**Robot**

**We need a Robot class that can move around on a two dimensional plane. It needs to be able to change its position, report its position and report its last move as described below. Implement a Robot class per the following specifications:**

|  |  |  |
| --- | --- | --- |
| **Fields** | | |
| **Data Type** | **Name** | **Description** |
| integer | currentX | The robot’s current x-coordinate in the 2D plane |
| integer | currentY | The robot’s current y-coordinate in the 2D plane |
| integer | previousX | The robot’s current x-coordinate in the 2D plane prior to its most recent movement |
| integer | previousY | The robot’s current x-coordinate in the 2D plane prior to its most recent movement |
| **Note:** The robot’s initial location is at (x,y) coordinate (0,5) | | |

|  |  |  |
| --- | --- | --- |
| **Parameterized Constructor** | | |
| **Data Type** | **Param. Name** | **Description** |
| integer | x | The value of currentX for the new Robot |
| integer | y | The value currentY for the new Robot |
| Note: The robot created by this constructor is considered to have spawned at (0,5) and moved to (currentX, currentY) so (previousX, previousY) starts as (0,5) | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Methods** | | | | |
| **Return Type** | **Method Name** | **Param. Type** | **Param. Name** | **Description** |
| void | moveX | integer | dx | Move the robot from current position (x,y) to new position (x +dx, y). Remember to maintain previousX |
| void | moveY | integer | dy | Move the robot from current position (x,y) to new position (x, y + dy). Remember to maintain previousY. |
| void | printCurrentCoordinates | no parameters | | Print two space-separated integers describing the robot’s current x and y coordinates. |
| void | printLastCoordinates | no parameters | | Print two space separated integers describing the robot’s previousX and previousY coordinates. This will be called after the robot has moved from position (0,5) at least once. |
| void | printLastMove | no parameters | | Print two space-separated values describing the robot’s most recent movement:    - If the last move was moveX(dx), print x dx where x i the actual character x and dx is the distance moved in the x-direction during last call to moveX    - If the last move was moveY(dy), print y dy where y is the actual character y and dy is the distance moved in the y-direction during last call to moveY |

**class Solution {**

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

**int x = 1;**

**int y = 5;**

**int dx = 1;**

**int dy = 1;**

**Robot firstRobot = new Robot();**

**firstRobot.printCurrentCoordinates();**

**Robot secondRobot = new Robot(x, y);**

**secondRobot.printCurrentCoordinates();**

**for(int i = 1; i <3; i++) {**

**secondRobot.moveX(dx);**

**secondRobot.printLastMove();**

**secondRobot.printCurrentCoordinates();**

**secondRobot.moveY(dy);**

**secondRobot.printLastCoordinates();**

**dx += i \* i;**

**dy -= i \* i;**

**}**

**}**

**}**

class Robot {

private int currentX;

private int currentY;

private int previousX;

private int previousY;

private boolean lastMoveX;

private boolean hasMoved;

public Robot() {

this.currentX = 0;

this.currentY = 5;

this.previousX = 0;

this.previousY = 5;

this.hasMoved = false;

}

public Robot(int x, int y) {

this.previousX = 0;

this.previousY = 5;

this.currentX = x;

this.currentY = y;

this.hasMoved = true;

}

public void moveX(int dx) {

this.previousX = this.currentX;

this.currentX += dx;

this.lastMoveX = true;

this.hasMoved = true;

}

public void moveY(int dy) {

this.previousY = this.currentY;

this.currentY += dy;

this.lastMoveX = false;

this.hasMoved = true;

}

public void printCurrentCoordinates() {

System.out.println(this.currentX + " " + this.currentY);

}

public void printLastCoordinates() {

if(this.hasMoved){

if(this.lastMoveX) {

System.out.println(this.previousX + " " + this.currentY);

} else {

System.out.println(this.currentX + " " + this.previousY);

}

}

}

public void printLastMove() {

if(this.lastMoveX) {

System.out.println("x " + (this.currentX - this.previousX));

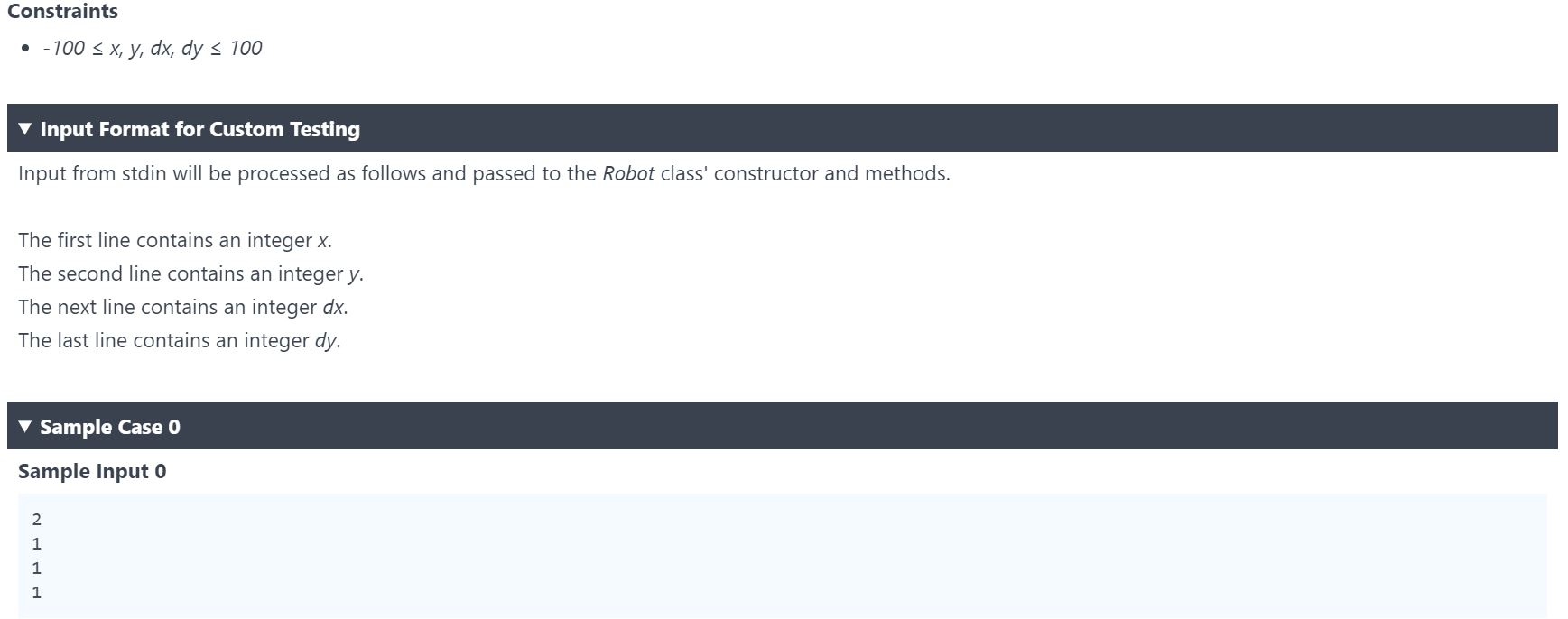
} else {

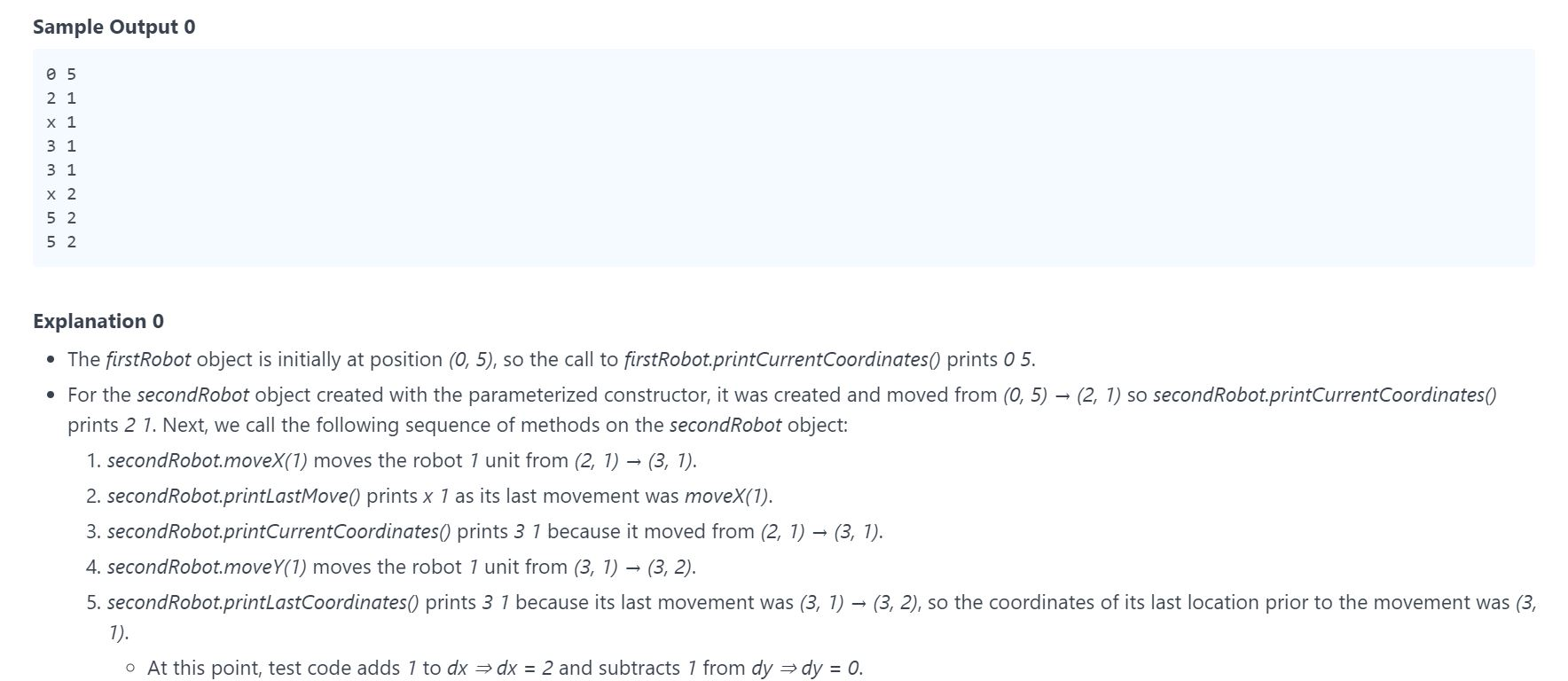
System.out.println("y " + (this.currentY - this.previousY));

}

}

}





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

**Alumns**

**There are a number of students standing in a single-file line. Each student is numbered sequentially from 0. Each student also has a binary digit associated with them, where 0 indicates the student’s performance is above average and 1 indicates performance is below average.**

**The teacher wants to group the students by performance on opposite ends of the line such that the number of adjacent pairs of students where one student is a 0 and the other student is a 1 is minimized. To accomplish this, any student can swap places in line with the student located immediately in front of or behind them. Each time a pair of students swap places, it counts as a move. Determine the minimum of moves needed to create an optimal configuration.**

**For example, there are n=4 students arranged as avg = [0,1,0,1]. With 1 move, switching students 1 and 2, we get the array [0,0,1,1] which is optimal.**

**Functional Description**

**Complete the function minMoves in the editor below. The function must return an integer denoting the minimum number of moves necessary to achieve an optimal configuration.**

**minMoves has the following parameter(s):**

**avg[avg[0], …. avg[n-1]]; an array ob binary digits.**

**import java.util.List;**

**import java.util.ArrayList;**

**class Result {**

public static int minMoves(List<Integer> avg) {

int movs=0;

int[] arrayOrdered = new int[avg.size()];

for(int i=0; i<avg.size(); i++) {

arrayOrdered[i] = avg.get(i);

}

for (int x = 0; x < arrayOrdered.length; x++) {

for (int i = 0; i < (arrayOrdered.length-x-1); i++) {

if(arrayOrdered[i] < arrayOrdered[i+1]) {

int tmp = arrayOrdered[i+1];

arrayOrdered[i+1] = arrayOrdered[i];

arrayOrdered[i] = tmp;

movs++;

}

}

}

return movs;

**}**

**}**

**public class Solution {**

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

**List<Integer> avg = new ArrayList<>();**

**avg.add(1);**

**avg.add(1);**

**avg.add(1);**

**avg.add(1);**

**avg.add(0);**

**avg.add(1);**

**avg.add(0);**

**avg.add(1);**

**int result = Result.minMoves(avg);**

**System.out.println(String.valueOf(result));**

**}**

**}**