Task 1

```
1.  class A{
2.    public int temp = 4;
3.    public int sum = 1;
4.    public int y = 2;
5.    public A(){
6.      y = temp - 2;
7.      sum = temp + 3;
8.      temp-=2;
9.    }
10.   public void methodA(int m, int n){
11.     int x = 0;
12.     y = y + m + (temp++);
13.     x = x + 2 +  n;
14.     sum = sum + x + y;
15.     System.out.println(x + " " + y+ " " + sum);
16.   }
17. }
18. class B extends A {
19.   public int x = 1;
20.   public int sum = 2;
21.   public B(){
22.     y = temp + 3 ;
23.     sum = 3 + temp + 2;
24.     temp-=1;
25.   }
26.   public B(B b){
27.     sum = b.sum;
28.     x = b.x;
29.   }
30.   public void methodB(int m, int n){
31.     int  y =0;
32.     y = y + this.y;
33.     x = this.y + 2 + temp;
34.     methodA(x, y);
35.     sum = x + y + super.sum;
36.     System.out.println(x + " " + y+ " " + sum);
37.   }
38. }
```

Consider the following code: [Answer on the 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 2

Given the following classes, write the code for the **BBAStudent** class so that the following output is printed when we run the **TestStudent** class.

```
Name : Default BBA Student Department: BBA

Name : Humty Dumty Department: BBA

Name : Little Bo Peep Department: BBA
```

```
public class Student{
  private String name = "Just a Student";
  private String department = "nothing";
  public void setDepartment(String dpt){
    this.department = dpt;
  }
  protected String getName(){
    return name;
  }
  protected void setName(String name){
    this.name = name;
  }

  public String toString(){
    return "Name : " + name + " Department: " + department;
  }
}
public class TestStudent{
  public static void printName(Student s){
    System.out.println(s.toString());
  }
  public static void main(String [] args){
    printName(new BBAStudent());
    printName(new BBAStudent("Humty Dumty"));
    printName(new BBAStudent("Little Bo Peep"));
  }
}
```

Task 3 **[??????????????????????????????????????]**

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 205.250720034533158246226034374 26

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

Write the **ComplexNumber** class so that the following code generates the output below:

```
public class Tester {
    public static void main(String[] args) {
        RealNumber rn = new ComplexNumber();
        System.out.println(rn);

        System.out.println("--------------------");

        rn = new ComplexNumber(5, 7);
        System.out.println(rn);

        System.out.println("--------------------");
        ComplexNumber cn = new ComplexNumber();
        cn.check();
    }
}
public class RealNumber {
    private double realValue;
    public double getRealValue() {
        return realValue;
    }
    public void setRealValue(double r) {
        realValue = r;
    }
    public RealNumber() {
        this(0);
```

```
    }
    public RealNumber(double r) {
        setRealValue(r);
    }
    public String toString() {
        return "RealPart: "+getRealValue();
    }
    public void ping() {
        System.out.println("I'm in RealNumber class");
    }
}
```

```
RealPart: 1.0
ImaginaryPart: 1.0
--------------------
RealPart: 5.0
ImaginaryPart: 7.0
--------------------
I'm in ComplexNumber class
I'm in RealNumber class
Checking ended.
```

Task 7

Write the **Mango** and the **Jackfruit** classes so that the following code generates the output below:

```
public class Test{
    public static void testFruit(Fruit f){
```

```
        System.out.println("----Printing Detail-----");
        if(f.hasFormalin()){
            System.out.println("Do not eat the "+f.getName()+".");
            System.out.println(f);
        }else{
            System.out.println("Eat the "+f.getName()+".");
            System.out.println(f);
        }
    }
    public static void main(String [] args){
        Mango m = new Mango();
        testFruit(m);
        Jackfruit j = new Jackfruit();
        testFruit(j);
    }
}
public class Fruit{
    private boolean formalin = false;
    public String name = "";
    public Fruit(boolean formalin, String name){
        this.formalin = formalin;
        this.name = name;
    }
    public String getName(){
        return name;
    }
    public boolean hasFormalin(){
```

```
        return formalin;
    }
}
```

```
----Printing Detail-----

Do not eat the Mango.

Mangos are bad for you

----Printing Detail-----

Eat the Jackfruit.

Jackfruits are good for you
```

Task 8

Write the CheckingAccount class so that the following code generates the output below

```
public class Account{
  protected double balance = 0.0;
  public Account(double balance){
    this.balance = balance;
  }
  public double getBalance(){
    return balance;
  }
}
public class TestAccount{
  public static void printBalance(Account a){
    System.out.println("Account Balance: " + a.getBalance());
  }
  public static void main(String [] args)
```

```
  {
    System.out.println("Number of Checking Accounts: " +
CheckingAccount.numberOfAccount);
    printBalance(new CheckingAccount());
    printBalance(new CheckingAccount(100.00));
    printBalance(new CheckingAccount(200.00));
    System.out.println("Number of Checking Accounts: " +
CheckingAccount.numberOfAccount);
  }
}
```

Number of Checking Accounts: 0

Account Balance: 0.0

Account Balance: 100.0

Account Balance: 200.0

Number of Checking Accounts: 3

Task 9

Write the **CSEStudent** and **CSE111Student** class so that the following code generates the output below **[**

```
public class Student{
  public String msg = "I love BU";
  public String shout(){
    return msg;
  }
}
```

```
public class TestStudent{
  public static void printShout(Student s){
    System.out.println("------------------");
    System.out.println(s.msg);
    System.out.println(s.shout());
  }
  public static void main(String [] args){
    Student s = new Student();
    CSEStudent cs = new CSEStudent();
    CSE111Student cs111 = new CSE111Student();
    System.out.println(s.msg);
    System.out.println(cs.msg);
    System.out.println(cs111.msg);
    printShout(s);
    printShout(cs);
    printShout(cs111);
  }
}
```

Output

```
I love BU

I want to transfer to CSE

I love Java Programming

------------------

I love BU

I love BU
```

```
------------------

I love BU

I want to transfer to CSE

------------------

I love BU

I love Java Programming
```

Task 10

Write the Car class so that the following code generates the output bellow

```
public class TestCars{

  public static void printCarDetail(Car c){

    System.out.println("Year: "+ c.getYear());

    System.out.println("Total Number of Cars: "+ c.getObjectCount());

  }

  public static void main(String [] args){

    System.out.println("Total Number of Cars: "+
Car.getObjectCount());

    System.out.println("==============================");

    Car c1 = new Car(2000);

    printCarDetail(c1);

    Car c2 = new Car(2006);

    printCarDetail(c2);

    Car c3 = new Car(2002);

    printCarDetail(c3);

    System.out.println("==============================");
```

```
    System.out.println("Total Number of Cars: "+
Car.getObjectCount());

  }

}
```

Total Number of Cars: 0

==========

Year: 2000

Total Number of Cars: 1

Year: 2006

Total Number of Cars: 2

Year: 2002

Total Number of Cars: 3

==========

Total Number of Cars: 3

Task 11

Given the following classes, write the code for the **Dog** and the **Cat** class so that the following output is printed when we run the **AnimalDriver** class.

```
Animal do not make sound

meow

bark
```

```
public class Animal {


  //Name of the Animal
```

```
  private String sound = "Animal Sound";

  //Default Constructor
  public Animal(){
  }

  //Overloaded Constructor
  Animal(String _sound){
    this.sound = _sound;
  }

  //Return sound
  public String makeSound(){
    return sound;
  }
}

public class AnimaDriver{

  public static void printSound(Animal a){
    System.out.println(a.makeSound());
  }

  public static void main(String [] args){
    Dog d1 = new Dog("bark");
    Cat c1 = new Cat("meow");
    Animal a1 = new Animal("Animal do not make sound");
```

```
    printSound(a1);

    printSound(c1);

    printSound(d1);

  }

}
```

Task 12

```
class A{
  public static int temp = 4;
  public int sum;
  public int y;
  public A(){
    y = temp - 2;
    sum = temp + 1;
    temp-=2;
  }
  public void methodA(int m, int n){
    int x = 0;
    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;
  public int sum;
  public B(){
    y = temp + 3 ;
    sum = 3 + temp + 2;
    temp-=2;
  }
  public B(B b){
    sum = b.sum;
    x = b.x;
    b.methodB(2,3);
  }
  public void methodB(int m, int n){
    int  y = 0;
    y = y + this.y;
    x = this.y + 2 + temp;
    methodA(x, y);
    sum = x + y + sum;
    System.out.println(x + " " + y+ " " + sum);
  }
}
```

Consider the following code:

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

x	y	sum

Task 13

```
class A{
  public int temp = 4;
  public int sum;
  public int y;
  public A(){
    y = temp - 2;
    sum = temp + 3;
    temp-=2;
  }
  public void methodA(int m, int n){
    int x = 0;
    y = y + m + (temp++);
    x = x + 2 +  n;
    sum = sum + x + y;
    System.out.println(x + " " + y+ " " + sum);
  }
}
class B extends A {
  public int x;
  public int sum;
  public B(){
    y = temp + 3 ;
    sum = 3 + temp + 2;
    temp-=1;
  }
  public B(B b){
    sum = b.sum;
    x = b.x;
  }
  public void methodB(int m, int n){
    int  y =0;
    y = y + this.y;
    x = this.y + 2 + temp;
    methodA(x, y);
    sum = x + y + sum;
    System.out.println(x + " " + y+ " " + sum);
  }
}
```

Consider the following code:

```
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 14

```
1  class A{
2    public int temp = 4;
3    public int sum = 1;
4    public int y = 2;
5    public A(){
6      y = temp - 2;
7      sum = temp + 3;
8      temp-=2;
9    }
10   public void methodA(int m, int n){
11     int x = 0;
12     y = y + m + (temp++);
13     x = x + 2 +  n;
14     sum = sum + x + y;
15     System.out.println(x + " " + y+ " " + sum);
16   }
17 }
18 class B extends A {
19   public int x = 1;
20   public int sum = 2;
21   public B(){
22     y = temp + 3 ;
23     sum = 3 + temp + 2;
24     temp-=1;
25   }
26   public B(B b){
27     sum = b.sum;
28     x = b.x;
29   }
30   public void methodB(int m, int n){
31     int  y =0;
32     y = y + this.y;
33     x = this.y + 2 + temp;
34     methodA(x, y);
35     sum = x + y + super.sum;
36     System.out.println(x + " " + y+ " " + sum);
37   }
38 }
```

Consider the following code:

```
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 15

```
class A{
    public static int temp = 3;
    public int sum;
    public int y;
    public A(){
        y = temp - 1;
        sum = temp + 2;
        temp-=2;
    }
    public void methodA(int m, int [] n){
        int x = 0;
        y = y + m + (temp++);
        x = x + 2 +  (++n[0]);
        sum = sum + x + y;
        n[0] = sum + 2;
        System.out.println(x + " " + y+ " " + sum);
    }
}
class B extends A {
    public static int x = 1;
    public int sum = 2;
    public B(){
        y = temp + 1 ;
        x = 3 + temp + x;
        temp-=2;
    }
    public B(B b){
        sum = b.sum + super.sum;
        x = b.x + x;
    }
    public void methodB(int m, int n){
        int [] y = {0};
        super.y = y[0] + this.y + m;
        x = super.y + 2 + temp - n;
        methodA(x, y);
        sum = x + y[0] + super.sum;
        System.out.println(x + " " + y[0]+ " " + sum);
    }
}
```

Consider the following code:

```
int x[] = {23};
A a1 = new A();
B b1 = new B();
B b2 = new B(b1);
```


`a1.methodA(1, x);`
`b2.methodB(3, 2);`
`a1.methodA(1, x);`			

Task 16

```
public class FinalT6A{
  public static int temp = 4;
  private int sum;
  private int y = 1;
  public FinalT6A(int x, int p){
    temp+=1;
    y = temp - p;
    sum = temp + x;
    System.out.println(x + " " + y+ " " + sum);
  }
  public void methodA(){
    int x=0, y =0;
    y = y + this.y;
    x = this.y + 2 + temp;
    sum = x + y + methodB(temp, y);
    System.out.println(x + " " + y+ " " + sum);
  }
  public int methodB(int temp, int n){
    int x = 0;
    y = y + (++temp);
    x = x + 3 +  n;
    sum = sum + x + y;
    System.out.println(x + " " + y+ " " + sum);
    return sum;
  }
}
```

What is the output of the following code sequence?

`FinalT6A q1 = new FinalT6A(2,1);`
`q1.methodA();`
`q1.methodA();`
	x	y	sum

Task 17

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
```