

CS 101: Introduction to Computer Science

Project 3 - Plotting Points on an Invisible Grid

Student Name Mohammad El-Abid

Student ID

Professor Yasser Elleithy

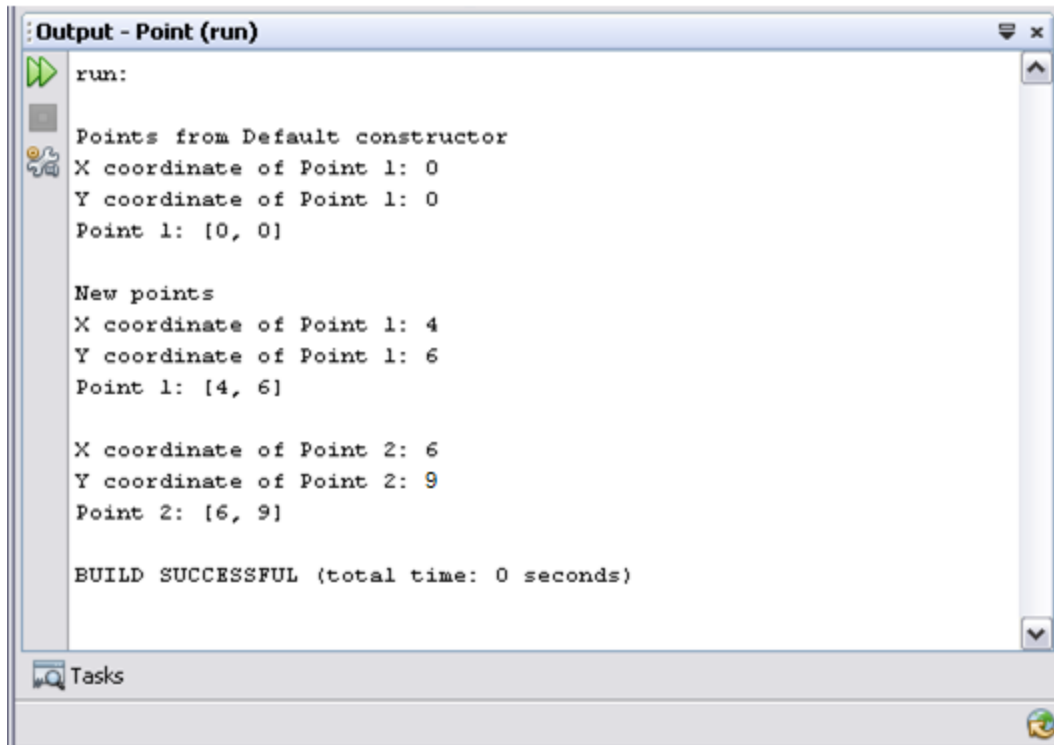
Date Friday, September 30, 2011

Table of Contents

<u>Abstract</u>	Page 3
<u>Introduction</u>	Page 3
<u>Screenshots</u>	Page 4
<u>Code</u>	Page 5
<u>Conclusion</u>	Page 9
<u>Works Used</u>	Page 9

Abstract

This application allows for the plotting of points on a Cartesian grid (x and y coordinates). We are given a driver that plots the points and outputs the points using three methods: `getX()`, `getY()`, and `toString()`



```
run:
Points from Default constructor
X coordinate of Point 1: 0
Y coordinate of Point 1: 0
Point 1: [0, 0]

New points
X coordinate of Point 1: 4
Y coordinate of Point 1: 6
Point 1: [4, 6]

X coordinate of Point 2: 6
Y coordinate of Point 2: 9
Point 2: [6, 9]

BUILD SUCCESSFUL (total time: 0 seconds)
```

Introduction

The purpose of this application was to make us comfortable with developing our own classes. As all of my other projects have had classes outside of the driver, it wasn't too hard for me. Because of this I decided to also add some methods that used popular formulas. Doing this I learned about the built in `Math` class and writing equations in Java. I also decided that the formulas weren't enough of a challenge and added a `Point3D` class and a `Graph` class (more about these classes in [conclusion](#)).

Screenshots

Application output to stdout:

```
[exec:exec]

Points from Default constructor
X coordinate of Point 1: 0
Y coordinate of Point 1: 0
Point 1: [0, 0]

New points
X coordinate of Point 1: 4
Y coordinate of Point 1: 6
Point 1: [4, 6]

X coordinate of Point 2: 6
Y coordinate of Point 2: 9
Point 2: [6, 9]

-----
BUILD SUCCESS
-----
```

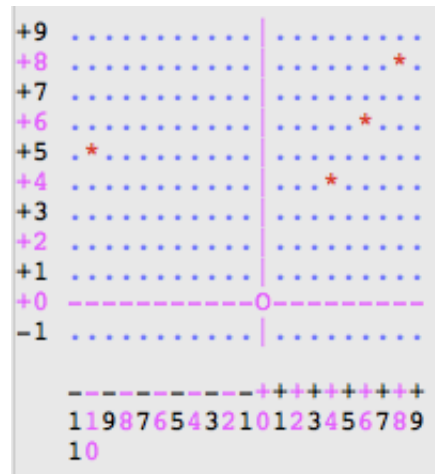
Test Suite of Point:

```
Testsuite: edu.bridgeport.cartesianpoints.PointTest
Tests run: 7, Failures: 0, Errors: 0, Time elapsed:
test:
Deleting: /var/folders/3q/3y6v4dzn1lb79x8k07cby4w000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Test Suit of Point3D:

```
Testsuite: edu.bridgeport.cartesianpoints.Point3DTest
Tests run: 7, Failures: 0, Errors: 0, Time elapsed: 0.085 sec
test:
Deleting: /var/folders/3q/3y6v4dzn1lb79x8k07cby4w00000gn/T/TE
BUILD SUCCESSFUL (total time: 1 second)
```

An example Graph:



Code

Only files needed to complete the homework are in the PDF, please see the included project for additional code such as the Point3D and Graph class as well as junit tests.

File: App.java (provided driver)

```
package edu.bridgeport.cartesianpoints;

/**
 * @author Yasser Elleithy [yelleithy at bridgeport.edu]
 */
public class App
{
    /**
     * Main driver for the application
     * @param arguments
     */
    public static void main(String[] arguments){
        // Create a new point using the default constructor
        Point myPoint1 = new Point();

        // Display the points
        System.out.println("\nPoints from Default constructor");
        System.out.println("X coordinate of Point 1: " +
myPoint1.getX());
        System.out.println("Y coordinate of Point 1: " +
myPoint1.getY());
        System.out.println("Point 1: " + myPoint1.toString() +
"\n");

        // Now change the points using the setPoint method
        myPoint1.setPoint(4, 6);

        // Display the points
        System.out.println("New points");
        System.out.println("X coordinate of Point 1: " +
myPoint1.getX());
        System.out.println("Y coordinate of Point 1: " +
myPoint1.getY());
        System.out.println("Point 1: " + myPoint1.toString() +
"\n");

        // Create a new point using another constructor
        Point myPoint2 = new Point(6, 9);

        // Display the points
        System.out.println("X coordinate of Point 2: " +
myPoint2.getX());
```

```

        System.out.println("Y coordinate of Point 2: " +
myPoint2.getY());
        System.out.println("Point 2: " + myPoint2.toString() +
"\n");
    }
}

```

File: Point.java

```

package edu.bridgeport.cartesianpoints;

/**
 * A class for handling points on a Cartesian grid.
 * @author Mohammad Typaldos [mohammad at reliablerabbit.com]
 */
public class Point {
    /**
     * The x coordinate of the point
     */
    private int xCord;
    /**
     * The y coordinate of the point
     */
    private int yCord;

    /**
     * Makes a new point on the origin
     */
    public Point(){
        xCord = yCord = 0;
    }

    /**
     * Makes a point at the coordinates passed into the
    constructor.
     * Uses setPoint internally
     * @param setXCord The X coordinate
     * @param setYCord The y coordinate
     * @see Point#setPoint(int, int)
     */
    public Point(int setXCord, int setYCord){
        setPoint(setXCord, setYCord);
    }

    /**
     * @return the x coordinate of the point
     */
    public int getX(){
        return xCord;
    }
}

```

```

    }

    /**
     * @return the y coordinate of the point
     */
    public int getY(){
        return yCord;
    }

    /**
     * Sets the x coordinate of the point
     * @param newXCord The new X coordinate
     */
    public void setX(int newXCord){
        xCord = newXCord;
    }

    /**
     * Sets the y coordinate of the point
     * @param newYCord The new y coordinate
     */
    public void setY(int newYCord){
        yCord = newYCord;
    }

    /**
     * Set the new coordinates
     * @param newXCord New x coordinate
     * @param newYCord New y coordinate
     */
    public void setPoint(int newXCord, int newYCord){
        setX(newXCord);
        setY(newYCord);
    }

    /**
     * @return returns a string, such as [x, y]
     */
    @Override
    public String toString(){
        return String.format("[%d, %d]", getX(), getY());
    }

    /**
     * Uses the distance formula to calculate the distance
    between two points
     * @param point The point that you want to know the distance
    between

```



```

        * @return positive double of the distance between the two
points.
    */
    public double distanceBetween(Point point){
        // d = |sqrt( (x2 - x1)^2 + (y2 - y1)^2 )|
        double distance;

        // (x2 - x1)^2
        distance = Math.pow(point.getX() - getX(), 2);
        // + (y2 - y1)^2
        distance += Math.pow(point.getY() - getY(), 2);
        // absolute value of the square root
        distance = Math.abs(Math.sqrt(distance));

        return distance;
    }

    /**
     * Uses the midpoint formula to create a new point that is
between the two points
     * @param point The end point of the line.
     * @return A point object that resides in the middle of the
line.
    */
    public Point midpointBetween(Point point){
        // x = (x1 + x2) / 2
        // y = (y1 + y2) / 2
        int x = (getX() + point.getX()) / 2;
        int y = (getY() + point.getY()) / 2;

        return new Point(x, y);
    }

    /**
     * Tells if points are equal to each other. Checks their X
and Y values
     * @param object The object to compare
     */
    @Override
    public boolean equals(Object object){
        if(this == object) return true; // same object?

        if(object.getClass() == Point.class){
            Point point = (Point) object; // now can cast so have
access to class methods
            return(
                getX() == point.getX() &&
                getY() == point.getY()
            ); // same points?
        }
    }

```

```

        } else if(object.getClass() == Point3D.class){
            // if 3D point, let its equals method handle the
check
            Point3D point = (Point3D) object;
            return(point.equals(this));
        } else {
            return false;
        }
    }

    @Override
    /**
     * Provide a unique int based on the data of this object.
     * Objects with the same data should have the same hashCode()
     */
    public int hashCode() {
        int hash = 3;
        hash = 29 * hash + this.xCord;
        hash = 29 * hash + this.yCord;
        return hash;
    }
}

```

Conclusion

I learned about annotations since I overwrote the `toString()` method that is defined in the `Object` class. When I did this my IDE alerted me that I needed to add an “`@Override`” before the method declaration to suppress warnings or errors by the compiler. I then researched annotations on the Java website to learn that there are two other annotations that are used by compilers. One is for producing a warning for each use of deprecated code, the other for suppressing errors if the compiler is failing to understand how the code will work.

After I completed the `Point` class, I also wrote a `Point3D` class for storing points that also have a Z axis. The `Point3D` class required me to rewrite and deprecate some inherited methods. I wanted a greater challenge, so I also wrote a `Graph` class. The `Graph` class allows for points to be plotted, then displayed in an ASCII format on the terminal with some colors (A standard set of letters and shapes, it stands for: American Standard Code for Information Interchange. Colors only display on supported terminal interfaces).

Works Used

[A Splash of Text Color With Your Java](#)