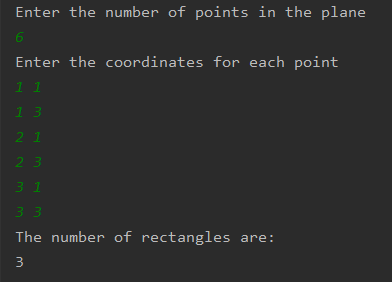
Rectangle Problem

# Objectives:

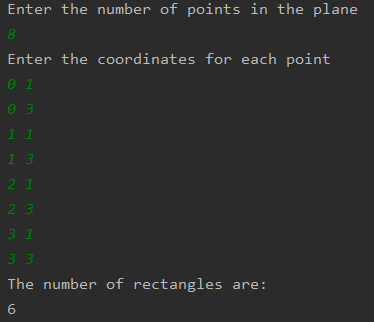
* Given an arbitrary number of points in the cartesian plane, determine the total number of rectangles which can be determined from them

# Example:

* If we have the following points in plane:
  + **(1,1), (1,3), (2,1), (2,3), (3,1), (3,3)**
* The output should be: **3**
* The code snippets are shown below:

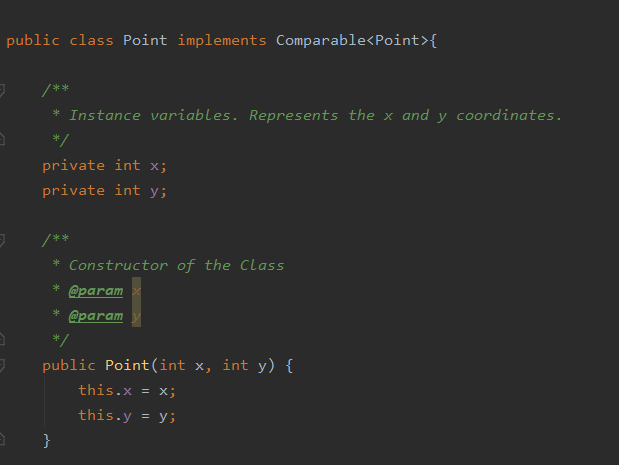


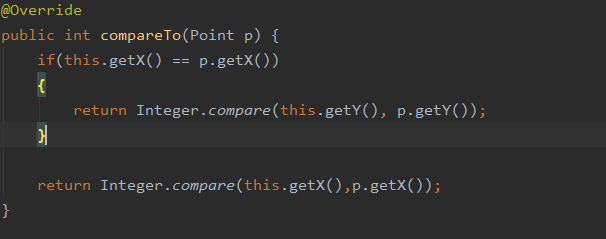
* If we have the following points in plane:
  + **(0,1), (0,3), (1,1), (1,3), (2,1), (2,3), (3,1), (3,3)**
* The output should be: **6**
* The code snippets are shown below:

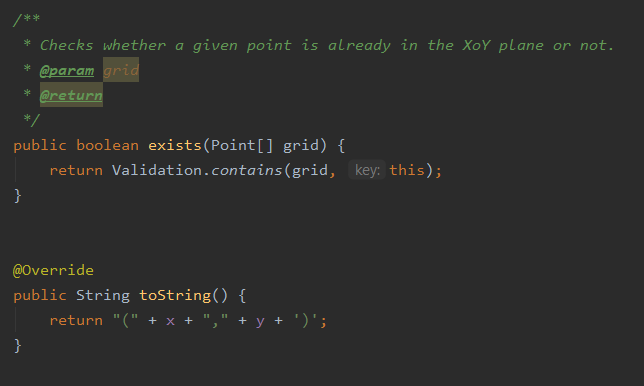


# Point Class

* This class has two instance variables, i.e. **x and y,** which are the coordinates of a given point in plane.
* The point will also be initialized using a constructor. In addition, for a proper encapsulation, the class also provides getters to access the fields.
* This class also implements the **Comparable interface,** in order make the comparison between objects easier. When implementing this interface, one must override the **compareTo()** method. This method checks first if the current point and another point given as parameter have the same x coordinate. If so, the the y coordinate is checked.
* The last two methods in this class are the **exist()** and **toString().** The first one checks if a point is already in the plane. The second one is responsible to return the string representation of the object.
* The point class is presented in the figures below

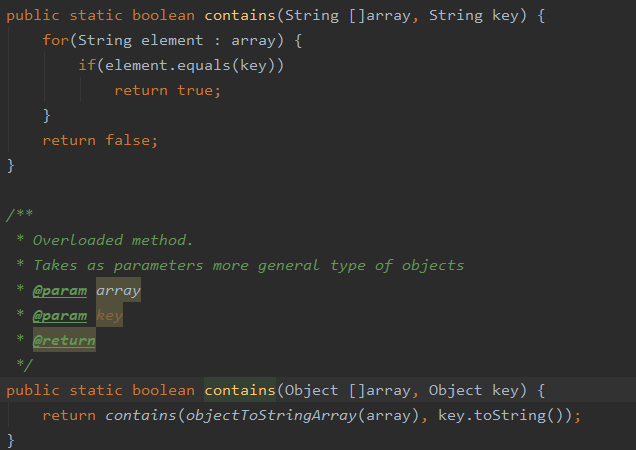


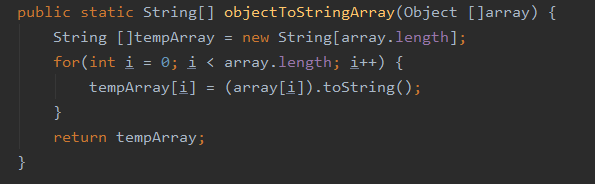
****

****

# Validation Class

* This class is responsible for the mechanism of checking if a point is already in the plane.
* For this purpose, we have two **contains()** methods.
* The last method is responsible to convert an array of generic objects into an array of strings.
* The methods of this class are shown in the figures below:



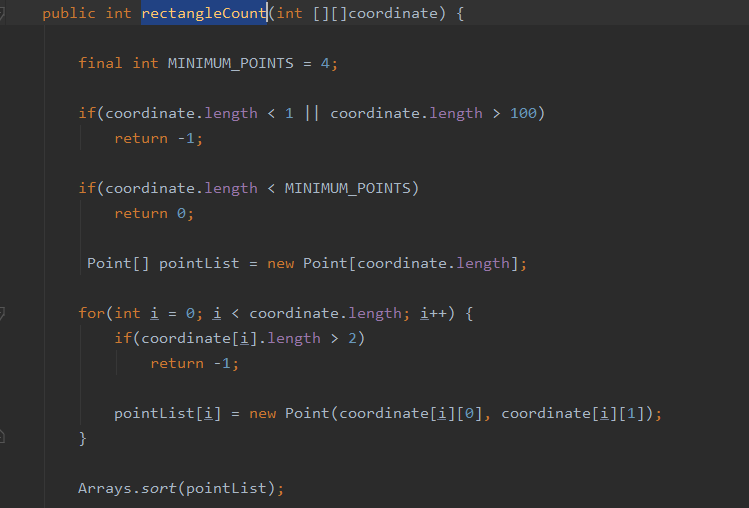


# RectangleNumber Class

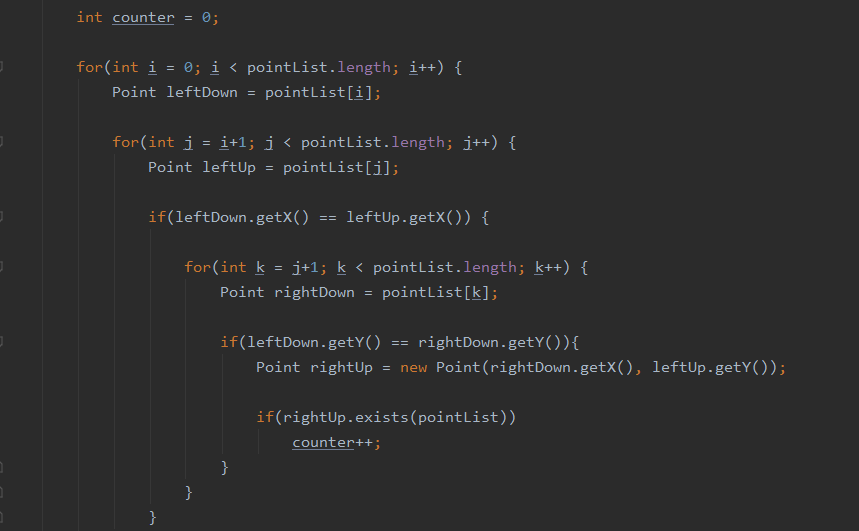
* This class implements the functionality of counting the number of rectangles.
* It provides a single method, **rectangleCount()**.
* The first part of the method provides some simple validation, i.e. if the user doesn’t

input at least 4 points, etc. After this validation, the points will be sorted.

* The first part of the method is shown below:



* When the points are sorted, the algorithm checks if 2 out of 4 points are on the same line. If so, it looks for the third point. After that, it checks if the third and first point are on the same line (the same y coordinate).
* If so, it determins the location of the last point and increment the counter by 1.



# Main Class

* This class contains 2 methods, one of the reads the coordinates from the keyboard and the other one is the main method which calls the first method
* These methods are shown in the figures below:

