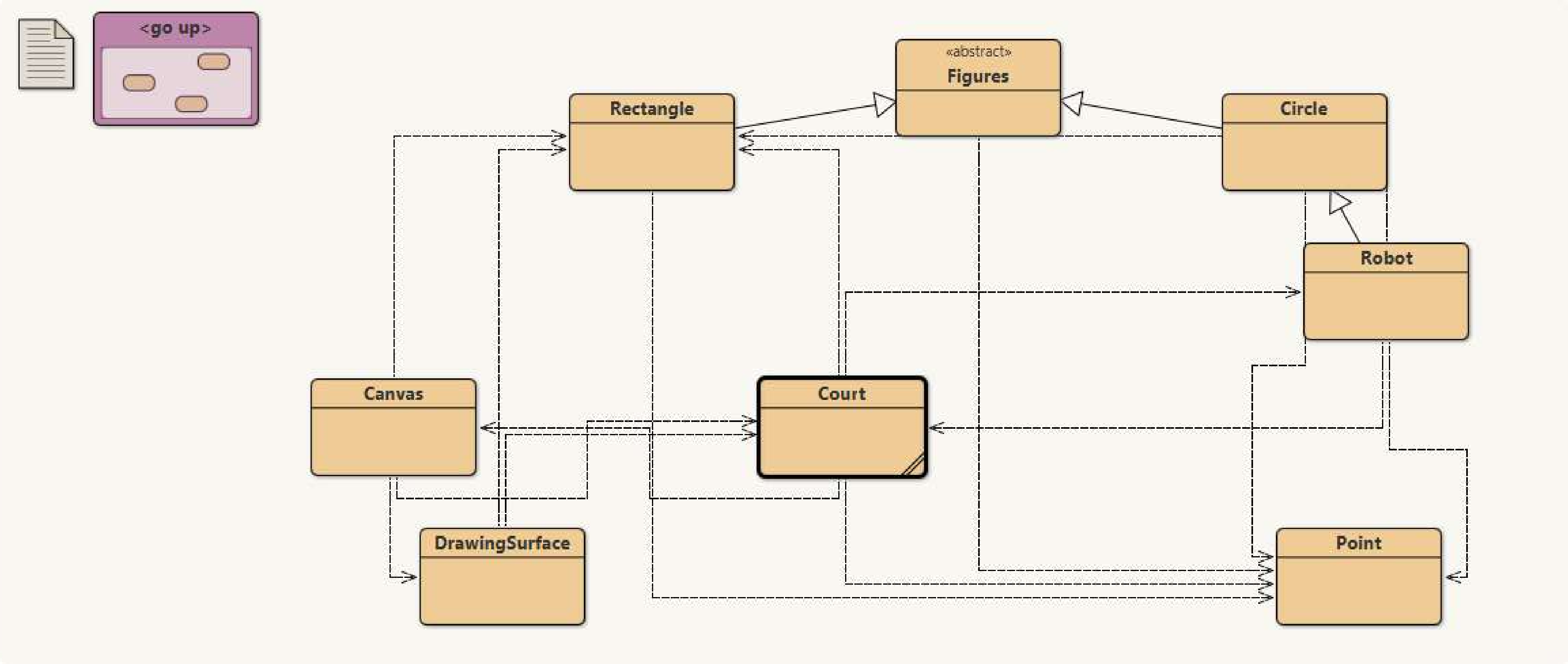

<go up>
«abstract»
Figures
Rectangle
Circle
Robot
Canvas
Court
DrawingSurface
Point

```java
package src.com.company;
import java.awt.*;


public class Circle extends Figures{
    public int rad;

    public Circle(Point position, int radius){
        rad = radius;
        wid = 2*radius;
        len = 2*radius;
        Point pos = position;
        String name;
        Color col = new Color(0, 0, 0);
    }

    public int getRadius() {
        return rad;
    }

    @Override
    public int minX() {
        return pos.getX();
    }

    @Override
    public int minY() {
        return pos.getY();
    }

    @Override
    public int maxX() {
        return pos.getX()+rad;
    }

    @Override
    public int maxY() { return pos.getY()+rad; }
}
```

```java
package src.com.company;

import javax.swing.*;
import java.awt.*;

import java.util.ArrayList;

public class Canvas {
    JFrame window;
    public DrawingSurface drawingSurface= new DrawingSurface();
    /**
     * The constructor of Canvas creates the windows for the simulation.
     */
    private Canvas() {
        window = new JFrame("simulation");
        window.setSize(Court.CWID, Court.CLEN); //no getter needed, variables are final
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//so the window closes properly
        window.setTitle("Robot Simulation");
        window.setBackground(Color.white);
        window.setVisible(true);//set window visible
        window.setResizable(false);
        window.getContentPane().setBackground(Color.white);//set background
        window.add(drawingSurface);
    }

    private static Canvas canvasSingleton;
    /**
    *Method checks if there is an existing Canvas and creates a new one if necessary -> Singleton
     */
    public static Canvas getCanvas() {
        if (canvasSingleton == null) {
            canvasSingleton = new Canvas();
        }
        return canvasSingleton;
    }

    public void wait(int ms) {
        try {
            Thread.sleep(ms);
        } catch (Exception END) {}
    }

    public void drawObs(ArrayList<Rectangle> Obs) {
        drawingSurface.repaintObs(Obs);
    }

    public void repaint(){
        drawingSurface.repaint();
    }

}
```

```java
package src.com.company;

import java.awt.Color;

public class Rectangle extends Figures{

    /**
     * Constructor for rectangles.
     * Currently all input methods as well as the rng only give positive integers, thus there is no need to ensure
     * positive values in this method.
     * @param position
     * @param width
     * @param length
     * @param name1
     * @param color
     */
    public Rectangle(Point position, int width, int length, String name1, Color color) {
        pos=position;
        len=length;
        wid=width;
        name=name1;
        if (color==Color.white) col=Color.BLACK; //replacing white color input with black
            else col=color;
    }

    //prints Attributes of the rectangle, was used for early troubleshooting
    public void printAtr(){
        System.out.println("Position:");
        pos.printPos();
        System.out.println("The X-Extent is "+wid);
        System.out.println("The Y-Extent is "+len);
        System.out.println("Object Name: "+name);
        System.out.println("Object Color: ");
        System.out.println("R: "+col.getRed());
        System.out.println("G: "+col.getGreen());
        System.out.println("B: "+col.getBlue());

    }

    /**
     * Lightweight method to check if Rectangles overlap each other.
     * This works by comparing the overlap for each dimension. If the overlap is true for two dimensions,
     * 'TRUE' is displayed.
     * @param r2
     * @return
     */
    public boolean checkOverlapR(Rectangle r2){
        Point p1=getPos();                      //Position of r1
        Point p2=r2.getPos();                   //Position of r2
        if (p2.getX()>=p1.getX() && p2.getX()<=p1.getX()+getWid()) {    //Check if p2 is INSIDE the x-Extent of r1
```

```
            //Check if the y-Extents overlap
            if (p2.getY()>=p1.getY() && p2.getY()<=p1.getY()+getLen()) return
true;//p2 in y-Ext. of r1
            if (p1.getY()>=p2.getY() && p1.getY()<=p2.getY()+r2.getLen()) return
true;//p1 in y- Ext of r2
        }
        if (p1.getX()>=p2.getX() && p1.getX()<=p2.getX()+r2.getWid()) { //Check if
p1 is INSIDE the x-Extent of r2
            //Same y-overlap test as above
            if (p2.getY()>=p1.getY() && p2.getY()<=p1.getY()+getLen()) return
true;
            if (p1.getY()>=p2.getY() && p1.getY()<=p2.getY()+r2.getLen()) return
true;
        }
        return false;
    }

    @Override
    public int minX() {
        return pos.getX();
    }

    @Override
    public int minY() {
        return pos.getY();
    }

    @Override
    public int maxX() {return pos.getX()+wid;}

    @Override
    public int maxY() {
        return pos.getY()+len;
    }
}
```

```java
package src.com.company;
import java.lang.Math;

public class Point {
    private int x;
    private int y;

    public Point(){}

    public Point(int newX, int newY)
    {x=newX; y=newY;}

    //Methods
    //getters & setters
    public int getX(){return this.x;}
    public int getY(){return this.y;}
    public void setX(int newX){this.x=newX;}
    public void setY(int newY){this.y=newY;}

    //Print
    public void printPos(){
        System.out.println("The X-Value is "+getX());
        System.out.println("The Y-Value is "+getY());}

    //Move
    public void moveBy(int dx, int dy){
        this.x=x+dx;
        this.y=y+dy; }

    public void moveByVec(Point vector){
        this.x=x+vector.getX();
        this.y=y+vector.getY();
    }
    //distance
    double getDist(Point P){
        return(Math.sqrt((Math.pow((P.getX())-x,2))+(Math.pow((P.getY())-y,2))));
};

    //Outputs Points to the Console
    public void printPoints(Point[] poi) {
        for (int i = 0; i < poi.length; i++) {
            System.out.println("Point "+i);
            poi[i].printPos();
        }
    }
}
```

```java
package src.com.company;

import java.util.ArrayList;
import javax.swing.JPanel;
import java.awt.*;

public class DrawingSurface extends JPanel {
    ArrayList <Rectangle> Obs;

    /**
     * The overwritten paintComponent Method calls the super Method to clear the
Canvas (advised on Stackoverflow)
     * We limited this method to only draw our singular robot. If more circles are
desired, this method must be
     * reworked.
     * @param g
     */
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);
        if (Obs!=null) {
            for (int i = 0; i < Obs.size(); i++) {
                Rectangle r = Obs.get(i);
                g.setColor(r.getCol());
                g.drawRect(r.getPos().getX(), r.getPos().getY(), r.getWid(),
r.getLen());
                g.fillRect(r.getPos().getX(), r.getPos().getY(), r.getWid(),
r.getLen());
            }
        }
        g.setColor(Court.maggse.getCol());
        g.fillOval(Court.maggse.pos.getX(), Court.maggse.pos.getY(),
Court.maggse.rad, Court.maggse.rad);
        g.setColor(Color.BLACK);
        g.drawOval(Court.maggse.pos.getX(), Court.maggse.pos.getY(),
Court.maggse.rad, Court.maggse.rad);
        }

    /**
     * We could not really adapt ourselves to the 'Zeichne Figuren' method from
the task and made the repaint a little
     * differently. We apologize for the inconvenience.
     * @param newObs
     */
    public void repaintObs(ArrayList<Rectangle> newObs){
        Obs=newObs;
        this.repaint();
    }
    public void repaint (){
        super.repaint();
    }


}
```

```java
package src.com.company;

import java.awt.Color;

public abstract class Figures {
    /**
     * Super Class for Rectangle and Circle combining their common attributes.
     */
    protected Point pos = new Point(0,0);
    protected String name;
    protected Color col = new Color(0, 0, 0);
    protected int len;    //extent of y
    protected int wid;    //extent of x

    //Getters & Setters
    public Point getPos(){return pos;}
    public String getName(){return name;}
    public Color getCol(){return col;}
    public int getLen(){return len;}
    public int getWid(){return wid;}

    public void setPos(Point newPos){pos=newPos;}
    public void setName(String newName){name=newName;}
    public void setLen(int newLen){len=newLen;}
    public void setWid(int newWid){wid=newWid;}
    public void setCol(Color newCol){
        if (newCol==Color.white) System.out.println("Color white is illegal.");
        else col=newCol;}
    public void setColN(String newCol){                      //Address a color by
its name.
        if (newCol.equals("WHITE")) System.out.println("Color white is illegal.");
        else col=Color.getColor(newCol);}

    //Moving around (Copied from the Point class to avoid nesting statements)
    public void moveBy(int dx, int dy){
        pos.setX(pos.getX()+dx);
        pos.setY(pos.getY()+dy);}

    public void moveByVec(Point vector){
        pos.setX(pos.getX()+vector.getX());
        pos.setY(pos.getY()+vector.getY());}


    /**
     * These methods are overridden in the subclasses.
     * @return
     */
    abstract public int minX();
    abstract public int minY();
    abstract public int maxX();
    abstract public int maxY();
}
```

```java
package src.com.company;
import java.awt.*;
import java.util.Scanner;


public class Robot extends Circle {

    //enum keywords for the question commands
    public enum Keyword {
        NAME,
        AGE,
        MANUFACTURER,
        SEX,
        END

    }

    //constructor
    public Robot(Point position, int radius) {
        super(position, radius);
        col = Color.ORANGE;     //Who doesn't like orange?
    }

    /**
     * Voice recognition algorithm according to the task sheet.
     */
    public void voiceRecognition() {
        //ask the robot questions, ends if END is the input

        Scanner scanner = new Scanner(System.in);
        boolean end = false;

        while (!end) {
            String command = new String();

            System.out.println("Please enter a command from the following keywords
to ask the robot a question:\n" +
                    "NAME\nAGE\nMANUFACTURER\nSEX\n" +
                    "Once you're done asking questions, enter the command \"END\"
:)");
            String input = scanner.nextLine().toUpperCase();
            //convert all the input to uppercase so the keywords are read properly

            if (input.contains("END")) {

                end = true;
            }//if the input is END, the loop stops

            //check if the input is a keyword
            for (Keyword keyword : Robot.Keyword.values()) {
                if (input.contains(keyword.toString())) {
                    command = keyword.toString();
                }
            }
            switch (command) {

                case "NAME":
                    System.out.println("My name is Maggse D'avis");
                    break;

                case "AGE":
                    System.out.println("I'm 3 Months old");
                    break;

                case "MANUFACTURER":
                    System.out.println("Made by 2 & 2");
                    break;

                case "SEX":
                    System.out.println("I'm a Robot bro");
                    break;

                case "END":
                    System.out.println("program ended");
                    break;


                default:
                    System.out.println("Please use the given commands!");
                    break;
            }
        }
    }

    /**
     * Checking if the Robot is touching the boundaries of the field.
     * @return
     */
    public boolean touchingBoundaryX() {
        if ((this.maxX() + 1) > Court.CWID) {
            return true;
        } else {
            return false;
        }
    }

    public boolean touchingBoundaryY() {
        if ((this.maxY() + 1) > Court.CLEN) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Avoiding the rectangles in a similar way we did the overlap check.
     * @param figures List of obstacles we want to avoid.
     * @return Boolean: Too close or not?
     */
    public boolean tooCloseTo_xEdge(Rectangle figures) {
        if (this.minX() < figures.maxX()) {         //ruling out some rectangles
we already passed.
            if (figures.minX() - this.maxX() < 5) {   //Dist to Object smaller
than 5
                if (this.maxY() > figures.minY() && this.minY() < figures.maxY())
{   //y-Extends overlap
                    return true;
                }
            }
        }
        return false;
    }
    //Same structure as above
    public boolean tooCloseTo_YEdge(Rectangle figures) {
        if (this.minY() < figures.maxY()) {
            if (figures.minY() - this.maxY() < 5) {   //Dist to Object smaller
than 5
                if (this.maxX() > figures.minX() && this.minX() < figures.maxX())
{   //x-Extends overlap
                    return true;
                }
            }
        }
        return false;
    }
}
```

```
package src.com.company;
import java.util.*;
import java.awt.Color;

public class Court {
    final public static int CWID = 1000;               //max X-Value on the court
    final public static int CLEN = 1000;               //max Y-Value on the court
    public Canvas canvas = Canvas.getCanvas();
    public ArrayList<Rectangle> obstacleList;
    public static Robot maggse = new Robot((new Point(0, 0)), 10);
    static Random rand = new Random();

    public Court() {
    }

    /**
     * The main Method is the first thing a third party programmer looks at. Thus
we kept it easy to read.
     * @param args
     */
    public static void main(String[] args) {
        Robot maggse = new Robot((new Point(0, 0)), 10);
        Scanner input = new Scanner(System.in);
        Court c = new Court();
        boolean end = false;
        while (!end) {


            System.out.println("Please choose a task for the robot to do\n" +
                    "if you want to ask the robot some questions please enter
\"QA\"\n" +
                    "to have the robot find the shortest path through a set of
coordinates enter path\n" +
                    "to have the robot run through a set of obstacle enter obst\n"
+
                    "enter \"END\" to end the program");
            String tasks = input.nextLine().toUpperCase();

            if (tasks.contains("END")) end = true;

            switch (tasks) {
                case "QA":
                    maggse.voiceRecognition();
                    break;

                case "PATH":{
                    c.pathLauncher();}
                    break;

                case "OBST":{
                    c.avoidObst();}
                    break;

                case "END":
                    System.out.println("buh bye");
                    System.exit(0);


                default:
                    System.out.println("unknown input, please try again.");
                    break;
            }
        }
    }

    /**
     * POI-Input. See further descriptions at submethods below.
     * @return
     */
    public Point[] inputPoints() {
        int k = limitInput(intInput("How many POIs will be selected?"),100);
        System.out.println(k+" Points will be chosen.");
        k++;
        Point[] poi = new Point[k];
        poi[0]=new Point(0, 0);
        for (int i = 1; i < k; i++) {
            poi[i] = new Point(0, 0);
            int x = limitInput((intInput("Insert X-Value for POI #" + i)), CWID);
            poi[i].setX(x);
            int y = limitInput(intInput("Insert Y-Value for POI #" + i), CLEN);
            poi[i].setY(y);
        }
        return poi;
    }    //Redundancy of x and y variables in the for-loop to increase
readability.



    /**
     * To increase the modularity and reduce the size of our program we decided to
implement
     * reusable input methods. The intinput method is secured by the try-catch
method.
     * To further protect the Program from its user, the limitinput method only
allows
     * inputs that are positive and below a certain max value. The Methods are
made in a way that
     * would allow later changes to the extend of the playing field as long as
those
     * changes limit to the first quadrant of coordinates.
     * @param Message
     * @return
     */
    private int intInput(String Message) {       //console input method for any
integers
        System.out.print(Message);
        System.out.println(" Please input a positive integer!");
        Scanner scInput = new Scanner(System.in);
        try {
            return scInput.nextInt();
        } catch (Exception e) {
    }

    private int limitInput(int input, int limit) {       //Method to limit the
value of positive integers
        if (input > limit) {                               //Also eliminates
negative integers.
            System.out.println("Your input was lowered to the maximum value of " +
limit);
            return limit;
        }
        if (input < 0) {
            System.out.println("Negative input detected, input value was set to
'0'!");
            return 0;
        }
        return input;
    }

    /**
     * Lightweight, versatile RNG method and its extension for a random Color.
     */
    private int rng(int min, int max) {
        int n = rand.nextInt(((max - min) + 1));    //RNG from 0 to
max-min-difference, +1 to include the max value
        n = n + min;                          //adding min to have an RNG between
min and max
        return n;
    }

    private Color randCol() {        //simple method for random colors using the
Color constructor
        Color rc = new Color(rng(0, 250), rng(0, 250), rng(0, 250));
        //Maximum of 250 to avoid white rectangles.
        return rc;                  //the r,g and b value each call the rng Method
    }

    /**
     *  Creating one random rectangle. The rectangles name is given by the caller
of this method.
     *  Interference cannot occur as there is currently only one Rectangle
generation Method per instance
     *  of 'court'
     * @param index Identifier
     * @return Random Rectangle
     */
    public Rectangle createRec(int index) {
        Point p = new Point(rng(0, 900), rng(0, 900)); //100 offset to keep
rectangles on the Court.
        //10x10 as minimum rectangle size.
        int w = rng(10, 100);
        int l = rng(10, 100);
        String n = ("Rectangle " + index);
        Color c = randCol();
        Rectangle r = new Rectangle(p, w, l, n, c);
        return r;
    }

    /**
     * Creates a user-given amount of Rectangles. Further documentation below!
     * @return
     */
    public ArrayList<Rectangle> createObstacleList() {
        int k = intInput("How many Rectangles should be created?");
        ArrayList<Rectangle> rectangles = new ArrayList<Rectangle>();
        for (int c = 0; c < k; c++) {
            /*
            The for-loop allows for the unsuccessful attempts to easily be tracked
to break the loop in case
            of too many attempts
             */
            for (int i = 0; i < 50; i++) {
                Rectangle r = createRec(i);
                boolean t1 = false;
                //t1 indicates if ANY of the checks returned true
                //comparing the new rectangle to the existing ones to prevent
overlap
                for (int j = 0; j < rectangles.size(); j++) {
                    t1 = r.checkOverlapR(rectangles.get(j));
                    if (t1) break;
                }
                //if there is no overlap, the new rectangle is added, else go to
next iteration
                if (!t1) {
                    rectangles.add(r);
                    break;
                }
                //break the loop & add r to list if no overlaps found
                if (i == 49) {
                    System.out.println(c + "Rectangles were created before
failure!");
                    return rectangles;
                }
            }    //end the method if creation failed 50 times,
        }
        System.out.println("All " + k + " rectangles created successfully!");
        return rectangles;
    }

    /**
     * POI Sorter.
     * We eventually decided against using the Arraysort class by java as it would
require
     * knowledge of the implementation of interfaces which currently exceed our
programming skill.
     * @param poi Unsorted POI List
     * @return neatly sorted POI List :)
     */
    public Point[] poiSort(Point[] poi) {
        for (int i = 0; i < poi.length -1; i++) {
    double dist1 = poi[i].getDist(poi[i+1]);
```

```
                if (poi[i].getDist(poi[n]) < dist1) {
                    Point p = poi[n];
                    poi[n] = poi[i+1];
                    poi[i+1] = p;
                }
                }
        }
        return poi;
    }


    /**
     * The method makes the robot avoid an amount ov user-generated Obstacles.
     * The robot has succeeded when it reaches the lower right corner.
     */
    public void avoidObst() {

        boolean rightSideClear = true;
        boolean botSideClear = true;
        obstacleList=createObstacleList();
        maggse.setPos(new Point(0,0));
        canvas.drawObs(obstacleList);

        while (maggse.maxX() < 1001 & maggse.maxY() < 1001) {

            for (Rectangle m : obstacleList) {
                if (maggse.tooCloseTo_xEdge(m) | maggse.touchingBoundaryX()) {
                    rightSideClear = false;
                }//touching right side

                if (maggse.tooCloseTo_YEdge(m) | maggse.touchingBoundaryY()) {
                    botSideClear = false;
                }//touching bottom side
            }

            if (rightSideClear && botSideClear) {maggse.moveBy(1,1);}

            if (!rightSideClear && botSideClear) {maggse.moveBy(0,1);}

            if (!rightSideClear && !botSideClear) {break;}

            if (rightSideClear && !botSideClear) {maggse.moveBy(1,0);
            }
            movementGraph(maggse.getPos(), 16); //16ms stall roughly transmits to
60fps
            rightSideClear = true;
            botSideClear = true;
        }
        if (maggse.maxX()>999 && maggse.maxY()>999) System.out.println("Task
finished successfully!");
        else {System.out.println("Maggse: Help! I'm stuck!");
            System.out.println("Task failed!");}
        maggse.setPos(new Point (0,0));
    }
    /**
     * This method makes the Robot find the quickest path to visit all given POI,
runtime solely depends on number of
     * POI as one movement always takes 4 seconds (100 steps * 40ms stall)
     */
    public void pathLauncher() {
        maggse.setPos(new Point(0,0));
        obstacleList=null;
        canvas.drawObs(null);
        obstacleList= new ArrayList<Rectangle>();
        Point[] poi = inputPoints();
        poi=poiSort(poi);
        for (int i=0; i < poi.length;i++){
            String name = ("Marker #"+i);
            obstacleList.add(new Rectangle(poi[i],10,10,name,Color.RED));
            System.out.println("Next Point: "+i);
            poi[i].printPos();
            moveTo(poi[i],100,40);
            canvas.drawObs(obstacleList);
        }
    }

    /**
     * The robot will take the same time for any movement, but the animation will
be smooth if there is sufficient.
     * Computing power of course :)
     */
    public void moveTo(Point target,int steps, int stall) {
        Point current = maggse.getPos();
        int stepX = (target.getX() - current.getX()) / steps;
        int stepY = (target.getY() - current.getY()) / steps;
        for (int i = 0; i < steps; i++) {
            current.moveBy(stepX,stepY);
            movementGraph(current, stall);
        }

    }

    /**
     * Versatile movement animation method for the POI path below.
     * @param nextP new robot position
     * @param stall waiting time
     */
    public void movementGraph(Point nextP, int stall){
        maggse.setPos(nextP);
        canvas.wait(stall);
        canvas.repaint();
    }
}
```