1.00 Introduction to Computers and Engineering Problem Solving

Quiz 2 - November 7, 2003

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	/ 20
Question 2	/ 5
Question 3	/ 40
Question 4	/ 20
Question 5	/ 15
Total	/ 100

Question 1. True or False (20 points)

Answer the following questions about Java by circling TRUE or FALSE as appropriate.

1) The value of a reference to an array cannot be changed once the reference is assigned.

TRUE **FALSE**

2) Suppose a function $f(x)$ has a unique root x_1. If we use the bisection method with an initial bracket that does not include x_1, we will never converge to the root.

TRUE FALSE

3) An abstract class can contain concrete methods.

TRUE FALSE

4) A class inheriting from an abstract class `A` must override all the methods of `A`.

TRUE **FALSE**

5) If a concrete class `Carpet` implements an interface `IColor`, then the statement

```
IColor c = new Carpet();
```

is legal.

TRUE FALSE

6) If a class inherits from another class, it can access all of the base class's data members and invoke all of the base class's methods.

TRUE **FALSE**

7) An interface can have concrete methods.

TRUE **FALSE**

8) A class can implement only one interface.

TRUE **FALSE**

9) A concrete class that implements an interface doesn't have to define all the methods in that interface, but just the ones the class needs.

TRUE **FALSE**

10) An interface can extend another interface.

Question 2. (5 points)

What is round-off error?
Please give one example that demonstrates its significance.

Question 3. Numerical Methods (40 points)

Ben Bitdiddle, a Java ace from the BetaGadgets Company, has now become a 1.00 TA. Wanting to entertain his students during tutorial, he introduces them to fixed-point iteration and asks them to write a small program to examine its behavior.

To find the roots of a function $f(x)$, fixed-point iteration requires that you rearrange $f(x)$ so that x is on the left-hand side of the equation. This means, for example, that given an equation $f(x) = x - g(x)$, the roots of $f(x)$ can be found by performing fixed point iteration on the equation $x = g(x)$.

In numerical terms, fixed point iteration operates by computing $x_{i+1} = g(x_i)$, where i and $i+1$ are successive iterations. For simplicity, we assume that $f(x)$ is continuous and can always be rearranged in the way described above.

After enlightening his students, Ben asks them to use fixed-point iteration to calculate the roots of the function $f(x) = 1+x-x^2 -x^3$; which means they have to solve the equation $x = x^3+x^2-1$.

Part 1

Ben reminds you that the first step in solving a numerical problem in Java is to write an interface that will reflect the type of functions you expect to deal with. In the space below, write an interface `IMath` that contains a single method `g()`. This method takes a `double` as its sole argument and returns a `double`.

Answer:

```
// INSERT YOUR CODE HERE

    public interface IMath {

        public double g(double x) ;

    }
```

Part 2

Now write a public class `MyFunc` that implements the interface you have written above. Remember that we are dealing with $g(x) = x^3+x^2-1$, which needs to be "encapsulated" somehow inside `MyFunc`.

Answer:

```
// INSERT YOUR CODE HERE

    public class MyFunc implements IMath {

        public double g(double x) {

            return x*x*x + x*x -1;
        }

    }
```

Part 3

We now reach the interesting part, which is the actual implementation of the fixed-point iteration method. You are provided with the following signature for it:

```
public double fixed (IMath func, double initial, double tol)
```

The first argument is an object of a class that implements `IMath`, the second argument is the initial guess and the third argument is the tolerance. We would like to stop iterating when the absolute difference between two consecutive estimates of the root becomes smaller (in absolute value) than the tolerance. Remember that we are iterating using the equation $x_{i+1} = g(x_i)$, where $g(x)$ is the function given in part 2.

Answer:

```
public class RootFinder{

  public double fixed (IMath func, double initial, double tol){

  double prev; //root estimate in previous iteration
  double current; //root estimate in current iteration
  double diff; //difference between 2 consecutive estimates


 // INSERT YOUR CODE HERE

        current = initial;

        do {
            prev = current;
            current = func.g(prev);
            diff = current - prev;
        }
        while (Math.abs(diff)>= tol);

        return current;


    }

}
```

Part 4

The final step is to write a `main()` method to test your program. You would like to find a root of $f(x) = 1+x-x^2 -x^3$ using fixed-point iteration. Use an initial guess of 0.5 and a tolerance of 1E-5.

Answer:

```java
public class Driver{

   public static void main (String[] args){

        double root;

        //INSERT YOUR CODE HERE

        RootFinder m = new RootFinder();
        root = m.fixed(new MyFunc(), 0.5, 1E-5);


        System.out.println(root);
   }
}
```

Question 4. Swing (20 points)

Below we define a class named `Drawing` that extends `JPanel`. Complete the body of the `paintComponent()` method so that it draws a rectangle in the center of the `JPanel`, whatever the size of the `JPanel` is.

The class `Drawing` has a constructor that takes in two arguments which are the length and width of the rectangle to be drawn in pixels.

Hint: `JPanel` has methods `getWidth()` and `getHeight()`.

```
public class Drawing extends JPanel{

    private int length, width;

    public Drawing(int l, int w) {
        length = l;
        width = w;
    }

    public void paintComponent(Graphics g) {

        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;

        int panelwidth = getWidth();
        int panelheight = getHeight();

        g2.drawRect((panelwidth-width)/2,
            (panelheight-length)/2, width, length);

        //using function fillRect instead of drawRect is
        //ok too
        //drawing four lines that forms a rectangle is ok
        //as well

    }
}
```

Question 5. URLs, MalformedURLException (15 points)

The class `java.net.URL` is used to represent a Uniform Resource Locator and points to a "resource" on the World Wide Web. One of the constructors in this class takes a single `String` argument:

```
public URL(String address)
```

With this constructor, you could create a URL for MIT's main webpage:

```
URL mit = new URL("http://web.mit.edu");
```

However, the documentation also says that this constructor will throw a `MalformedURLException` if the `String` argument isn't a properly formed web address. For example, because the `String` argument below isn't a valid URL, the following constructor invocation will throw an exception:

```
URL problem = new URL("O'Doyle RULES!");
```

Note that the constructor for the `URL` class has all the logic that determines whether a given `String` could represent a valid URL. You don't have to worry about doing this.

Write code in the `main()` method below that will:

1. Ask the user to enter a web address using `JOptionPane's` `showInputDialog()` method.

2. Attempt to create a `URL` object based on their input.

   a. If valid, print out `"Valid URL: "` and the address.

   b. If it's not valid, print out `"Invalid URL: "` and the address.

If someone ran this `main()` method and entered `"http://web.mit.edu"` the output would be `"Valid URL: http://web.mit.edu"`.

If someone ran this method and entered `"42"` the output would be `"Invalid URL: 42"`.

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

   String s = JOptionPane.showInputDialog(null,"Enter URL");
   try {
      URL u = new URL(s);
      System.out.println("Valid URL: " + s);
   } catch (MalformedURLException e) {
      System.out.println("Invalid URL: " + s);
      System.exit(1);
   }
   System.exit(0);

   // You can invoke System.exit() with either
   // 0 or 1: this isn't important.  However, invoking
   // showMessageDialog started the AWT Event Thread
   // and the program will hang without an explicit
   // System.exit().




}
```