Question 1: (20 points)

Part (a): (6 points)
Define “method signature.” Be concise. **Method signature is the method name and input parameter types (including the order), but excludes the return type**

Part (b): (11 points)
Consider class Q1b below. The specification and header for method `timeInSec` is shown but the method body is hidden. Assume that method `timeInSec` has been correctly implemented—it can be called from method `main`. The numbered statements are attempts to call method `timeInSec`. Write on the blank following each call the word “**good**” if the call is correctly written or the word “**bad**” if the call is incorrect.

```java
public class Q1b {
    /** = number of seconds from 00:00:00 to h:m:s. Return whole seconds only. */
    public static int timeInSec(int h, int m, double s) {
        // Code not shown. Assume method is implemented correctly.
        // ...
    }
}
```

```java
public static void main(String[] args){
    int h=20, m=58;  //hour, minute
double s=12.6;   //second
    int sec;
    sec= timeInSec(20, 58, 12.6);    //1. **good**
    sec= timeInSec(h, m, s);    //2. **good**
    sec= timeInSec(h, m, (int) s);    //3. **good**
    double d= timeInSec(h, m, (int) s);  //4. **good**
    sec= timeInSec(h, (int) Math.round(m + s/60)); //5. **bad**
    sec= timeInSec(new String(h+":"+m+":"+s)); //6. **bad**
    System.out.println(timeInSec(20, 58, s));  //7. **good**
    sec= Q1b.timeInSec(20, 58, s);    //8. **good**
    sec= class.timeInSec(20, 58, s);    //9. **bad**
    sec= static.timeInSec(20, 58, s);    //10. **bad**
    sec= this.timeInSec(20, 58, s);    //11. **bad**
}
```

Part (c): (3 points)
Our textbook (and Program Live) discusses a technique used in programming that was used also by Edgar Allan Poe in writing his poem *The Raven*. What is this technique?

**Top-Down design** or **iterative refinement**
Question 2: (20 points)

Consider the sequence

\[1, 2, -3, 4, 5, -6, 7, 8, -9, \ldots\]

Given \(n \geq 0\), write a program fragment to display the first \(n\) terms of the sequence in reverse order. Also display the sum of the sequence. For example, if \(n = 4\), the sequence displayed should be

\[4, -3, 2, 1\]

and the sum of the sequence is \(4 + (-3) + 2 + 1 = 4\). Do not use arrays.

```java
public class Q2 {
    public static void main(String[] args) {
        System.out.println("Enter a positive integer:");
        int n = JLiveRead.readInt();  // Number of terms in the sequence to print
        // Assume n is positive
        int sum;                      // Sum of the first n terms in the sequence
        num= n;
        sum= 0;
        int value;  // The value of the sequence to print and sum
        for ( ; num > 0 ; num-- ) {
            if ( num%3==0 )
                value= -num;
            else
                value= num;
            System.out.println(value);
            sum += value;
        }
        System.out.println("The sum of the first " + n + " terms is " + sum);
    }
}
```
Question 3: (40 points)

Complete classes Rectangle and Q3 below. Class Rectangle represents a rectangle and has the following variables and methods:

- Instance variables width and height: the width and height (type double) of a Rectangle
- A constructor that has two parameters: double width, double h
- Instance method area() returns the area (type double) of the current Rectangle
- Instance method isSquare() returns true if the current Rectangle is a square, false otherwise
- Instance method cutHalf() cuts off vertically half of the current Rectangle
- Instance method toString() gives a String description of the dimensions (width and height) and area of the current Rectangle
- Class method average(Rectangle r1, Rectangle r2) returns a new Rectangle with dimensions that are the average values between Rectangles r1 and r2
- Do not define any other instance or class variables/methods

Class Q3 is a client class of Rectangle. Class Q3 has a single method main where you will create one Rectangle. Then you will repeatedly cut the Rectangle in half until it becomes a square or its area is less than a specified value. Print the information of the final Rectangle.

Read through both incomplete classes before you start writing. Follow the specifications above and in the comments. You must use the variable and parameter names and types as specified above. Use encapsulation (use the modifiers private and public appropriately). To indicate that a blank (or box) should be left empty, draw a diagonal line across the blank or box.

```java
/** A rectangle */
class Rectangle {

    private double width;    //width of the Rectangle
    private double height;   //height of the Rectangle

    /** Constructor: assign values to the fields */
    public Rectangle (double width , double h ){
        this.width= width;
        height= h;
    }

    /** = Get area of this Rectangle */
    public double area() {
        return width * height;
    }

    //Class Rectangle continues on next page
}
```
public boolean isSquare() {
    return width==height;
}

public void cutHalf() {
    width /= 2;
}

public String toString() {
    return height + "-by-" + width + 
        " rectangle has area " + area();
}

public static Rectangle average(Rectangle r1, Rectangle r2) {
    return new Rectangle( (r1.width+r2.width)/2, 
        (r1.height+r2.height)/2 );
}
/* Class Q3, client of class Rectangle (Question 3, continued) */
public class Q3 {
    public static void main(String[] args) {

        // Create a Rectangle object, use reference variable rec:
        double w = JLiveRead.readDouble();  // width of rec
        double h = JLiveRead.readDouble();  // height of rec

        Rectangle rec = new Rectangle(w, h);

        // Repeatedly cut rec in half until it becomes a square OR until its area
        // is less than MINarea. Display Rectangle rec's data at the end.
        final double MINarea = 10;

        for (;
            !rec.isSquare() && rec.area() >= MINarea;
            rec.cutHalf() ) ;

        System.out.println(rec);

        // Above, need to have ; or {} after for-loop header

        /* Another CORRECT loop condition:
        ! ( rec.isSquare() || rec.area() < MINarea )

        A WRONG loop condition:
        ! rec.isSquare() || rec.area() >= MINarea
        */

    } // method main
} // class Q3
Question 4: (20 points)

Typically, a student has a bursar account representing the amount that she or he owes the university. Design a class `Account` whose instances represent students’ bursar accounts. An account is associated with a student name and a student ID and has a balance. It should be possible to retrieve these values. Furthermore, it should be possible to charge to (increase balance of) and make payment to (decrease balance of) the account. It should be possible to determine if the account has an owing balance so that a statement can be printed (showing the student name, ID, and balance).

Design the class by writing variable declarations and method specifications and headers. Use meaningful variable and method names. Specifications (comments) must be concise. Do not write the method bodies!

```java
/** A student’s bursar account */
class Account {
    private String name; // Student name
    private String id; // Student ID (type int is ok)
    private double balance; // Account balance

    public Account(String name, String id, double initBalance) {
        // Constructor
    }

    /** Getter methods */
    public String getName() {
        // Get name
    }
    public String getID() {
        // Get ID
    }
    public double getBalance() {
        // Get balance
    }

    /** Charge an amount on this Account */
    public void charge(double amount) {
        // Charge amount
    }

    /** Make a payment (amount) to this Account */
    public void paymentMade(double amount) {
        // Make payment
    }

    /** = This Account has an owing balance */
    public boolean hasOwingBalance() {
        // Check for owing balance
    }

    /** = String description of Account data */
    public String toString() {
        // Return string representation
    }
}
```

- One must define instance variables using comments.
- Most methods should have specifications (comments) above the header.
- One may choose to have a method for printing instead of a `toString` method.
- One may write just one method to determine owing balance and to print (not `toString`), but the specification must be very clear about all its functionality.
- One may choose to have an instance variable `isOwing` instead of an instance method `hasOwingBalance`.
- The only methods that need parameters are the constructor, `charge`, and `paymentMade`.
- If the constructor doesn’t have parameters, then there must be setter methods.
- The constructor does not have to have a parameter for initial balance, assuming that all accounts open with 0 balance.
- Methods `charge` and `paymentMade` do not need to return anything, but if one makes them return a value, the value must be clearly specified (to explain why).
- Methods `charge` and `paymentMade` may be combined as one method, but the specification then must clearly talk about positive vs. negative parameter values.