QUEEN'S UNIVERSITY
SCHOOL OF COMPUTING

CISC212, FALL TERM, 2007
FINAL EXAMINATION
7pm to 10pm, 10 DECEMBER 2007, Jeffery Hall

Instructor: Alan McLeod

If the instructor is unavailable in the examination room and if doubt exists as to the interpretation of any problem, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Please write your answers in the boxes provided. Extra space is available on page 14 of the exam. The back of any page can be used for rough work. This exam refers exclusively to the use of the Java language. Comments are not required in the code you write. For full marks, code must be efficient as well as correct.

This is a closed book exam. No computers or calculators are allowed.

Student Number:

Problem 1:    / 10  Problem 4:    / 10
Problem 2:    / 10  Problem 5:    / 5
Problem 3:    / 30  Problem 6:    / 5

TOTAL:    / 70
Problem 1) [5 marks for a) and 5 marks for b)]

a) In lecture we found that the machine epsilon for a double is 1.1102230246251565E-16, and for a float, 5.9604645E-8. Write the output of the following program beside each println() statement. Remember that a float literal has a capital F appended to the number.

```java
public class Problem1 {
    public static void main(String[] args) {
        System.out.println(1.0 + 1.0e-16 == 1.0); // true
        System.out.println(1.0 + 1.2e-16 == 1.0); // false
        System.out.println(1.0e20 + 1.0e3 == 1.0e20); // true
        System.out.println(1.0e-20 + 1.0e-35 == 1.0e-20); // false
        System.out.println(1.0F + 5.0E-8F == 1.0F); // true
    }
}
```

b) The two equations shown below are algebraically equivalent. Circle the equation that you would use to calculate the value of \( f(x) \) for small values of \( x \) (below about 1.0E-7 radians) in a Java program. Remember that \( \cos(x) \) approaches 1, as \( x \) approaches zero. For example Math.cos(1.0E-7) evaluates to 0.99999999999995.

\[
f(x) = 1 - \cos(x) \quad f(x) = \frac{\sin^2(x)}{1 + \cos(x)}
\]

Why did you choose this equation over the other? (One or two sentences only, please.)

As \( x \) approaches zero the first equation approaches zero, but the second equation approaches \( \sin^2(x)/2 \) instead. The second equation is less affected by round-off error and is thus more likely to provide a more accurate value.
Problem 2) [10 marks]
A Maclaurin series can be used to calculate $e^x$:

$$e^x = 1 + \sum_{n=1}^{\infty} \frac{x^n}{n!} = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots$$

Write a method that uses the Kahan Summation Algorithm to produce a more accurate value for this sum. The principle behind the algorithm is that each time the sum is calculated the portion of the individual term that is not added to the sum is calculated and then returned back into the calculation of the next term. The method accepts a `double` value `x`, and returns the estimated value of $e^x$. Remember that you can use the `Math.pow` method to calculate $x^n$, as in `Math.pow(x, n)`.

```java
public static double exponentialKahan (double num) {
    double sum = 1;
    double prevSum;
    double factorial = 1;
    double term;
    int termNum = 1;

    double remainder = 0;
    double tempSum;

    do {
        prevSum = sum;
        factorial = factorial * termNum;
        term = Math.pow(num, termNum) / factorial + remainder;
        tempSum = sum + term;
        remainder = term - (tempSum - sum);
        sum = tempSum;
        termNum++;
    } while (sum != prevSum);

    return sum;
}
```

// end exponentialKahan
Problem 3) [30 marks]
For this problem you must write three classes using proper encapsulation:

- BPMeasurement – a class to hold blood pressure data.
- IllegalBP – an exception class thrown by BPMeasurement.
- BPCollection – holds a collection of BPMeasurement objects.

Here is a testing class and its output to show how BPCollection works:

```java
class TestBPCollection {
    public static void main(String[] args) {
        BPCollection bpData = new BPCollection();
        bpData.add(140, 70);
        bpData.add(135, 60);
        bpData.add(172, 80);
        bpData.add(130, 65);
        bpData.add(143, 72);
        bpData.add(160, 81);
        bpData.add(170, 90);
        bpData.add(240, 70); // an illegal one!
        bpData.add(140, 20); // another illegal one
        bpData.add(65, 120); // last illegal one
        bpData.add(120, 65);
        bpData.add(165, 72);

        System.out.println("Collection has " + bpData.getSize() + " elements: ");
        System.out.println(bpData);
        System.out.println("Minimum: " + bpData.getMin());
        System.out.println("Maximum: " + bpData.getMax());
        System.out.println("Average: " + bpData.getAverage());
    }
}
/*OUTPUT:*
Illegal BP measurement: 240 over 70
Illegal BP measurement: 140 over 20
Illegal BP measurement: 65 over 120
Collection has 9 elements:
140 over 70
135 over 60
172 over 80
130 over 65
143 over 72
160 over 81
170 over 90
120 over 65
165 over 72
Minimum: 120 over 65
Maximum: 172 over 80
Average: 148 over 72 */
Problem 3, Cont.)
A blood pressure measurement consists of two integer numbers – the systolic and the diastolic pressures in mmHg. The systolic pressure must be higher than the diastolic and the systolic must be less than 200 and the diastolic greater than 40 mmHg. For your BPMeasurement class write one constructor, both accessors, a toString, an equals and a compareTo method. The BPMeasurement class must implement the Comparable<BPMeasurement> interface, which will force it to have a compareTo method with the following signature:

    public int compareTo(BPMeasurement otherBP)

Compare BPMeasurement objects based first on just the systolic pressure. Use the diastolic pressure only if the systolic pressures are the same. Equality for the equals method is defined as both systolic and diastolic pressures being the same.

For your BPCollection class, you may use either an array or an ArrayList<BPMeasurement> object as the underlying data structure. To sort the underlying data structure, you may invoke Arrays.sort(array, startIndex, finishIndex) on the array, where the Arrays class is available from the java.util package. If you use an ArrayList instead, you may invoke Collections.sort(ArrayList). Both ArrayList and Collections are in the java.util package. The sort() methods sort the collection in situ, meaning that the array or ArrayList is passed by reference and is sorted in place. Do not write your own sorting code – that is not necessary.

Write your exception class below and the other two classes on the following two pages:

```java
public class IllegalBP extends Exception {

    public IllegalBP() {
        super("Illegal blood pressure measurement.");
    }

    public IllegalBP(String arg0) {
        super(arg0);
    }
}
```
public class BPMeasurement implements Comparable<BPMeasurement> {

    private int systolic;
    private int diastolic;

    public BPMeasurement (int sys, int dias) throws IllegalBP {
        if (sys < dias || sys > 200 || dias < 40)
            throw new IllegalBP("Illegal BP measurement: " + sys
                    + " over " + dias);

        systolic = sys;
        diastolic = dias;
    }

    public int getSystolic () { return systolic; }
    public int getDiastolic () { return diastolic; }

    public String toString () {
        return systolic + " over " + diastolic;
    }

    public boolean equals (Object o) {
        if (o instanceof BPMeasurement) {
            BPMeasurement otherBP = (BPMeasurement)o;
            return (otherBP.systolic == systolic) &&
                (otherBP.diastolic == diastolic);
        }
        return false;
    }

    public BPMeasurement clone () {
        BPMeasurement bpm = null;
        try {
            bpm = new BPMeasurement(systolic, diastolic);
        } catch (IllegalBP e) {}
        return bpm;
    }

    public int compareTo (BPMeasurement otherBP) {
        if (systolic != otherBP.systolic)
            return systolic - otherBP.systolic;
        else
            return diastolic - otherBP.diastolic;
    }
}

} // end BPMeasurement
import java.util.ArrayList;
import java.util.Collections;

public class BPCollection {

    private ArrayList<BPMeasurement> bpList = new ArrayList<BPMeasurement>();

    public void add (int systolic, int diastolic) {
        BPMeasurement bpm = null;
        try {
            bpm = new BPMeasurement(systolic, diastolic);
            bpList.add(bpm);
        } catch (IllegalBP e) {
            System.out.println(e.getMessage());
        }
    } // end add

    public BPMeasurement getMin () {
        Collections.sort(bpList);
        return bpList.get(0);
    } // end getMin

    public BPMeasurement getMax () {
        Collections.sort(bpList);
        return bpList.get(bpList.size() - 1);
    } // end getMax

    public int getSize () {
        return bpList.size();
    } // end getSize

    public BPMeasurement getAverage () {
        int sumSys = 0;
        int sumDias = 0;
        for (BPMeasurement bpm : bpList) {
            sumSys += bpm.getSystolic();
            sumDias += bpm.getDiastolic();
        }
        BPMeasurement bpAvg = null;
        try {
            bpAvg = new BPMeasurement (sumSys / getSize(), sumDias / getSize());
        } catch (IllegalBP e) {} 
        return bpAvg;
    } // end getAverage

    public String toString () {
        String output = "";
        for (BPMeasurement bpm : bpList)
            output += bpm + "\n";
        return output;
    } // end toString

} // end BPCollection
Problem 4) [10 marks]
For this problem you are supplied with three classes and an interface. You must write the fourth class, called MissingClass.

Here is the interface:

```java
public interface Specifications {
    int CONSTANT = 1000;
    int methodThree (int numOne, int numTwo);
} // end Specifications interface
```

Here are two hierarchy classes:

```java
public abstract class BaseClass {
    private int baseNum;

    public BaseClass (int baseNum) {
        this.baseNum = baseNum;
    }

    public int getBaseNum () {
        return baseNum;
    }

    public int methodOne (int numOne) {
        return numOne + baseNum;
    }

    public abstract int methodTwo (int numOne);
} // end BaseClass

public abstract class NextClass extends BaseClass {
    private int nextNum;

    public NextClass (int numOne, int numTwo) {
        super(numOne);
        nextNum = numTwo;
    }

    public int methodOne () {
        return getBaseNum() + nextNum;
    }
} // end NextClass
```
Problem 4, Cont.)
Here is a testing class whose output shows how MissingClass must operate:

```java
public class TestMissingClass {
    public static void main(String[] args) {
        MissingClass mc = new MissingClass(1, 20, 300);
        System.out.println(mc.methodOne(500));
        System.out.println(mc.methodTwo(100));
        System.out.println(mc.methodThree(200, 400));
    }
}
```

/* OUTPUT:
501
1301
621 */

Write the minimal amount of code to satisfy TestMissingClass. MissingClass must extend NextClass and implement Specifications and cannot be abstract. The only mathematical operation used in MissingClass is addition. You cannot use literal values in your expressions.

```java
public class MissingClass extends NextClass implements Specifications {
    private int missingNum;

    public MissingClass (int numOne, int numTwo, int numThree) {
        super(numOne, numTwo);
        missingNum = numThree;
    }

    public int methodTwo (int numOne) {
        return CONSTANT + getBaseNum() + missingNum;
    }

    public int methodThree (int numOne, int numTwo) {
        return methodOne() + numOne + numTwo;
    }
}
```
Problem 5) [5 marks]

The following GUI code contains an anonymous class:

```java
import javax.swing.JFrame;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.event.ActionListener; // an interface
import java.awt.event.ActionEvent;

public class ChangeMessageWindow extends JFrame{

    private JLabel message = new JLabel("First Message");

    public ChangeMessageWindow() {
        super();
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Change Message Window");

        add(message, BorderLayout.NORTH);

        JButton changeMessage = new JButton("Change Message");
        changeMessage.addActionListener(new ActionListener() {
            public void actionPerformed (ActionEvent e) {
                message.setText("Second Message");
            }
        });

        add(changeMessage, BorderLayout.SOUTH);
    } // end ChangeMessageWindow constructor
}
```

When the “Change Message” button is clicked the message label text changes from “First Message” to “Second Message”.

On the next page, change this program so that the anonymous class is replaced with an inner class.

Most of the old code is provided, so you don’t have to re-write it.
Problem 5, Cont.)  
Add your new code to the following, where indicated:

```java
import javax.swing.JFrame;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.event.ActionListener;  // an interface
import java.awt.event.ActionEvent;

public class ChangeMessageWindow extends JFrame{

    private JLabel message = new JLabel("First Message");

    public ChangeMessageWindow() {
        super();
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Change Message Window");
        add(message, BorderLayout.NORTH);
        JButton changeMessage = new JButton("Change Message");
        changeMessage.addActionListener(new ChangeListener());
        add(changeMessage, BorderLayout.SOUTH);
    } // end ChangeMessageWindow constructor

    private class ChangeListener implements ActionListener {
        public void actionPerformed (ActionEvent e) {
            message.setText("Second Message");
        }
    } // end ChangeListener

} // end ChangeMessageWindow
```
Problem 6) [5 marks]
Here is a complete GUI window definition:

```java
import javax.swing.JFrame;
import java.awt.FlowLayout;
import java.awt.BorderLayout;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JTextField;
import javax.swing.JLabel;

public class GUILayout extends JFrame {
    public GUILayout() {
        super();
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Exam Window");
        setResizable(false);
        setLayout(new BorderLayout());

        JPanel panel1 = new JPanel(new FlowLayout());
        JButton exitProgram = new JButton("Exit Program");
        panel1.add(exitProgram);

        JPanel panel2 = new JPanel(new BorderLayout());
        panel2.add(panel1, BorderLayout.EAST);
        JLabel label = new JLabel("A Label");
        panel3.add(label);
        panel2.add(panel3, BorderLayout.WEST);
        add(panel2, BorderLayout.SOUTH);

        JPanel panel4 = new JPanel(new FlowLayout());
        JLabel anotherLabel = new JLabel("Another Label");
        JTextField textField = new JTextField("Text Entry");
        panel4.add(anotherLabel);
        panel4.add(textField);
        add(panel4, BorderLayout.NORTH);

        JPanel panel5 = new JPanel(new FlowLayout());
        JButton anotherButton = new JButton("Another Button");
        add(panel5, BorderLayout.CENTER);
    }
}
```

In the empty frame on the next page, sketch what you think this window will look like when it is shown. Use the following key to draw your components:

- **A label:** A Label
- **A textfield:** Text Entry
- **A button:** Exit Program
Problem 6, Cont.)

Another Label  Text Entry
Another Button

A Label  Exit Program