new JPanel();
		controlPanel = new JPanel();
		numLabel = new JLabel("Generated Number = ");
		numField = new JTextField(12);

		// Initialize quitButton and add action listener
		quitButton = new JButton("Quit");
		quitButton.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				// Problem 1
			}
		}
		);

		// Initialize paintButton and add action listener
		paintButton = new JButton("Paint");
		paintButton.addActionListener(new ActionListener()
		{
			public void actionPerformed(ActionEvent e)
			{
				// Problem 2
			}
		}
		);

		// Add components to container
		controlPanel.add(numLabel);
		controlPanel.add(numField);
		controlPanel.add(paintButton);
		controlPanel.add(quitButton);
		this.getContentPane().add(paintPanel, "Center");
		this.getContentPane().add(controlPanel, "South");
	}

	public static void main(String[] args)
	{
		// Problem 3
	}
}
```

Part 1:
Complete `actionPerformed` for the "Quit" button. When the user clicks this button, the application should terminate.

```
public void actionPerformed(ActionEvent e) {




}
```

Part 2:
Complete `actionPerformed` for the "Paint" button. When user clicks this button:

1. Generate a random `double` number using `random()` method in the `Math` class.
2. Update `JTextField` with this number.
3. Paint the window based on this number: RED if the number **is less than or equal to 0.5** and BLUE if the number **is greater than 0.5** (You will find the `setBackground()` method of `JPanel` useful).

```
public void actionPerformed(ActionEvent e) {









}
```

Part 3:
Complete the `main()` method, which will:

1. Create a new instance of the `RandomPainter` class.
2. Set the default close operation, so the program terminates when the user closes the window
3. Make the application visible

```
public static void main(String[] args) {









}
```

Problem 5: Basic Numerical Methods and Recursion (20 points)

You are given a version of Java that doesn't contain a method to calculate the square root of a number. Using recursion, we can build one ourselves. Recall that if *y* is the square root of *x*, then $y^2 = x$, so $x / y = y$.

This gives us an idea for an algorithm:

- **Guess** some value *g* for *y* and **test** it.
- Compute x / g.
- If x / g is **close enough** to *g*, return *g*. Otherwise, try a **better guess**.

You may assume that this algorithm will converge to a result. *x* and *g* are positive.

You will write methods that are in a class called `SqrtClass` that contains four methods: `closeEnough()`, `betterGuess()`, `sqrt()` and `test()`.

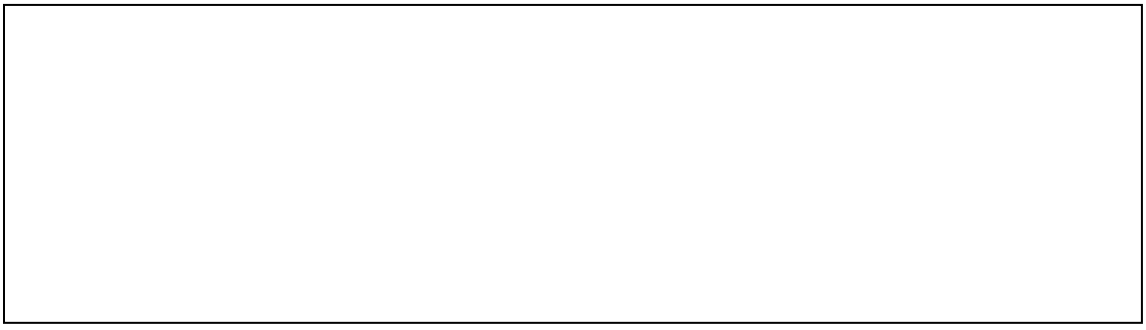
The `sqrt` method is completed for you here:

```
public static double sqrt(double x) {

   return test(x, 1);

}
```

You will need two helper methods to write the `test` method. The first is `closeEnough()`, which returns true if two `doubles`, *a* and *x*, are within .001 of each other.

Part 1: Complete the code for `closeEnough()`:

```
private static boolean closeEnough(double a, double x)
{
```



```
}
```

Part 2: Complete the method `betterGuess()`. `betterGuess()` should return a value closer to the square root of *x* than *g* is. This method brings *x* / *g* closer to *g* by choosing a new guess that is the **average** of *x* / *g* and *g*.

```
private static double betterGuess(double x, double g)
{
```

```
}
```

Part 3: Now complete the test method, using the `closeEnough()` and `betterGuess()` methods:

```
private static double test(double x, double g)
{
```

```
}
```