Piacente Cristian 866020

Assignment A7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Derive a set of test cases that satisfy the MC/DC criterion for the program below, and show that the MC/DC criterion is indeed satisfied. Consider all decision points in the program.

Provide test cases in JUnit style: A test case starts with calling the constructor of the class with concrete values of the parameters, may (or may not) call method pointIsOnSide with concrete values of the parameters, and includes an appropriate assertion.

```
public class Rectangle {
  private final int x1; //top-left vertex
  private final int y1; //top-left vertex
  private final int x2; //bottom-right vertex
  private final int y2; //bottom-right vertex

  public Rectangle(int x1, int y1, int x2, int y2) {
    if (x1 >= x2 || y1 <= y2) {
      throw new RuntimeException("invalid vertices");
    }
    this.x1 = x1;
    this.x2 = x2;
    this.y1 = y1;
    this.y2 = y2;
  }

  public boolean pointIsOnSide(int x, int y) {
    if (((x == x1 || x == x2) && y <= y1 && y >= y2) || ((y == y1 || y == y2) && x >= x1 && x <= x2)) {
      return true;
    }
    return false;
  }
}
```

To derive a set of test cases that satisfies the MC/DC criterion for the given program, we will have 2 final tables, since there are 2 decision points in the program.

For each decision point we can follow the following steps:

- Start from the truth table of the compound condition (if pratical)
- Apply short-circuit evaluation to delete all the basic conditions which don't get evaluated
- For each basic condition c
    - Choose a row r1 and a row r2 such that:
        - r1 and r2 have different values for c
        - r1 and r2 have different outcomes
        - Each other basic condition has the same evaluation in r1 and r2 (or is not evaluated in one of the rows, in this case replace "don't care" with the value from the other row)
    - Add r1 and r2 to the test suite
- Provide a concrete JUnit test for each row in the table.

Considering the **first decision point**, let's define

A: x1 >= x2
B: y1 <= y2

The first compound condition can be written as **A || B**.

We have the following truth table, before applying short-circuit evaluation:

	A	B	Outcome
1	False	False	False
2	False	True	True
3	True	False	True
4	True	True	True

Piacente Cristian 866020

Assignment A7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Let's apply short-circuit evaluation and obtain the table below, where – means the basic condition doesn't get evaluated ("don't care") and the redundant rows have been deleted:

	A	B	Outcome
1	False	False	False
2	False	True	True
3	True	-	True

Now let's choose for each basic condition two rows that satisfy MC/DC criterion.

Basic condition A:

	A	B	Outcome
1	**False**	False	False
2	False	True	True
3	**True**	-	True

We must replace B, on the 3rd row, with False (from the 1st row).

Basic condition B:

	A	**B**	Outcome
1	False	**False**	False
2	False	**True**	True
3	True	False	True

Every row has been used for satisfying MC/DC criterion, so we can't exclude any combination of truth values.

We can provide a concrete JUnit test for each row, w.r.t the definition of the basic condition A, B given before (also shown below)

A: x1 >= x2
B: y1 <= y2

Here is the **JUnit test suite for the first decision point**:

1. x1 < x2 && y1 > y2
   → We can call the constructor using the concrete parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 and then include an appropriate assertion. Here's the **JUnit test**:
   ```
   int x1 = 0;
   int y1 = 1;
   int x2 = 2;
   int y2 = 0;
   Rectangle rect = new Rectangle(x1, y1, x2, y2);
   assertTrue(x1 < x2 && y1 > y2);
   ```
2. x1 < x2 && y1 <= y2
   → We can call the constructor using the concrete parameters x1 = 0, y1 = 0, x2 = 1, y2 = 1 and then include an appropriate assertion. Here's the **JUnit test**:
   ```
   int x1 = 0;
   int y1 = 0;
   int x2 = 1;
   int y2 = 1;
   Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception
   assertTrue(x1 < x2 && y1 <= y2);
   ```
   → When executing the test case, the constructor will throw an **exception**, since the outcome of the compound condition, in the if statement, is true
3. x1 >= x2 && y1 > y2
   → We can call the constructor using the concrete parameters x1 = 1, y1 = 1, x2 = 0, y2 = 0 and then include an appropriate assertion. Here's the **JUnit test**:
   ```
   int x1 = 1;
   int y1 = 1;
   ```

```
int x2 = 0;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception
assertTrue(x1 >= x2 && y1 > y2);
```

➔ When executing the test case, the constructor will throw an **exception**, since the outcome of the compound condition, in the if statement, is true

To summarize, this is the **resulting table for the first decision point**:

Test case number	x1 >= x2	y1 <= y2	Outcome
1	**False**	**False**	False
2	False	**True**	True
3	**True**	False	True

We can now consider the **second decision point**.

Let's define

C: x == x1
D: x == x2
E: y <= y1
F: y >= y2
G: y == y1
H: y == y2
I: x >= x1
J: x <= x2

The second compound condition can be written as **((C || D) && E && F) || ((G || H) && I && J)**.

It's not pratical to start from the 2^8 = 256 combinations, so for this decision point we'll start from the short-circuit evaluated combinations, then we'll exclude the unsatisfiable ones and finally apply MC/DC criterion.

To avoid forgetting about some combinations, let's delve into an organized analysis of the compound condition:

- Outcome = True
  - ((C || D) && E && F) = True
    5. C = False, D = True, E = True, F = True
    6. C = True, E = True, F = True
  - ((C || D) && E && F) = False, ((G || H) && I && J) = True
    - C = False, D = False,
      ((G || H) && I && J) = True
      7. C = False, D = False,
         G = False, H = True, I = True, J = True
      8. C = False, D = False,
         G = True, I = True, J = True
    - (C || D) = True, E = False,
      ((G || H) && I && J) = True
      - C = False, D = True, E = False,
        ((G || H) && I && J) = True
        9. C = False, D = True, E = False,
           G = False, H = True, I = True, J = True
        10. C = False, D = True, E = False,
            G = True, I = True, J = True
      - C = True, E = False,
        ((G || H) && I && J) = True
        11. C = True, E = False,
            G = False, H = True, I = True, J = True
        12. C = True, E = False,
            G = True, I = True, J = True

        - (C || D) = True, E = True, F = False,
          ((G || H) && I && J) = True
            - C = False, D = True, E = True, F = False,
              ((G || H) && I && J) = True
                13. C = False, D = True, E = True, F = False,
                    G = False, H = True, I = True, J = True
                14. C = False, D = True, E = True, F = False,
                    G = True, I = True, J = True
            - C = True, E = True, F = False,
              ((G || H) && I && J) = True
                15. C = True, E = True, F = False,
                    G = False, H = True, I = True, J = True
                16. C = True, E = True, F = False,
                    G = True, I = True, J = True

- Outcome = False
    - ((C || D) && E && F) = False, ((G || H) && I && J) = False
        - C = False, D = False,
          ((G || H) && I && J) = False
            17. C = False, D = False,
                G = False, H = False
            - C = False, D = False,
              (G || H) = True, I = False
                18. C = False, D = False,
                    G = False, H = True, I = False
                19. C = False, D = False,
                    G = True, I = False
            - C = False, D = False,
              (G || H) = True, I = True, J = False
                20. C = False, D = False,
                    G = False, H = True, I = True, J = False
                21. C = False, D = False,
                    G = True, I = True, J = False
        - (C || D) = True, E = False,
          ((G || H) && I && J) = False
            - C = False, D = True, E = False,
              ((G || H) && I && J) = False
                22. C = False, D = True, E = False,
                    G = False, H = False
                - C = False, D = True, E = False,
                  (G || H) = True, I = False
                    23. C = False, D = True, E = False,
                        G = False, H = True, I = False
                    24. C = False, D = True, E = False,
                        G = True, I = False
                - C = False, D = True, E = False,
                  (G || H) = True, I = True, J = False
                    25. C = False, D = True, E = False,
                        G = False, H = True, I = True, J = False
                    26. C = False, D = True, E = False,
                        G = True, I = True, J = False
            - C = True, E = False,
              ((G || H) && I && J) = False
                27. C = True, E = False,
                    G = False, H = False
                - C = True, E = False,
                  (G || H) = True, I = False
                    28. C = True, E = False,
                        G = False, H = True, I = False

Piacente Cristian 866020

Assignment A7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

29. C = True, E = False,
    G = True, I = False

- C = True, E = False,
  (G || H) = True, I = True, J = False
  30. C = True, E = False,
      G = False, H = True, I = True, J = False
  31. C = True, E = False,
      G = True, I = True, J = False

- (C || D) = True, E = True, F = False,
  ((G || H) && I && J) = False
  - C = False, D = True, E = True, F = False,
    ((G || H) && I && J) = False
    32. C = False, D = True, E = True, F = False,
        G = False, H = False
    - C = False, D = True, E = True, F = False,
      (G || H) = True, I = False
      33. C = False, D = True, E = True, F = False,
          G = False, H = True, I = False
      34. C = False, D = True, E = True, F = False,
          G = True, I = False
    - C = False, D = True, E = True, F = False,
      (G || H) = True, I = True, J = False
      35. C = False, D = True, E = True, F = False,
          G = False, H = True, I = True, J = False
      36. C = False, D = True, E = True, F = False,
          G = True, I = True, J = False
  - C = True, E = True, F = False,
    ((G || H) && I && J) = False
    37. C = True, E = True, F = False,
        G = False, H = False
    - C = True, E = True, F = False,
      (G || H) = True, I = False
      38. C = True, E = True, F = False,
          G = False, H = True, I = False
      39. C = True, E = True, F = False,
          G = True, I = False
    - C = True, E = True, F = False,
      (G || H) = True, I = True, J = False
      40. C = True, E = True, F = False,
          G = False, H = True, I = True, J = False
      41. C = True, E = True, F = False,
          G = True, I = True, J = False

This is the short-circuit evaluation of ((C || D) && E && F) || ((G || H) && I && J) (it continues on the next page):

	C	D	E	F	G	H	I	J	Outcome
5	False	True	True	True	-	-	-	-	True
6	True	-	True	True	-	-	-	-	True
7	False	False	-	-	False	True	True	True	True
8	False	False	-	-	True	-	True	True	True
9	False	True	False	-	False	True	True	True	True
10	False	True	False	-	True	-	True	True	True
11	True	-	False	-	False	True	True	True	True
12	True	-	False	-	True	-	True	True	True
13	False	True	True	False	False	True	True	True	True
14	False	True	True	False	True	-	True	True	True
15	True	-	True	False	False	True	True	True	True
16	True	-	True	False	True	-	True	True	True
17	False	False	-	-	False	False	-	-	False
18	False	False	-	-	False	True	False	-	False

19	False	False	-	-	True	-	False	-	False
20	False	False	-	-	False	True	True	False	False
21	False	False	-	-	True	-	True	False	False
22	False	True	False	-	False	False	-	-	False
23	False	True	False	-	False	True	False	-	False
24	False	True	False	-	True	-	False	-	False
25	False	True	False	-	False	True	True	False	False
26	False	True	False	-	True	-	True	False	False
27	True	-	False	-	False	False	-	-	False
28	True	-	False	-	False	True	False	-	False
29	True	-	False	-	True	-	False	-	False
30	True	-	False	-	False	True	True	False	False
31	True	-	False	-	True	-	True	False	False
32	False	True	True	False	False	False	-	-	False
33	False	True	True	False	False	True	False	-	False
34	False	True	True	False	True	-	False	-	False
35	False	True	True	False	False	True	True	False	False
36	False	True	True	False	True	-	True	False	False
37	True	-	True	False	False	False	-	-	False
38	True	-	True	False	False	True	False	-	False
39	True	-	True	False	True	-	False	-	False
40	True	-	True	False	False	True	True	False	False
41	True	-	True	False	True	-	True	False	False

Before continuing

- we must eliminate the combinations which are contradictions, e.g. if we have y > y1 and y == y1
- we must fill some “don’t care” values to only obtain satisfiable compound conditions: for instance, if we have x == x2 and the basic condition x <= x2 is not evaluated (“don’t care”), then x <= x2 must be True.

Let’s also replace the basic conditions C, D, E, F, G, H, I, J with their definition given before (also shown below)

C: x == x1
D: x == x2
E: y <= y1
F: y >= y2
G: y == y1
H: y == y2
I: x >= x1
J: x <= x2

Here is the new table (it continues on the next page), which we must consider:

	x == x1	x == x2	y <= y1	y >= y2	y == y1	y == y2	x >= x1	x <= x2	Outcome
5	False	True	True	True	-	-	-	True	True
6	True	-	True	True	-	-	True	-	True
7	False	False	-	True	False	True	True	True	True
8	False	False	True	-	True	-	True	True	True
9	False	True	False	True	False	True	True	True	True
11	True	-	False	True	False	True	True	True	True
14	False	True	True	False	True	False	True	True	True
16	True	-	True	False	True	False	True	True	True
17	False	False	-	-	False	False	-	-	False
18	False	False	-	True	False	True	False	-	False
19	False	False	True	-	True	-	False	-	False
20	False	False	-	True	False	True	True	False	False
21	False	False	True	-	True	-	True	False	False
22	False	True	False	-	False	False	-	True	False
23	False	True	False	True	False	True	False	True	False
27	True	-	False	-	False	False	True	-	False
30	True	False	False	True	False	True	True	False	False
32	False	True	True	False	False	False	-	True	False

Piacente Cristian 866020

Assignment A7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

34	False	True	True	False	True	False	False	True	False
37	True	-	True	False	False	False	True	-	False
41	True	False	True	False	True	False	True	False	False

Now, let's choose for each basic condition two rows that satisfy MC/DC criterion.
Only the two chosen rows, for each basic condition, will be shown below.
Each "don't care" value is replaced with the concrete value from the other row.

Basic condition x == x1:

	x == x1	x == x2	y <= y1	y >= y2	y == y1	y == y2	x >= x1	x <= x2	Outcome
6	**True**	False	True	True	False	True	True	False	True
20	**False**	False	True	True	False	True	True	False	False

Basic condition x == x2:

	x == x1	**x == x2**	y <= y1	y >= y2	y == y1	y == y2	x >= x1	x <= x2	Outcome
5'	False	**True**	True	True	False	True	False	True	True
18	False	**False**	True	True	False	True	False	True	False

Basic condition y <= y1:

	x == x1	x == x2	**y <= y1**	y >= y2	y == y1	y == y2	x >= x1	x <= x2	Outcome
5'	False	True	**True**	True	False	True	False	True	True
23	False	True	**False**	True	False	True	False	True	False

Basic condition y >= y2:

	x == x1	x == x2	y <= y1	**y >= y2**	y == y1	y == y2	x >= x1	x <= x2	Outcome
5"	False	True	True	**True**	True	False	False	True	True
34	False	True	True	**False**	True	False	False	True	False

Basic condition y == y1:

	x == x1	x == x2	y <= y1	y >= y2	**y == y1**	y == y2	x >= x1	x <= x2	Outcome
14	False	True	True	False	**True**	False	True	True	True
32	False	True	True	False	**False**	False	True	True	False

Basic condition y == y2:

	x == x1	x == x2	y <= y1	y >= y2	y == y1	**y == y2**	x >= x1	x <= x2	Outcome
9	False	True	False	True	False	**True**	True	True	True
22	False	True	False	True	False	**False**	True	True	False

Basic condition x >= x1:

	x == x1	x == x2	y <= y1	y >= y2	y == y1	y == y2	**x >= x1**	x <= x2	Outcome
9	False	True	False	True	False	True	**True**	True	True
23	False	True	False	True	False	True	**False**	True	False

Basic condition x <= x2:

	x == x1	x == x2	y <= y1	y >= y2	y == y1	y == y2	x >= x1	**x <= x2**	Outcome
7	False	False	True	True	False	True	True	**True**	True
20	False	False	True	True	False	True	True	**False**	False

Piacente Cristian 866020

Assignment A7 – MCDC

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Here is the **resulting table for the second decision point**:

Test case number	x == x1	x == x2	y <= y1	y >= y2	y == y1	y == y2	x >= x1	x <= x2	Outcome
5’	False	**True**	**True**	True	False	True	False	True	True
5”	False	True	True	**True**	True	False	False	True	True
6	**True**	False	True	True	False	True	True	False	True
7	False	False	True	True	False	True	True	**True**	True
9	False	True	False	True	False	**True**	**True**	True	True
14	False	True	True	False	**True**	False	True	True	True
18	False	**False**	True	True	False	True	False	True	False
20	**False**	False	True	True	False	True	True	**False**	False
22	False	True	False	True	False	**False**	True	True	False
23	False	True	**False**	True	False	True	**False**	True	False
32	False	True	True	False	**False**	False	True	True	False
34	False	True	True	**False**	True	False	False	True	False

It means we only need **12 tests**.

We can now provide a concrete JUnit test for each row.

To conclude, here is the **JUnit test suite for the second decision point**:

5’. x != x1 && x == x2 && y <= y1 && y >= y2 && y != y1 && y == y2 && x < x1 && x <= x2

→ We can simplify the condition by rewriting it as x < x1 && x == x2 && y < y1 && y == y2, which means x1 > x2 && y1 > y2 && x == x2 && y == y2

→ We can exploit the constructor parameters x1 = 1, y1 = 1, x2 = 0, y2 = 0 **from the JUnit test case 3.**, call the method pointIsOnSide with the parameters x = x2, y = y2 and include an appropriate assertion. Here’s the **JUnit test**:

```
int x1 = 1;
int y1 = 1;
int x2 = 0;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception like in the test case 3.
int x = x2;
int y = y2;
assertTrue(rect.pointIsOnSide(x, y));
```

→ When executing the test case, the constructor will throw an **exception**, since x1 >= x2

5”. x != x1 && x == x2 && y <= y1 && y >= y2 && y == y1 && y != y2 && x < x1 && x <= x2

→ We can simplify the condition by rewriting it as x < x1 && x == x2 && y == y1 && y > y2, which means x1 > x2 && y1 > y2 && x == x2 && y == y1

→ We can exploit the constructor parameters x1 = 1, y1 = 1, x2 = 0, y2 = 0 **from the JUnit test case 3.**, call the method pointIsOnSide with the parameters x = x2, y = y1 and include an appropriate assertion. Here’s the **JUnit test**:

```
int x1 = 1;
int y1 = 1;
int x2 = 0;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception like in the test case 3.
int x = x2;
int y = y1;
assertTrue(rect.pointIsOnSide(x, y));
```

→ When executing the test case, the constructor will throw an **exception**, since x1 >= x2

6. x == x1 && x != x2 && y <= y1 && y >= y2 && y != y1 && y == y2 && x >= x1 && x > x2

→ We can simplify the condition by rewriting it as x == x1 && x > x2 && y < y1 && y == y2, which means x1 > x2 && y1 > y2 && x == x1 && y == y2

→ We can exploit the constructor parameters x1 = 1, y1 = 1, x2 = 0, y2 = 0 **from the JUnit test case 3.**, call the method pointIsOnSide with the parameters x = x1, y = y2 and include an appropriate assertion. Here’s the **JUnit test**:

```
int x1 = 1;
int y1 = 1;
```

int x2 = 0;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception like in the test case 3.
int x = x1;
int y = y2;
assertTrue(rect.pointIsOnSide(x, y));

➜ When executing the test case, the constructor will throw an **exception**, since x1 >= x2

7. x != x1 && x != x2 && y <= y1 && y >= y2 && y != y1 && y == y2 && x >= x1 && x <= x2

➜ We can simplify the condition by rewriting it as x > x1 && x < x2 && y < y1 && y == y2, which means x1 < x2 && y1 > y2 && x > x1 && x < x2 && y == y2

➜ We can exploit the constructor parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 **from the JUnit test case 1.**, call the method pointIsOnSide with the parameters x = 1, y = y2 and include an appropriate assertion. Here's the **JUnit test**:

int x1 = 0;
int y1 = 1;
int x2 = 2;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2);
int x = 1;
int y = y2;
assertTrue(rect.pointIsOnSide(x, y));

9. x != x1 && x == x2 && y > y1 && y >= y2 && y != y1 && y == y2 && x >= x1 && x <= x2

➜ We can simplify the condition by rewriting it as x > x1 && x == x2 && y > y1 && y == y2, which means x1 < x2 && y1 < y2 && x == x2 && y == y2

➜ We can exploit the constructor parameters x1 = 0, y1 = 0, x2 = 1, y2 = 1 **from the JUnit test case 2.**, call the method pointIsOnSide with the parameters x = x2, y = y2 and include an appropriate assertion. Here's the **JUnit test**:

int x1 = 0;
int y1 = 0;
int x2 = 1;
int y2 = 1;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception like in the test case 2.
int x = x2;
int y = y2;
assertTrue(rect.pointIsOnSide(x, y));

➜ When executing the test case, the constructor will throw an **exception**, since y1 <= y2

14. x != x1 && x == x2 && y <= y1 && y < y2 && y == y1 && y != y2 && x >= x1 && x <= x2

➜ We can simplify the condition by rewriting it as x > x1 && x == x2 && y == y1 && y < y2, which means x1 < x2 && y1 < y2 && x == x2 && y == y1

➜ We can exploit the constructor parameters x1 = 0, y1 = 0, x2 = 1, y2 = 1 **from the JUnit test case 2.**, call the method pointIsOnSide with the parameters x = x2, y = y1 and include an appropriate assertion. Here's the **JUnit test**:

int x1 = 0;
int y1 = 0;
int x2 = 1;
int y2 = 1;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception like in the test case 2.
int x = x2;
int y = y1;
assertTrue(rect.pointIsOnSide(x, y));

➜ When executing the test case, the constructor will throw an **exception**, since y1 <= y2

18. x != x1 && x != x2 && y <= y1 && y >= y2 && y != y1 && y == y2 && x < x1 && x <= x2

➜ We can simplify the condition by rewriting it as x < x1 && x < x2 && y < y1 && y == y2, which means y1 > y2 && x < x1 && x < x2 && y == y2

➜ We can exploit the constructor parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 **from the JUnit test case 1.**, call the method pointIsOnSide with the parameters x = -1, y = y2 and include an appropriate assertion. Here's the **JUnit test**:

int x1 = 0;

```
int y1 = 1;
int x2 = 2;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2);
int x = -1;
int y = y2;
assertFalse(rect.pointIsOnSide(x, y));
```

20. x != x1 && x != x2 && y <= y1 && y >= y2 && y != y1 && y == y2 && x >= x1 && x > x2

    → We can simplify the condition by rewriting it as x > x1 && x > x2 && y < y1 && y == y2,
    which means y1 > y2 && x > x1 && x > x2 && y == y2

    → We can exploit the constructor parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 **from the JUnit test case 1.**,
    call the method pointIsOnSide with the parameters x = 3, y = y2 and include an appropriate assertion.
    Here's the **JUnit test**:

```
int x1 = 0;
int y1 = 1;
int x2 = 2;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2);
int x = 3;
int y = y2;
assertFalse(rect.pointIsOnSide(x, y));
```

22. x != x1 && x == x2 && y > y1 && y >= y2 && y != y1 && y != y2 && x >= x1 && x <= x2

    → We can simplify the condition by rewriting it as x > x1 && x == x2 && y > y1 && y > y2,
    which means x1 < x2 && y > y1 && y > y2 && x == x2

    → We can exploit the constructor parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 **from the JUnit test case 1.**,
    call the method pointIsOnSide with the parameters x = x2, y = 2 and include an appropriate assertion.
    Here's the **JUnit test**:

```
int x1 = 0;
int y1 = 1;
int x2 = 2;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2);
int x = x2;
int y = 2;
assertFalse(rect.pointIsOnSide(x, y));
```

23. x != x1 && x == x2 && y > y1 && y >= y2 && y != y1 && y == y2 && x < x1 && x <= x2

    → We can simplify the condition by rewriting it as x < x1 && x == x2 && y > y1 && y == y2,
    which means x1 > x2 && y1 < y2 && x == x2 && y == y2

    → We can call the constructor using the parameters x1 = 1, y1 = 0, x2 = 0, y2 = 1,
    call the method pointIsOnSide with the parameters x = x2, y = y2 and include an appropriate assertion.
    Here's the **JUnit test**:

```
int x1 = 1;
int y1 = 0;
int x2 = 0;
int y2 = 1;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception
int x = x2;
int y = y2;
assertFalse(rect.pointIsOnSide(x, y));
```

    → When executing the test case, the constructor will throw an **exception**, since x1 >= x2

32. x != x1 && x == x2 && y <= y1 && y < y2 && y != y1 && y != y2 && x >= x1 && x <= x2

    → We can simplify the condition by rewriting it as x > x1 && x == x2 && y < y1 && y < y2,
    which means x1 < x2 && y < y1 && y < y2 && x == x2

    → We can exploit the constructor parameters x1 = 0, y1 = 1, x2 = 2, y2 = 0 **from the JUnit test case 1.**,
    call the method pointIsOnSide with the parameters x = x2, y = -1 and include an appropriate assertion.
    Here's the **JUnit test**:

```
int x1 = 0;
int y1 = 1;
```

```
int x2 = 2;
int y2 = 0;
Rectangle rect = new Rectangle(x1, y1, x2, y2);
int x = x2;
int y = -1;
assertFalse(rect.pointIsOnSide(x, y));
```

34. x != x1 && x == x2 && y <= y1 && y < y2 && y == y1 && y != y2 && x < x1 && x <= x2

    ➜ We can simplify the condition by rewriting it as x < x1 && x == x2 && y == y1 && y < y2, which means x1 > x2 && y1 < y2 && x == x2 && y == y1

    ➜ We can exploit the constructor parameters x1 = 1, y1 = 0, x2 = 0, y2 = 1 **from the JUnit test case 23.**, call the method pointIsOnSide with the parameters x = x2, y = y1 and include an appropriate assertion. Here's the **JUnit test**:

```
int x1 = 1;
int y1 = 0;
int x2 = 0;
int y2 = 1;
Rectangle rect = new Rectangle(x1, y1, x2, y2); // exception
int x = x2;
int y = y1;
assertFalse(rect.pointIsOnSide(x, y));
```

    ➜ When executing the test case, the constructor will throw an **exception**, since x1 >= x2.