

ITP2200

Introduction to Software Testing

Bogdan Marculescu

Lecture 3

Graph Coverage

Logic Coverage

So, more programming this week?



What do we remember from last time?



We mentioned “Coverage”

```
4  public class TriangleClassifier {  
5      @  
6          public static String classify(int a, int b, int c) {  
7              if (a <= 0 || b <= 0 || c <= 0) {  
8                  return "NOT_A_TRIANGLE";  
9              }  
10             if (a == b && b == c) {  
11                 return "EQUILATERAL";  
12             }  
13             int max = Math.max(a, Math.max(b, c));  
14             if ((max == a && max - b - c >= 0) ||  
15                 (max == b && max - a - c >= 0) ||  
16                 (max == c && max - a - b >= 0)) {  
17                 return "NOT_A_TRIANGLE";  
18             }  
19             if (a == b || b == c || a == c) {  
20                 return "ISOSCELES";  
21             } else {  
22                 return "SCALENE";  
23             }  
24         }  
25     }  
26 }  
27 }  
28 }  
29 }
```

```
@Test  
public void testScalene() throws Exception {  
    int a = 5;  
    int b = 7;  
    int c = 9;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase( anotherString: "SCALENE"));  
}  
  
@Test  
public void testIsosceles() throws Exception{  
    int a = 5;  
    int b = 5;  
    int c = 7;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase( anotherString: "ISOSCELES"));  
    assertFalse(result.equalsIgnoreCase( anotherString: "SCALENE"));  
}
```

Code is at

<https://github.com/bogdanmarculescu/itp2200>

We mentioned “Coverage”

The screenshot shows an IDE interface with several tabs at the top: TriangleClassifierTest.java, TriangleClassifier.java, and SortingHelper.java. The main area displays the Java code for TriangleClassifier.java. The code defines a class that classifies triangles based on their side lengths. A coverage analysis window is overlaid on the right side of the screen, titled 'Coverage: TriangleClassifierTest'. It shows a summary: '100% classes, 87% lines covered in package 'e...'' and a detailed table:

Element	Class, %	Method, %	Line, %
tests	100% (1/1)	100% (2/2)	100% (14/14)
TriangleClassifier	100% (1/1)	100% (1/1)	70% (7/10)

The code itself includes various if statements and return statements for different triangle types.

Interpreting Coverage:

What happens to code that is not executed?

How should we change our test to cover those lines as well?

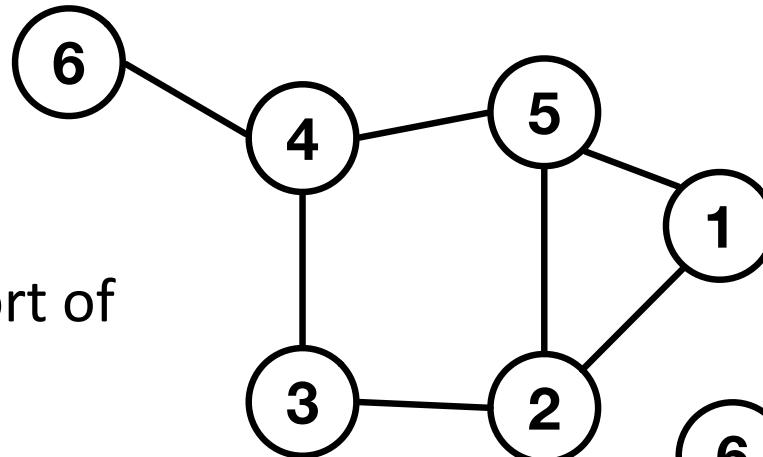
A (very brief) overview of graphs

A graph:

Structure

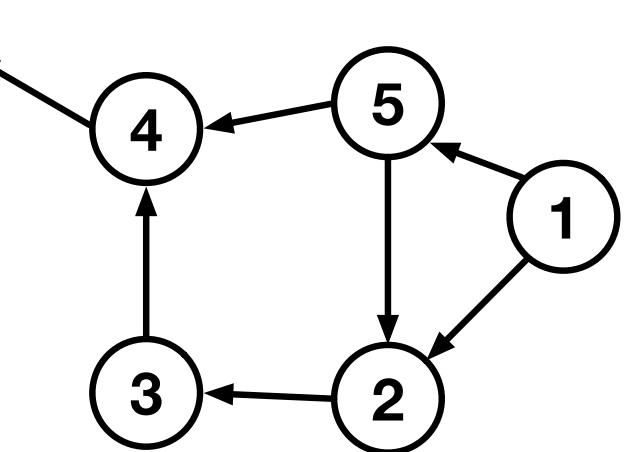
Nodes (Vertices, points)

- Objects that have some sort of connection



Edges

- Connection between them (can be directional or not)



Path

- A way of “moving” through the graph

Code as a Graph

Abstraction of code:

Nodes – lines of code

Edges - transitions between lines (i.e. what gets executed)

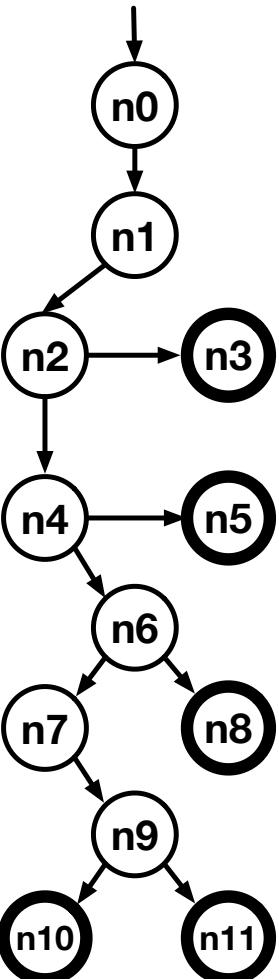
```
4  public class TriangleClassifier {  
5      @ ...  
6      public static String classify(int a, int b, int c) {  
7          if (a <= 0 || b <= 0 || c <= 0) {  
8              return "NOT_A_TRIANGLE";  
9          }  
10         if (a == b && b == c) {  
11             return "EQUILATERAL";  
12         }  
13         int max = Math.max(a, Math.max(b, c));  
14         if ((max == a && max - b - c >= 0) ||  
15             (max == b && max - a - c >= 0) ||  
16             (max == c && max - a - b >= 0)) {  
17                 return "NOT_A_TRIANGLE";  
18             }  
19             if (a == b || b == c || a == c) {  
20                 return "ISOSCELES";  
21             } else {  
22                 return "SCALENE";  
23             }  
24         }  
25     }  
26 }
```

Code as a Graph

Abstraction of code:

Nodes – lines of code

Edges - transitions between lines (i.e. what gets executed)



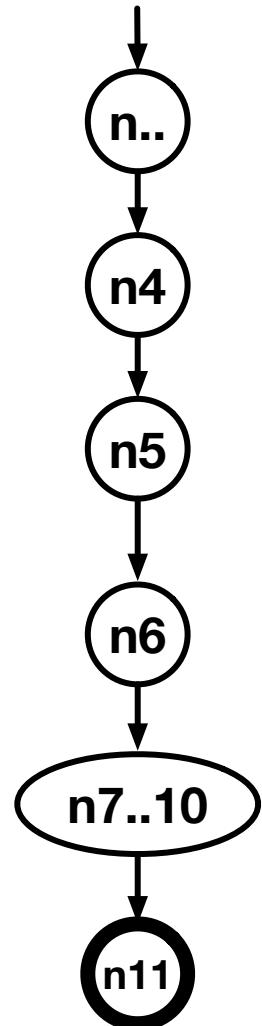
```
4  public class TriangleClassifier {  
5      @ ...  
6          public static String classify(int a, int b, int c) {  
7              if (a <= 0 || b <= 0 || c <= 0) {  
8                  return "NOT_A_TRIANGLE";  
9              }  
10             if (a == b && b == c) {  
11                 return "EQUILATERAL";  
12             }  
13             int max = Math.max(a, Math.max(b, c));  
14             if ((max == a && max - b - c >= 0) ||  
15                 (max == b && max - a - c >= 0) ||  
16                 (max == c && max - a - b >= 0)) {  
17                 return "NOT_A_TRIANGLE";  
18             }  
19             if (a == b || b == c || a == c) {  
20                 return "ISOSCELES";  
21             } else {  
22                 return "SCALENE";  
23             }  
24         }  
25     }  
26 }
```

Code as a Graph

Abstraction of code:

Nodes – lines of code

Edges - transitions between lines (i.e. what gets executed)



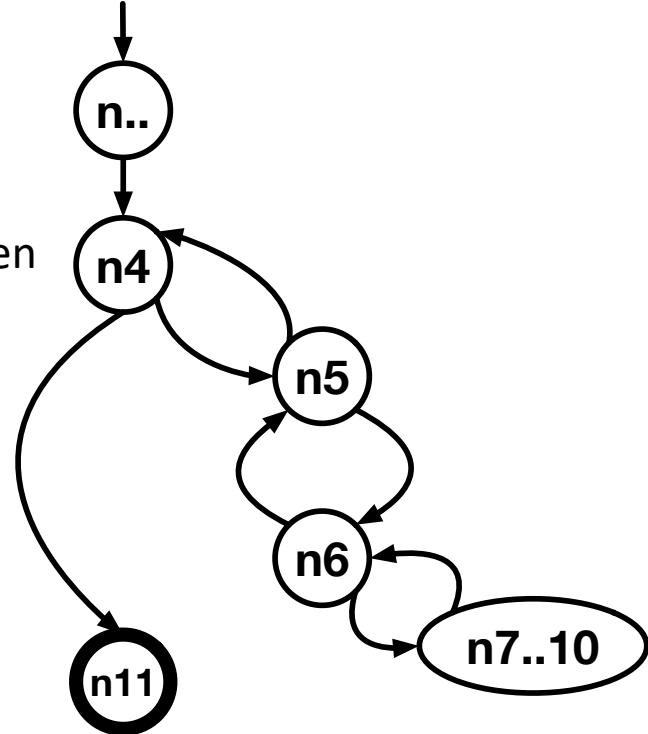
```
public class SortingHelper {  
    public static int[] sortArray(int[] array){  
        int n = array.length;  
        int[] result = array.clone();  
        int temp = 0;  
        for(int i=0; i < n; i++) // Looping through the array length  
        {  
            for(int j=1; j < (n-i); j++)  
            {  
                if(result[j-1] > result[j])  
                {  
                    //swap elements  
                    temp = result[j-1];  
                    result[j-1] = result[j];  
                    result[j] = temp;  
                }  
            }  
        }  
        return result;  
    }  
}
```

Code as a Graph

Abstraction of code:

Nodes – lines of code

Edges - transitions between lines (i.e. what gets executed)



```
public class SortingHelper {  
    public static int[] sortArray(int[] array){  
        int n = array.length;  
        int[] result = array.clone();  
        int temp = 0;  
        for(int i=0; i < n; i++) // Looping through the array length  
        {  
            for(int j=1; j < (n-i); j++)  
            {  
                if(result[j-1] > result[j])  
                {  
                    //swap elements  
                    temp = result[j-1];  
                    result[j-1] = result[j];  
                    result[j] = temp;  
                }  
            }  
        }  
        return result;  
    }  
}
```

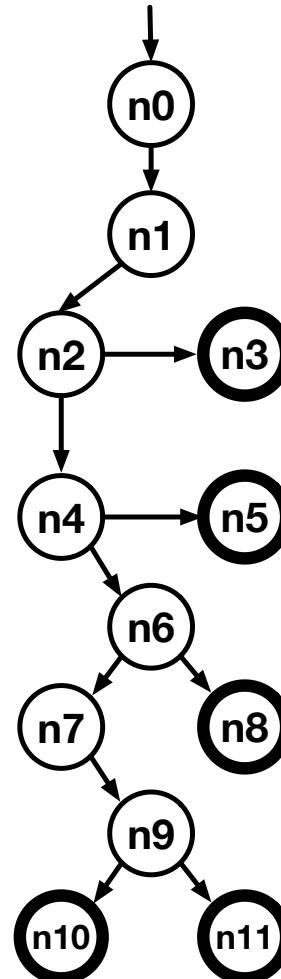
Back to our code

```
4  public class TriangleClassifier {  
5      @  
6      public static String classify(int a, int b, int c) {  
7          if (a <= 0 || b <= 0 || c <= 0) {  
8              return "NOT_A_TRIANGLE";  
9          }  
10         if (a == b && b == c) {  
11             return "EQUILATERAL";  
12         }  
13         int max = Math.max(a, Math.max(b, c));  
14         if ((max == a && max - b - c >= 0) ||  
15             (max == b && max - a - c >= 0) ||  
16             (max == c && max - a - b >= 0)) {  
17             return "NOT_A_TRIANGLE";  
18         }  
19         if (a == b || b == c || a == c) {  
20             return "ISOSCELES";  
21         } else {  
22             return "SCALENE";  
23         }  
24     }  
25 }  
26 }  
27 }  
28 }  
29 }
```

```
@Test  
public void testScalene() throws Exception {  
    int a = 5;  
    int b = 7;  
    int c = 9;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase( anotherString: "SCALENE"));  
}  
  
@Test  
public void testIsosceles() throws Exception{  
    int a = 5;  
    int b = 5;  
    int c = 7;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase( anotherString: "ISOSCELES"));  
    assertFalse(result.equalsIgnoreCase( anotherString: "SCALENE"));  
}
```

Code as a Graph

```
4  public class TriangleClassifier {  
5      @  
6          public static String classify(int a, int b, int c) {  
7              if (a <= 0 || b <= 0 || c <= 0) {  
8                  return "NOT_A_TRIANGLE";  
9              }  
10             if (a == b && b == c) {  
11                 return "EQUILATERAL";  
12             }  
13             int max = Math.max(a, Math.max(b, c));  
14             if ((max == a && max - b - c >= 0) ||  
15                 (max == b && max - a - c >= 0) ||  
16                 (max == c && max - a - b >= 0)) {  
17                     return "NOT_A_TRIANGLE";  
18                 }  
19                 if (a == b || b == c || a == c) {  
20                     return "ISOSCELES";  
21                 } else {  
22                     return "SCALENE";  
23                 }  
24             }  
25         }  
26     }  
27 }  
28 }
```

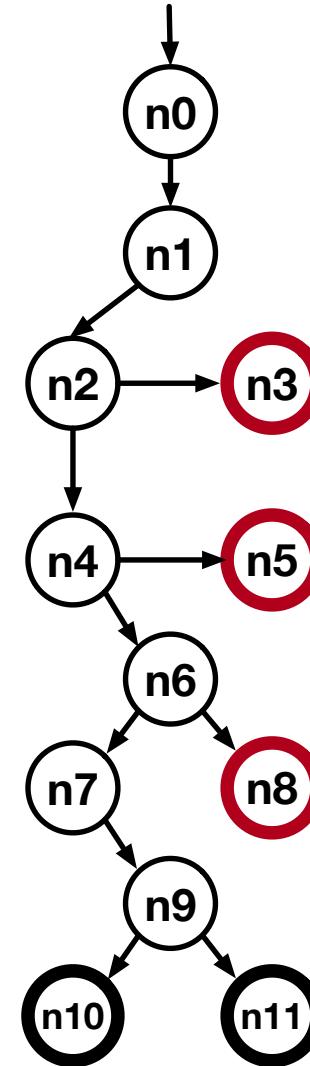


```
@Test  
public void testScalene() throws Exception {  
    int a = 5;  
    int b = 7;  
    int c = 9;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase("SCALENE"));  
}  
  
@Test  
public void testIsosceles() throws Exception{  
    int a = 5;  
    int b = 5;  
    int c = 7;  
  
    String result = TriangleClassifier.classify(a, b, c);  
    assertTrue(result.equalsIgnoreCase("ISOSCELES"));  
    assertFalse(result.equalsIgnoreCase("SCALENE"));  
}
```

We mentioned “Coverage”

The screenshot shows an IDE interface with several tabs at the top: TriangleClassifierTest.java, TriangleClassifier.java, and SortingHelper.java. The main area displays the code for TriangleClassifier.java. The code implements a triangle classifier based on side lengths a, b, and c. It handles invalid inputs (non-positive), equilateral triangles, isosceles triangles, scalene triangles, and non-triangles. A coverage analysis window is overlaid on the right side of the code editor, showing a summary: 100% classes, 87% lines covered in package 'e...'. Below this, a detailed table provides coverage metrics for tests, methods, and individual lines:

Element	Class, %	Method, %	Line, %
tests	100% (1/1)	100% (2/2)	100% (14/14)
TriangleClassifierTest	100% (1/1)	100% (1/1)	70% (7/10)



We mentioned “Coverage”

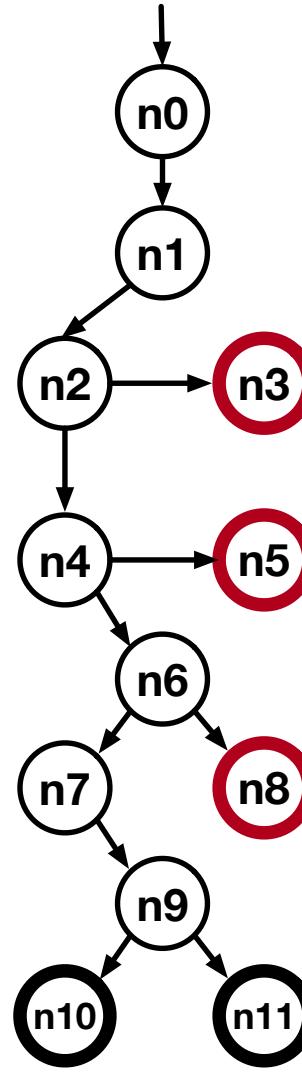
TriangleClassifierTest.java

```
1 package ex03;
2
3 public class TriangleClassifier {
4     @
5         public static String classify(int a, int b, int c) {
6             if (a <= 0 || b <= 0 || c <= 0) {
7                 return "NOT_A_TRIANGLE";
8             }
9
10            if (a == b && b == c) {
11                return "EQUILATERAL";
12            }
13
14            int max = Math.max(a, Math.max(b, c));
15
16            if ((max == a && max - b - c >= 0) ||
17                (max == b && max - a - c >= 0) ||
18                (max == c && max - a - b >= 0)) {
19                    return "NOT_A_TRIANGLE";
20                }
21
22            if (a == b || b == c || a == c) {
23                return "ISOSCELES";
24            } else {
25                return "SCALENE";
26            }
27        }
28    }
29 }
```

TriangleClassifier

Coverage: TriangleClassifierTest

Element	Class, %	Method, %	Line, %
tests	100% (1/1)	100% (2/2)	100% (14/14)
TriangleClassifier	100% (1/1)	100% (1/1)	70% (7/10)



How can we cover
the remaining
nodes?

Are all nodes
reachable?

(Should we?
Is it worth doing?)

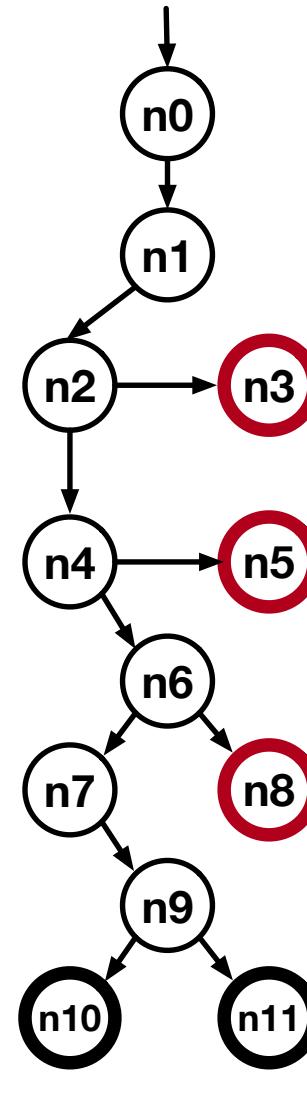
Code!!!

The screenshot shows an IDE interface with three tabs: TriangleClassifierTest.java, TriangleClassifier.java, and SortingHelper.java. The TriangleClassifier.java tab contains the following code:

```
1 package ex03;
2
3 public class TriangleClassifier {
4     @
5         public static String classify(int a, int b, int c) {
6             if (a <= 0 || b <= 0 || c <= 0) {
7                 return "NOT_A_TRIANGLE";
8             }
9
10            if (a == b && b == c) {
11                return "EQUILATERAL";
12            }
13
14            int max = Math.max(a, Math.max(b, c));
15
16            if ((max == a && max - b - c >= 0) ||
17                (max == b && max - a - c >= 0) ||
18                (max == c && max - a - b >= 0)) {
19                    return "NOT_A_TRIANGLE";
20                }
21
22            if (a == b || b == c || a == c) {
23                return "ISOSCELES";
24            } else {
25                return "SCALENE";
26            }
27        }
28    }
29}
```

The Coverage report shows 100% classes, 87% lines covered in package 'e...', with a table:

Element	Class, %	Method, %	Line, %
tests	100% (1/1)	100% (2/2)	100% (14/14)
TriangleClassifier	100% (1/1)	100% (1/1)	70% (7/10)



Let's add tests to cover the remaining nodes!



Høyskolen
Kristiania



Questions so far?

Exercise for the Seminar



Rooms:

A3-01

A4-03

A5-10

A5-18

Exercise for the Seminar



Self-organized.

In repo:

Stats.java - Contains code for some basic stats on a group of numbers.

tests/StatsTest.java - Contains scaffolding for tests.

SO:

Draw up the graph for Stats.java

Write up what paths you want to cover (ideally all nodes will be covered)

Write up the tests for those paths.

Run (with coverage)!

Exercise for the Seminar



When done

PatternMatcher.java - Contains code for some string pattern matching.

tests/PatTest.java - Contains scaffolding for tests.

SO:

Draw up the graph for PatternMatcher.java

Write up what paths you want to cover (ideally all nodes will be covered)

Write up the tests for those paths.

Run (with coverage)!