IT-314 LAB-9

Software Engineering

Prof. Saurabh Tiwari

Student ID : 202201117

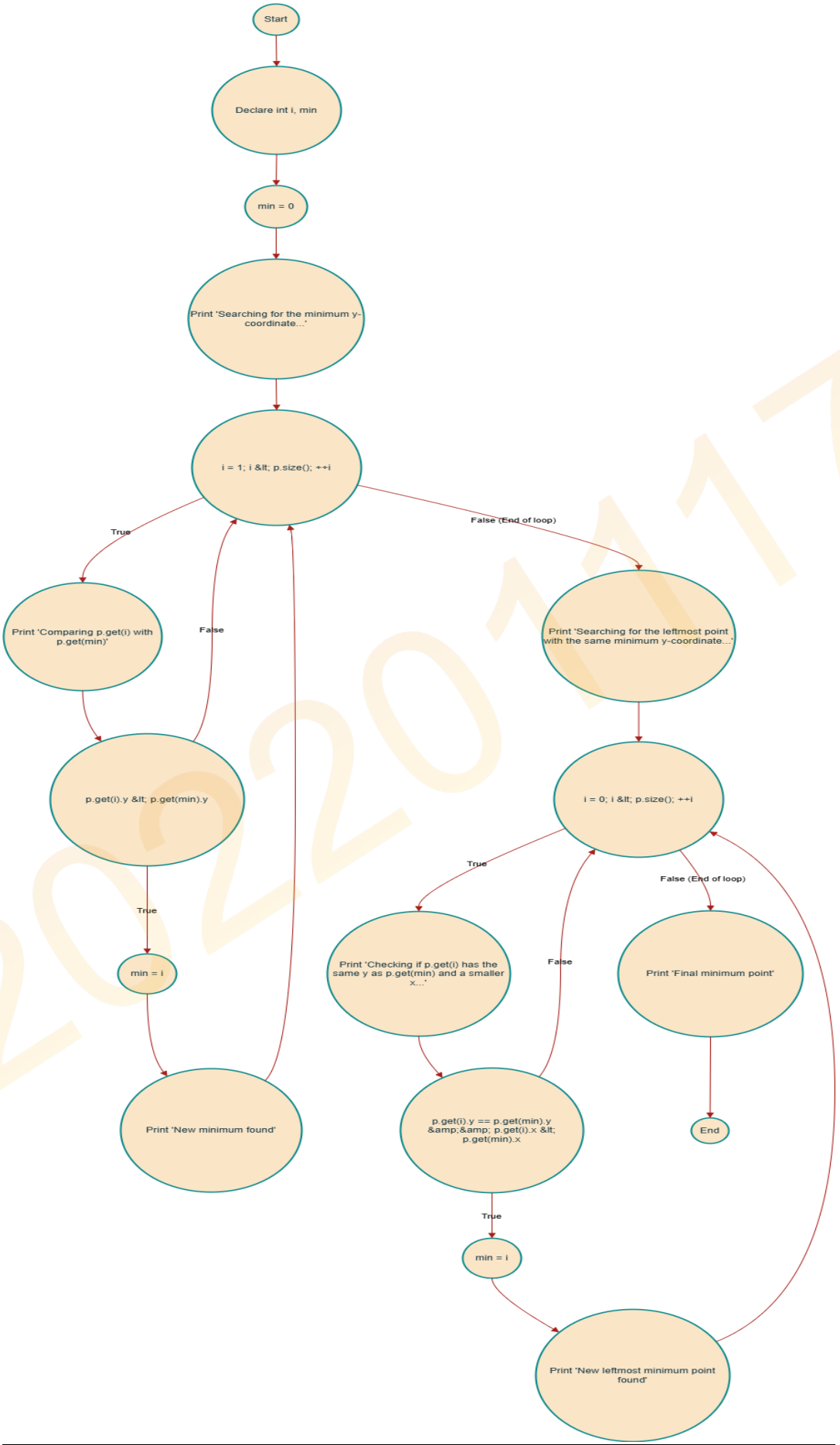
Name : Rushi Makadiya

➢ Question-1

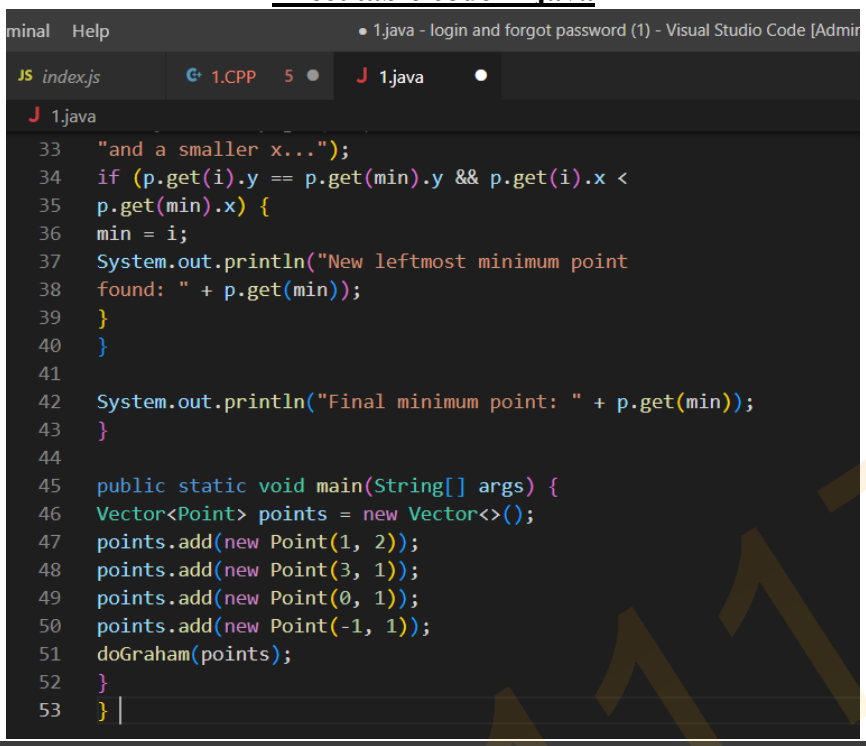
Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.

```
Vector doGraham(Vector p) {
        int i,j,min,M;

        Point t;
        min = 0;

        // search for minimum:
        for(i=1; i < p.size(); ++i) {
            if( ((Point) p.get(i)).y <
                        ((Point) p.get(min)).y )
            {
                min = i;
            }
        }

        // continue along the values with same y component
        for(i=0; i < p.size(); ++i) {
            if(( ((Point) p.get(i)).y ==
                        ((Point) p.get(min)).y ) &&
                (((Point) p.get(i)).x >
                        ((Point) p.get(min)).x ))
            {
                min = i;
            }
```

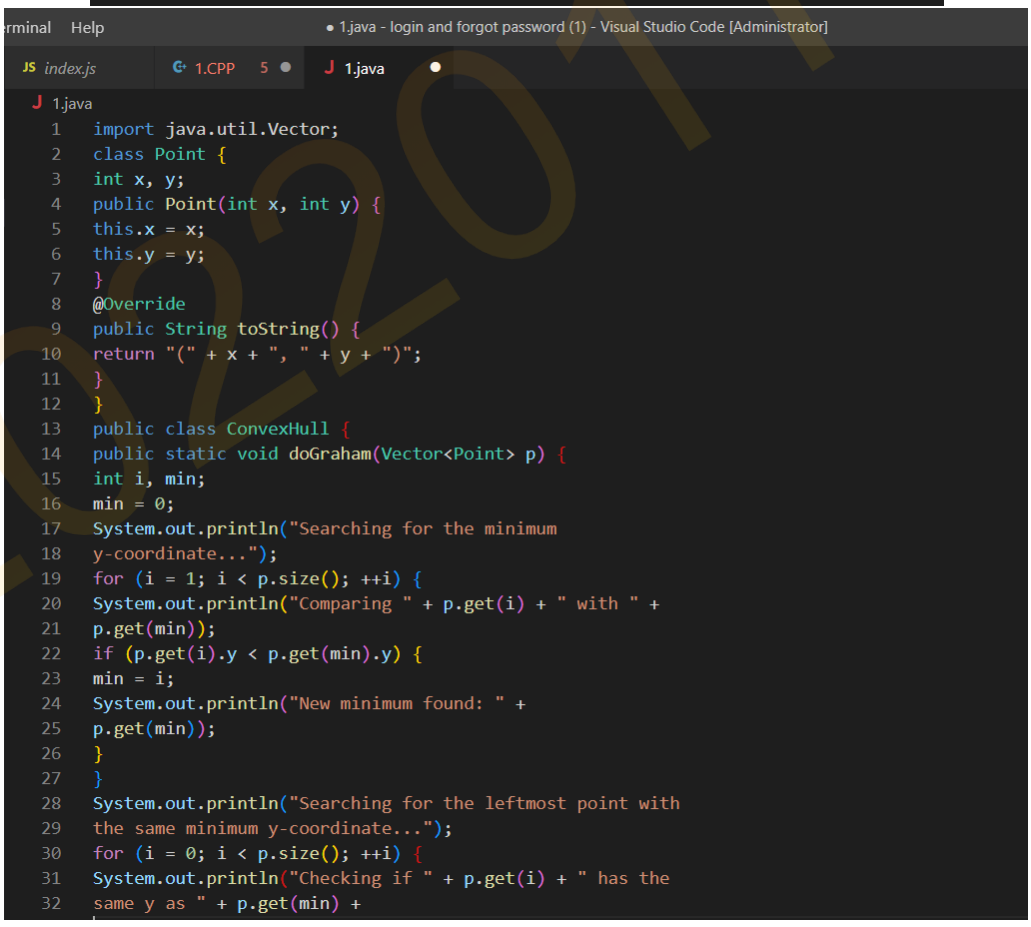
Flow chart



➢ Executable code in java



```
minal  Help                    • 1.java - login and forgot password (1) - Visual Studio Code [Admir
JS index.js   1.CPP  5 ●   1.java ●
1.java
33   "and a smaller x...");
34   if (p.get(i).y == p.get(min).y && p.get(i).x <
35   p.get(min).x) {
36   min = i;
37   System.out.println("New leftmost minimum point
38   found: " + p.get(min));
39   }
40   }
41
42   System.out.println("Final minimum point: " + p.get(min));
43   }
44
45   public static void main(String[] args) {
46   Vector<Point> points = new Vector<>();
47   points.add(new Point(1, 2));
48   points.add(new Point(3, 1));
49   points.add(new Point(0, 1));
50   points.add(new Point(-1, 1));
51   doGraham(points);
52   }
53   }
```



```
rminal  Help                    • 1.java - login and forgot password (1) - Visual Studio Code [Administrator]
JS index.js   1.CPP  5 ●   1.java ●
1.java
1   import java.util.Vector;
2   class Point {
3   int x, y;
4   public Point(int x, int y) {
5   this.x = x;
6   this.y = y;
7   }
8   @Override
9   public String toString() {
10  return "(" + x + ", " + y + ")";
11  }
12  }
13  public class ConvexHull {
14  public static void doGraham(Vector<Point> p) {
15  int i, min;
16  min = 0;
17  System.out.println("Searching for the minimum
18  y-coordinate...");
19  for (i = 1; i < p.size(); ++i) {
20  System.out.println("Comparing " + p.get(i) + " with " +
21  p.get(min));
22  if (p.get(i).y < p.get(min).y) {
23  min = i;
24  System.out.println("New minimum found: " +
25  p.get(min));
26  }
27  }
28  System.out.println("Searching for the leftmost point with
29  the same minimum y-coordinate...");
30  for (i = 0; i < p.size(); ++i) {
31  System.out.println("Checking if " + p.get(i) + " has the
32  same y as " + p.get(min) +
```

➢ **Construct test sets for your flow graph that are adequate for the following criteria: a. Statement Coverage. b. Branch Coverage. c. Basic Condition Coverage.**

a. Statement Coverage

Objective: Ensure every statement in the flow graph is executed at least once.

➢ **Test Case 1:**

1. **Inputs:** Any list p with more than one point (e.g., [(0, 2), (1, 3), (2, 1)])
2. **Description:** This will traverse through the entire flow, covering statements related to finding the minimum y-coordinate and identifying the leftmost minimum point.

➢ **Test Case 2:**

1. **Inputs:** [(3, 3), (3, 3), (4, 4)]
2. **Description:** This checks for points with identical y-coordinates and ensures that the logic for determining the leftmost point executes correctly.

b. Branch Coverage

Objective: Ensure every branch (true/false) from each decision point is executed.

➢ **Test Case 1:**

1. **Inputs:** [(0, 2), (1, 3), (2, 1)]
2. **Description:** This will take the true branch when finding the minimum y-coordinate.

➢ **Test Case 2:**

1. **Inputs:** [(3, 3), (3, 3), (4, 4)]
2. **Description:** This tests the scenario where y-coordinates are equal, triggering the branch for comparing x-coordinates.

- **Test Case 3:**

1. **Inputs:** [(2, 3), (2, 2), (3, 4)]
2. **Description:** This ensures the flow takes the false branch when checking for new minimum y-coordinates and the leftmost point logic.

c. Basic Condition Coverage

Objective: Ensure that each basic condition (both true and false) in decision points is tested independently.

- **Test Case 1:**

  - **Inputs:** [(2, 2), (3, 3), (4, 4)]
  - **Description:** This will evaluate both conditions for y-coordinate comparisons.

- **Test Case 2:**
  - **Inputs:** [(2, 2), (2, 2), (2, 3)]
  - **Description:** This checks the scenario where y-coordinates are identical, but the x-coordinate condition is evaluated.

- **Test Case 3:**
  - **Inputs:** [(4, 2), (3, 3), (2, 4)]
  - **Description:** This ensures that both conditions in the loop are executed, confirming the robustness of the function's logic.

Mutation Testing

To identify potential mutations in the code (i.e., deletions, changes, or insertions of code) that would result in failures undetected by the current test set, we can utilize a mutation testing tool.

Types of Possible Mutations

Typical mutation types include:

- **Relational Operator Changes:** Modify <= to < or == to != in the conditions.
- **Logic Changes:** Remove or invert a branch in an if-statement.
- **Statement Changes:** Modify assignments or statements to assess if the effect remains undetected.

Potential Mutations and Their Effects

- **Changing the Comparison for Leftmost Point:**
  - **Mutation:** In the second loop, change p.get(i).x < p.get(min).x to p.get(i).x <= p.get(min).x.
  - **Effect:** This would cause the function to select points with the same x-coordinate as the leftmost, potentially violating the uniqueness of the minimum point.
  - **Undetected by Current Tests:** The current tests do not cover cases where multiple points share identical y and x values, which would reveal if the function mistakenly allows duplicates as the leftmost point.
- **Altering the y-Coordinate Comparison to <= in the First Loop:**
  - **Mutation:** Change p.get(i).y < p.get(min).y to p.get(i).y <= p.get(min).y in the first loop.
  - **Effect:** This would allow points with the same y-coordinate but different x-coordinates to overwrite min, potentially selecting a non-leftmost minimum point.
  - **Undetected by Current Tests:** The current test set lacks scenarios where several points have identical y-coordinates; this mutation would remain undetected. Additional tests with points sharing the same y but differing x values are necessary.
- **Removing the Check for x-coordinate in the Second Loop:**
  - **Mutation:** Remove the condition p.get(i).x < p.get(min).x in the second loop.
  - **Effect:** This would cause the function to select any point with the same minimum y-coordinate as "leftmost," disregarding its x-coordinate.
  - **Undetected by Current Tests:** The existing tests do not specifically check for points with identical y but different x values to confirm if the correct leftmost point is selected.

Additional Test Cases to Detect These Mutations

To identify these mutations, we can introduce the following test cases:

- ✓ **Detect Mutation 1:**

    1. **Test Case:** [(0, 2), (0, 2), (1, 2)]
    2. **Expected Result:** The leftmost minimum should remain (0, 2) despite duplicates.
    3. **Description:** This test will detect if the mutation with x <= mistakenly allows duplicate points.

- ✓ **Detect Mutation 2:**
    1. **Test Case:** [(1, 3), (0, 3), (4, 2)]
    2. **Expected Result:** The function should select (4, 2) as the minimum point based on the y-coordinate.
    3. **Description:** This case confirms if using <= for y comparisons inadvertently overwrites the minimum point.

- ✓ **Detect Mutation 3:**
    1. **Test Case:** [(3, 2), (2, 2), (1, 2)]
    2. **Expected Result:** The leftmost point (1, 2) should be chosen.
    3. **Description:** This will reveal if the x-coordinate check was mistakenly removed.

These additional test cases would enhance the robustness of the test suite, ensuring that mutations do not go undetected.

❖ Lab Execution

Q1) After generating the control flow graph (CFG), we verified its accuracy by comparing it with the CFG generated by the Control Flow Graph Factory Tool and the Eclipse flow graph generator.

Answer:

Control Flow Graph Factory: **YES**

Q2) Determine the minimum number of test cases needed to achieve code coverage based on the specified criteria.

Answer:

- **Statement Coverage:** 3 test cases
- **Branch Coverage:** 3 test cases
- **Basic Condition Coverage:** 3 test cases
- **Path Coverage:** 3 test cases

Summary of Minimum Test Cases:

- Total: 3 (Statement) + 3 (Branch) + 3 (Basic Condition) + 3 (Path) = **12 test cases**