BRAC University

Department of Computer Science and Engineering

CSE111 (Programming Language-II)

Task 1

```
class A{
  public static int temp = 1;
  public int sum = 1;
  public int y = 1;
  public A(){
    y = temp - 1;
    sum = temp + 1;
    temp-= 1;
  }
  public void methodA(int m, int n){
    int x = 1;
    y = y + m + (temp++);
    x = x + 1 +  n;
    sum = sum + x + y;
    System.out.println(x + " " + y+ " " + sum);
  }
}
class B extends A {
  public static int x = 1;
  public B(){
    y = temp + 1 ;
    x = 1 + temp + x;
    temp -= 1;
  }
  public B(B b){
    sum = b.sum + sum;
    b.x = b.x + sum;
  }
  public void methodB(int m, int n){
    int  y = 1, temp = 1;
    y = y + this.y + m;
    x = y + 1 + temp - n;
    methodA(x, y);
    sum = x + y + sum;
    System.out.println(x + " " + y+ " " + sum);
  }
}
```

What is the output of the following code sequence?

```
A  a1 = new A();
B  b1 = new B(); B b2 =
   new B(b1);
   a1.methodA(1, 1);
   b1.methodA(1, 2);
   b2.methodB(3, 2);
```

x	y	sum

Task 2

```
class A{
    public static int temp = 1;
    public int sum = 1;
    public int y = 1;
    public A(){
        y = temp - 1;
        sum = temp + 1;
        temp-= 1;
    }
    public void methodA(int m, int n){
        int x = 1;
        y = y + m + (temp++);
        x = x + 1 + n;
        sum = sum + x + y;
        System.out.println(x + " " + y+ " " + sum);
    }
}
class B extends A {
    public static int x = 1;
    public int sum = 1;
    public int y = 1;
    public B(){
        y = temp + 1 ;
        x = 1 + temp + x;
        temp -= 1;
    }
    public B(B b){
        sum = b.sum + this.sum;
        b.x = b.x + super.sum;
    }
    public void methodB(int m, int n){
        int y = 1, temp = 1;
        y = y + this.y + m;
        x = y + 1 + temp - n;
        methodA(x, super.y);
        sum = x + y + sum;
        System.out.println(x + " " + this.y+ " " + sum);
    }
}
```

What is the output of the following code sequence?

```
A  A a1 = new A();
B  B b1 = new B();
C  B b2 = new B(b1);
   a1.methodA(1, 1);
   b1.methodA(1, 2);
   b2.methodB(3, 2);
```

x	y	sum

Task 3

```
class A{
    public static int temp = 1;
    public int sum = 1;
    public int y = 1;
    public A(){
        y = temp - 1;
        sum = temp + 1;
        temp-= 1;
    }
    public A(int y){
        this.y = temp - y;
        this.sum = temp + 1;
        temp *= 4;
    }
    public void methodA(int m, int n){
        int x = 1;
        y = y + m + (temp++);
        x = x + 1 + n;
        sum = sum + x + y;
        System.out.println(x + " " + y+ " " + sum);
    }
}
class B extends A {
    public static int x = 1;
    public int sum = 1;
    public int y = 1;
    public B(){
        y = temp + 1 ;
        x = 1 + temp + x;
        temp -= 1;
    }
    public B(B b){
        super(A.temp);
        sum = b.sum + this.sum;
        b.x = b.x + super.sum;
    }
    public void methodA(int m, int n){
        int y = 1, temp = 1;
        y = y + this.y + m;
        x = y + 1 + temp - n;
        super.methodA(x, super.y);
        sum = x + y + sum;
        System.out.println(x + " " + this.y+ " " + sum);
    }
}
```

What is the output of the following code sequence? **[Answer on question paper]**

```
A  a1 = new A();
B  b1 = new B(); B b2 =
   new B(b1);
   a1.methodA(1, 1);
   b1.methodA(1, 2);
   b2.methodB(3, 2);
```

x	y	sum

Task 4

Design a "Vehicle" class. A vehicle assumes that the whole world is a 2 dimensional graph paper. It maintains its x and y coordinates (both are integers). The vehicle gets manufactured (constructed?) at (0,0) coordinate.
Write a user class called "Vehicle". It must have methods to move up, down, left, right and a toString method for printing current coordinate.
Note: All moves are 1 step. That means a single call to any move method changes value of either x or y or both by 1.
Take help from:
http://www.javabeginner.com/learn-java/java-tostring-method
http://cscie160-distance.com/toString.html

```
public class VehicleUser{
    public static void main(String[] args){
        Vehicle car = new Vehicle();
System.out.println(car.toString());
car.moveUp();
System.out.println(car.toString());
car.moveLeft();
System.out.println(car.toString());
car.moveDown();
System.out.println(car.toString());
car.moveRight();
// see, output for following two lines are same because toString() is
automatically called. So, you can omit toString when printing.
System.out.println(car.toString());
System.out.println(car);
    }
}
```

Expected Output:

```
(0, 0)
(0, 1)
(-1, 1)
(-1, 0)
(0, 0)
(0, 0)
```

Task 5

Design a Vehicle2010 class which inherits movement methods from Task1 and adds new methods called move UpperRight, UpperLeft, LowerRight, LowerLeft. Each of these diagonal move methods must re use two inherited and appropriate move methods. Write user class as well which will show that all of your methods are working. A small user class is shown below as an example.

Note: All moves are 1 step. That means a single call to any move method changes value of either x or y or both by 1.

Additionally, you have to write an "**equals**" method which tests if significant class properties are same (in this case x and y).

Take help on "equals" method from:

- http://www.ibiblio.org/java/course/week4/37.html
- http://www.javaworld.com/javaworld/jw-06-2004/jw-0614-equals.html
- http://www.artima.com/lejava/articles/equality.html

```
public class Vehicle2010User{
    public static void main(String[] args){
        Vehicle2010 car = new Vehicle2010();
System.out.println(car);
car.moveLowerLeft();
System.out.println(car);

        Vehicle2010 car2 = new Vehicle2010();
        car2.moveLeft();
System.out.println(car.equals(car2));
        car2.moveDown();
System.out.println(car.equals(car2));
    }
}
```

Expected Output:

```
(0, 0)
(-1, -1)
false
true
```

Task 6

Create a class **SavingsAccount,** which will use a **static** class variable to store the **annualInterestRate** for all account holders.

- Each object of the class contains a **private** instance variable **savingsBalance** indicating the amount the saver currently has on deposit.
- Provide method **calculateMonthlyInterest()** to calculate the monthly interest [by multiplying the **savingsBalance** by **annualInterestRate** divided by 12], this interest should be added to **savingsBalance**.
- Provide a **static** method **modifyInterestRate()** that sets the **annualInterestRate** to a new value.

Write a driver program to test class **SavingsAccount**.

- Instantiate two **SavingsAccount** objects, **saver1** and **saver2**, with balances $20000.00 and $30000.00, respectively using constructor.
- Set **annualInterestRate** to 4.2%, then calculate the monthly interest and print the new balances for each of the savers using **printSavingsBalance( )** method.

  *Then set the **annualInterestRate** to 5.5% and calculate the next month's interest and .print the new balances for each of the savers*

Task 7

Assume that a bank maintains two kinds of accounts for its customers, one called savings account and the other current account. The savings account provides compound interest (5%) and withdrawal facilities. The current account provides no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge in imposed.

Create a class **Account** that stores customer name, account number, type of account and balance. From this **Account** class derive the classes **CurrentAccount** and **SavingsAccount** to make them more specific to their requirements. Include the necessary methods in order to achieve the following tasks:

- Initialize all instance variables through constructors. [use super()]
- Accept deposit from a customer & update the balance [**depAmount** (...)]
- Display the balance [**showBalance** ( )]
- Compute and deposit interest [**computeInterest** ( )]
- Permit withdrawal and update the balance [**withdraw** (...)]
- Check for the minimum balance (assume $500), impose penalty (print a message, if necessary) and restrict the withdrawal of balance.

[Use only methods to initialize the class members and other tasks.]

Take Help from:

- **http://www.javaworld.com/cgi-bin/mailto/x_java.cgi?pagetosend=/export/home/httpd/javaworld/jw-10-2000/jw-1013-constructors.html&pagename=/jw-10-2000/jw-1013-constructors.html&pageurl=http:// www.javaworld.com/jw-10-2000/jw-1013-constructors.html&site=jw_core**

Task 8

Write a java program, which calculates "the area of a circle" and "the volume of a sphere" by overriding the space() method in the subclass. [Extend the following super class 'Point' with necessary overriding of specific method. DO NOT CHANGE THE POINT CLASS].

```
class Point {
        private double radius;
        Point ( double r) {
                radius = r;
        }
        double space ( ) {
                System.out.println("Space for a Point can't be defined");
                return 0;
        }
        protected double getRadius(){
                return radius;
        }
}
// create new sub class
```

Implement the above program to include constructors for all the subclasses & use 'super( )' to call super-class constructors, if necessary.

Sample Input/Output

```
Enter radius of Circle: 5
Creating a Circle … done!
The area of the Circle is 78.539816339744830961566084581988

Enter radius of Sphere: 7
Creating a Sphere … done!
The area of the Sphere is 205.25072003453315824622603437426
```

Task 9

Write an abstract class ‘**Instrument**’ which will have abstract method ‘**play**’, ‘**adjust**’& concrete method ‘**compose**’.

Use the abstract class ‘Instrument’ to create class ‘**Guitar**’, ‘**Keyboard**’& ‘**Violin**’.

Create instance (object) of every classes invoking(calling) every method. The method will print any message with ‘Instrument name’ and ‘Purpose’,

- The method ‘play’ for ‘Violin’ class will print “In the playing method of Violin”

Task 10

Write a java application as follows:

Create a Student Interface ‘**StInterface**’ with the methods ‘**setName**’, ‘**setID**’, ‘**getName**’**and** ‘**getID**’.

Create the class ‘**Student**’ with ‘name’, ‘id’ and ‘address field’ implementing the ‘**StInterface**’ to manipulate the Student information using the necessary methods.

Create an array of objects of **Student**. Then input the number of students to allocate student array dynamically and take Student information. Now print the student list alphabetically.

Task 11

Mutant Flatworld Explorers
http://online-judge.uva.es/p/v1/118.html
Hint: It is similar to Task 1 and 2 but here you will have to maintain a two dimensional character array

Background

Robotics, robot motion planning, and machine learning are areas that cross the boundaries of many of the subdisciplines that comprise Computer Science: artificial intelligence, algorithms and complexity, electrical and mechanical engineering to name a few. In addition, robots as ``turtles'' (inspired by work by Papert, Abelson, and diSessa) and as ``beeper-pickers'' (inspired by work by Pattis) have been studied and used by students as an introduction to programming for many years.

This problem involves determining the position of a robot exploring a pre-Columbian flat world.

The Problem

Given the dimensions of a rectangular grid and a sequence of robot positions and instructions, you are to write a program that determines for each sequence of robot positions and instructions the final position of the robot.

A robot *position* consists of a grid coordinate (a pair of integers: x-coordinate followed by y-coordinate) and an orientation (N,S,E,W for north, south, east, and west). A robot *instruction* is a string of the letters '*L*', '*R*', and '*F*' which represent, respectively, the instructions:

- *Left*: the robot turns left 90 degrees and remains on the current grid point.
- *Right*: the robot turns right 90 degrees and remains on the current grid point.
- *Forward*: the robot moves forward one grid point in the direction of the current orientation and mantains the same orientation.

The direction *North* corresponds to the direction from grid point (*x*,*y*) to grid point (*x*,*y*+1). Since the grid is rectangular and bounded, a robot that moves ``off'' an edge of the grid is lost forever. However, lost robots leave a robot ``scent'' that prohibits future robots from dropping off the world at the same grid point. The scent is left at the last grid position the robot occupied before disappearing over the edge. An instruction to move ``off'' the world from a grid point from which a robot has been previously lost is simply ignored by the current robot.

Hint: For your convenience, you may mark cells having the scent with 'X' or any character you like to mean forbidden cells.

The Input

The first line of input is the upper-right coordinates of the rectangular world, the lower-left coordinates are assumed to be 0,0.

The remaining input consists of a sequence of robot positions and instructions (two lines per robot). A position consists of two integers specifying the initial coordinates of the robot and an orientation (N,S,E,W), all separated by white space on one line. A robot instruction is a string of the letters 'L', 'R', and 'F' on one line.

Each robot is processed sequentially, i.e., finishes executing the robot instructions before the next robot begins execution.

Input is terminated by end-of-file.

You may assume that all initial robot positions are within the bounds of the specified grid. The maximum value for any coordinate is 50. All instruction strings will be less than 100 characters in length.

The Output

For each robot position/instruction in the input, the output should indicate the final grid position and orientation of the robot. If a robot falls off the edge of the grid the word ``LOST" should be printed after the position and orientation.

Sample Input

```
5 3
1 1 E
RFRFRFRF
3 2 N
FRRFLLFFRRFLL
0 3 W
LLFFFLFLFL
```

Sample Output

```
1 1 E
3 3 N LOST
2 3 S
```