How to Make CFG?

Introduction of CFG Algorithm

200711470 정재호

200711457 윤홍국

200711438 송인근

CFG Introduction

- What is CFG?
- Statement of Purpose

What is CFG?

A control flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

Statement of Purpose

Reachability or different coverage strategies are some of the most common objectives. CFG-s are also build for static analysis reasons. Static analysis is the type of examination which does not consist on the execution of the code being analyzed but rather gaining information from other sources like documentation, reviews, formal methods or automated tools for analysis. From the evaluation only we can not get such information like reachability or coverage. This leads to the idea to use a specialized coverage tool to supply the necessary information about this measurements. After examining some know tools like EMMA, Coverlipse and others [linkCTools] to expand the functionalities, CodeCover turned out as the most fitting white box coverage tool for this plug-in and has been successfully integrated. It is a fully optional feature and can be activated or not without having an impact on the CFG generator.

Statement Processing

- Expression statement
- If statement
- Ternary operation
- For and While statement
- Do-while statement
- Switch case statement
- Try-catch statement

Expression statement

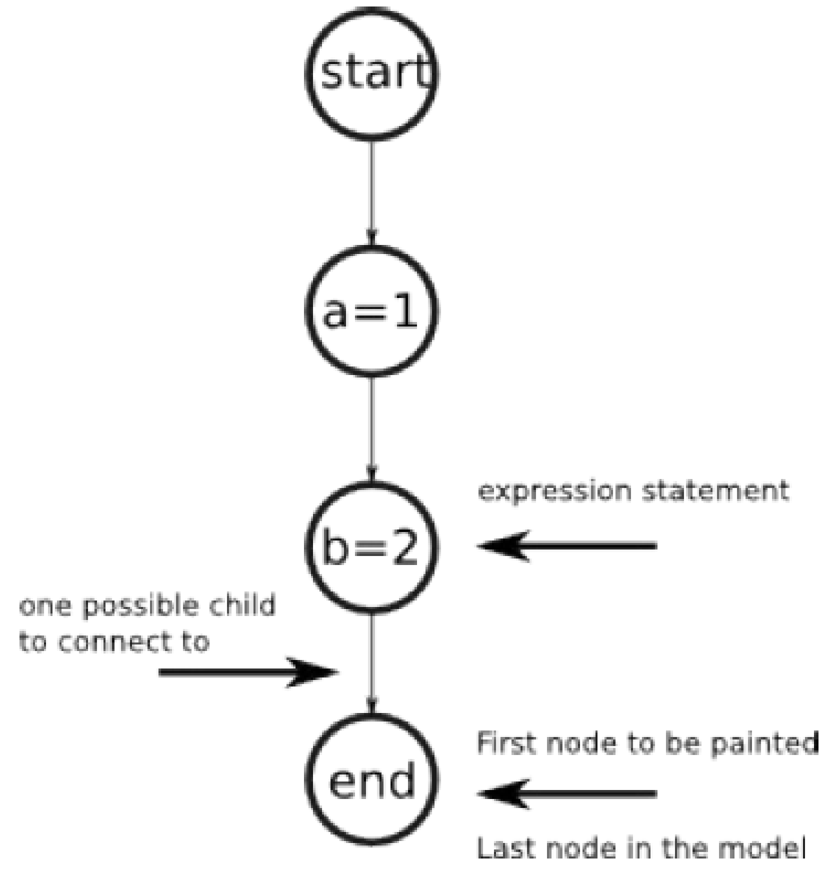
Expression statements are the nodes that do not have child nodes and for this reason the node is created and connected to its next node coming from the recursion.

Expression statement

Example:

```
void expression_statement(int b,int a) {
      a = 1;
      b = 2;
}
```

[Code 6.1 expression method]



[Figure 6.1 graph model for the expression statement]

If statement

- The difference between expression-statements and if-statements is that the last ones can contain child nodes.

- The second child is the content of the then-expression and the third one of the else expression but there is no difference between them.

- The first child of a node is the next node and not a real child of that node but, as if-statements contain children they need to be referenced as well.

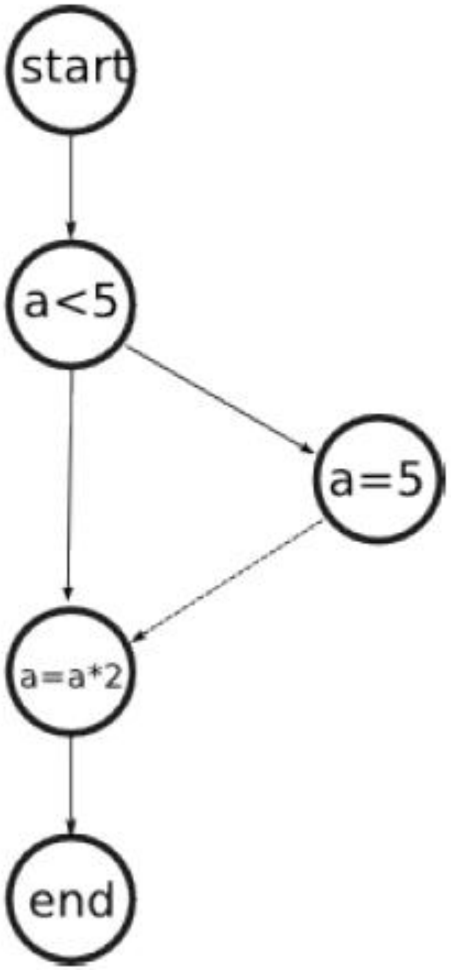
- Depending on the existence of an else statement there could not be a direct connection between if node and its next node.

If statement

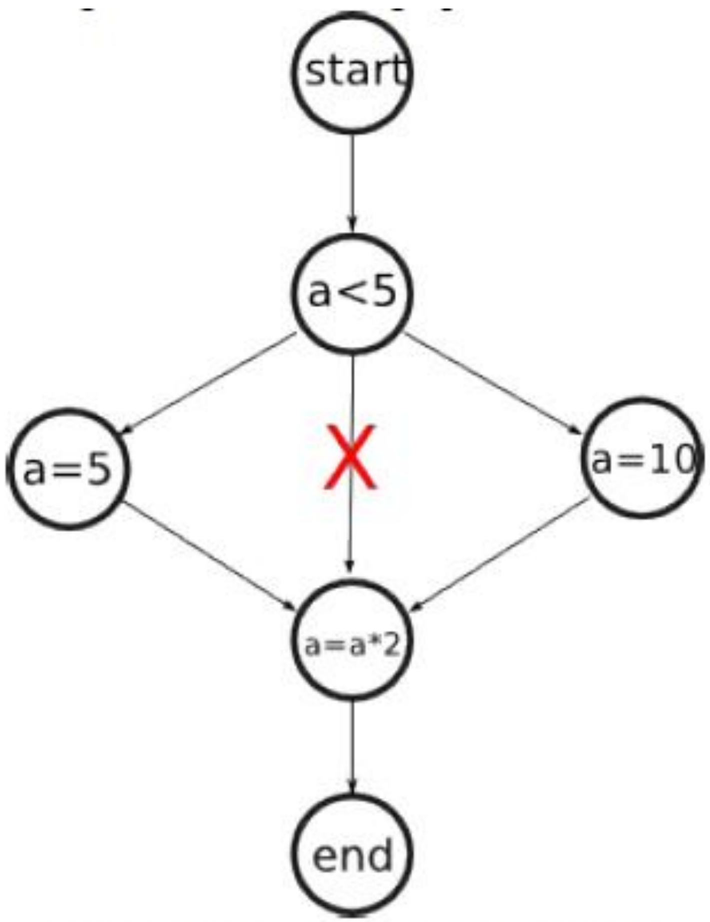
Example:

```
void ifTestMethod(int a) {
      if (a < 5) {
            a = 5;
      }
      a = a*2;
}
```

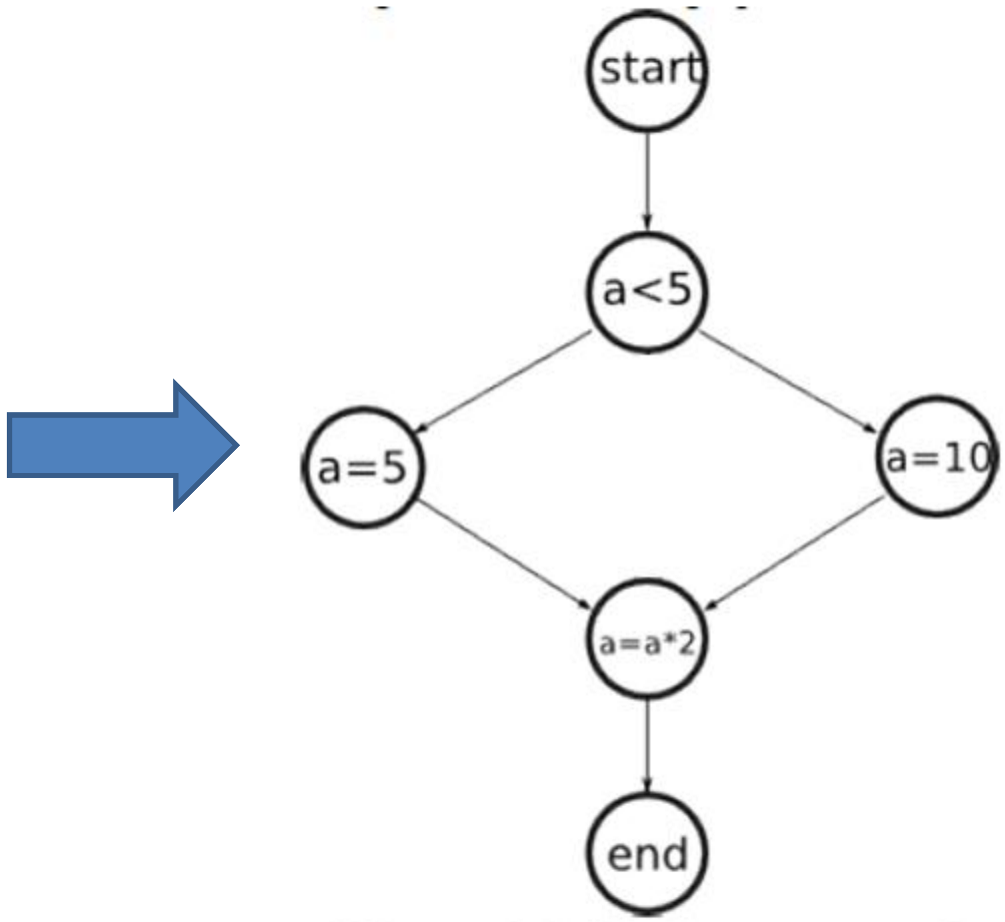
[Code 6.3 If statement]



If statement



[Figure 6.5 If else statement]



[Figure 6.5 If else statement]

Ternary operation

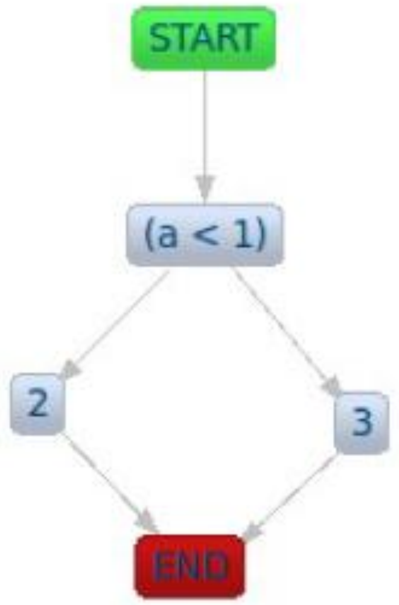
- The ternary operator is basically a short form of the if-else-statement.
- It has three arguments on the right side and the first one is the expression evaluation.
- The second and the third arguments are executed depending on this evaluation.
- For this reason as the choice is made in an if-else-form where both of them are always present, it was good to think of it as an if-else-statement and treat it as such.
- The ternary statement is added to the tree as an if-node and from that point it is used as such by the painting algorithm.

Ternary operation

Example:

```
void ternaryTestMethod(int b,int a) {
      b = (a<1)?2:3;
}
```

[Code 6.4 If statement]



[Figure 6.7 Ternary statement]

For and While statement

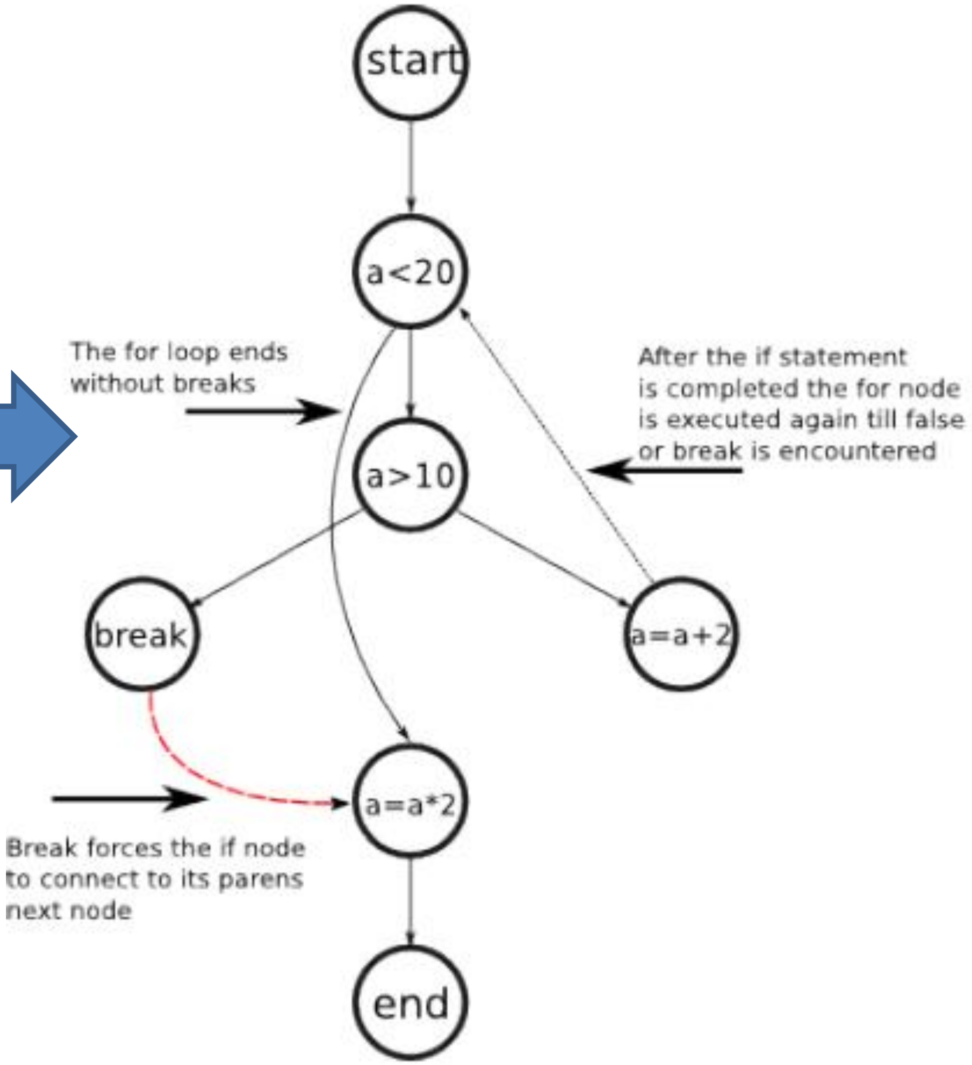
- From the logical interpretation there is no difference in the treatment of this two nodes.
- The expression evaluation is done at the beginning of the block for both cases.
- So that they are both processed as one case.
- They have one more child which is the inner part of the statement.
- It can be of any type and they are recursively build knowing that there is just a parent node and that is the for/while node.
- This is important for the case that a break or continue statement is encountered and at that point it must be decided whether it should connect to the parent (in case of a continue node) or to its next node (in case of a break node) to exit the block.

For and While statement

Example:

```
void forTestMethod(int a) {
      for (a = 1; a < 20;) {
            if (a > 10) {
                  break;
            } else {
                  a = a + 2;
            }
      }
      a = a*2;
}
```

[Code 6.5 For statement]


start
a<20
The for loop ends without breaks
After the if statement is completed the for node is executed again till false or break is encountered
a>10
break
a=a+2
a=a*2
Break forces the if node to connect to its parens next node
end

Do-while statement

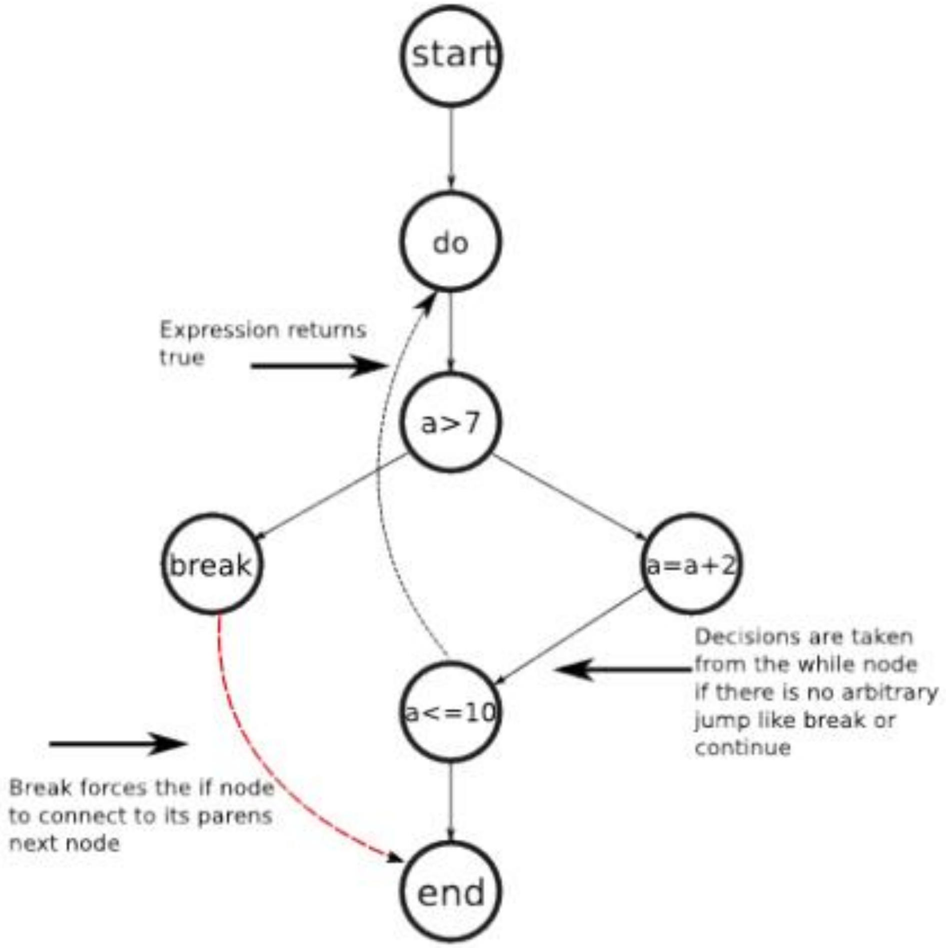
- Do statements are similar to the for and while statements with the exception that we have do nodes to represent this statement and the expression examination is done at the end of the block.
- This doesn't lead to a reaction change for its children.
- Basically the expression goes at the bottom of the block and it can be escaped from the bottom of the block without referencing the top.

Do–while statement

Example:

```
void doWhileTestMethod(int a) {
      do {
            if (a > 7) {
                  break;
            } else {
                  a = a + 2;
            }
      } while (a <= 10);
}
```

[Code 6.6 Do-while statement]


start
do
Expression returns true
a>7
break
a=a+2
a<=10
Decisions are taken from the while node if there is no arbitrary jump like break or continue
Break forces the if node to connect to its parens next node
end

Switch case statement

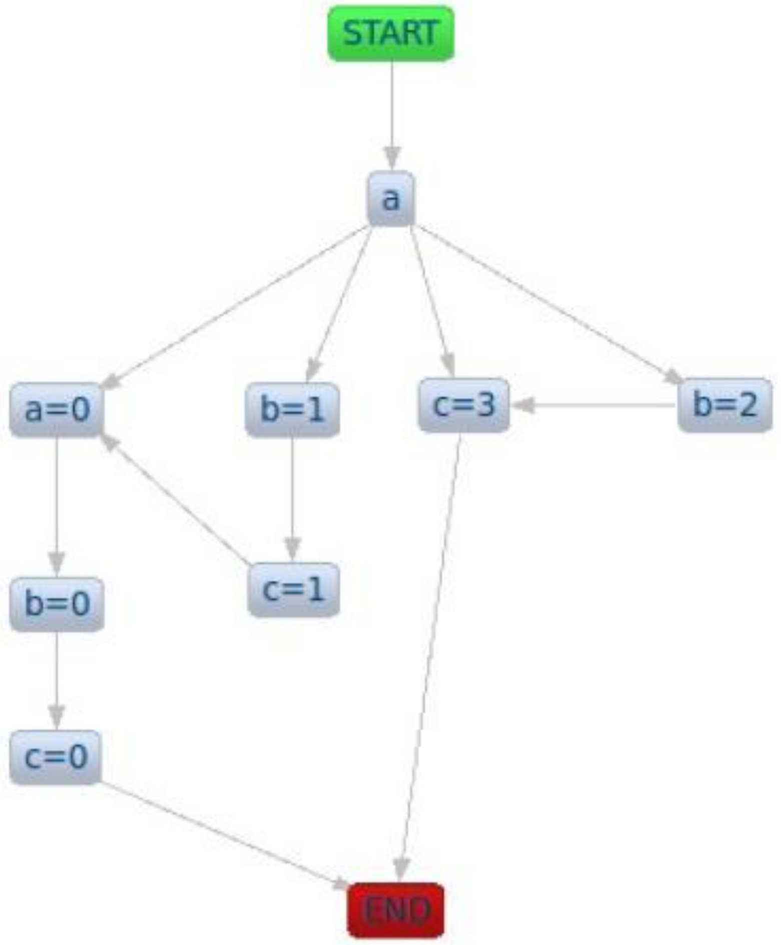
- If the if-statement allows just one of two possible choices, with the switch-case statement it is possible to chose out of an undefined number of choices.
- The switch node has as much children +1 where the first one is the next node and the rest are the cases or the default case.
- As every branch represents a decision, every case should have its own branch.
- If one case ends with a break statement, than it connects strait to the end.
- But depending on the switch variable it can have previews cases leading to it or not.

Switch case statement

Example:

```
void switchTestMethod(int a, int b, int c) {
        switch (a) {
        case 1:
                b = 2;
        case 2:
                c = 3;
                break;
        case 3:
                b = 1;
                c = 1;
        default:
                a = 0;
                b = 0;
                c = 0;
        }
}
```

[Code 6.7 Switch case statement]



Try-catch statement

- Try-catch-finally like switch nodes have an undefined number of child nodes. It can have many catch block and/or a finally block.
- Each of the catch blocks is interpreted as a branch which is executed if an error occurred in the try block.
- The finally block is the one which gets executed no matter if all the statements in the try-block or catch-blocks were fully and successfully executed.
- Therefore this is good reason to treat the finally-block as the next node in the raw.
- Try blocks are a bit more complicated concerning the unpredictable jumps that can occur during the execution and need to be examined in more details.

Try–catch statement

- A try-catch statement can almost be seen as a if-else statement, and sometimes incorrectly used as such instead of if-else statements even though Java literature discourages the usage of try-catch statements as normal control flow.

- If the try-block is executed successfully than none of the catch block is entered and the next node in the raw is referenced.

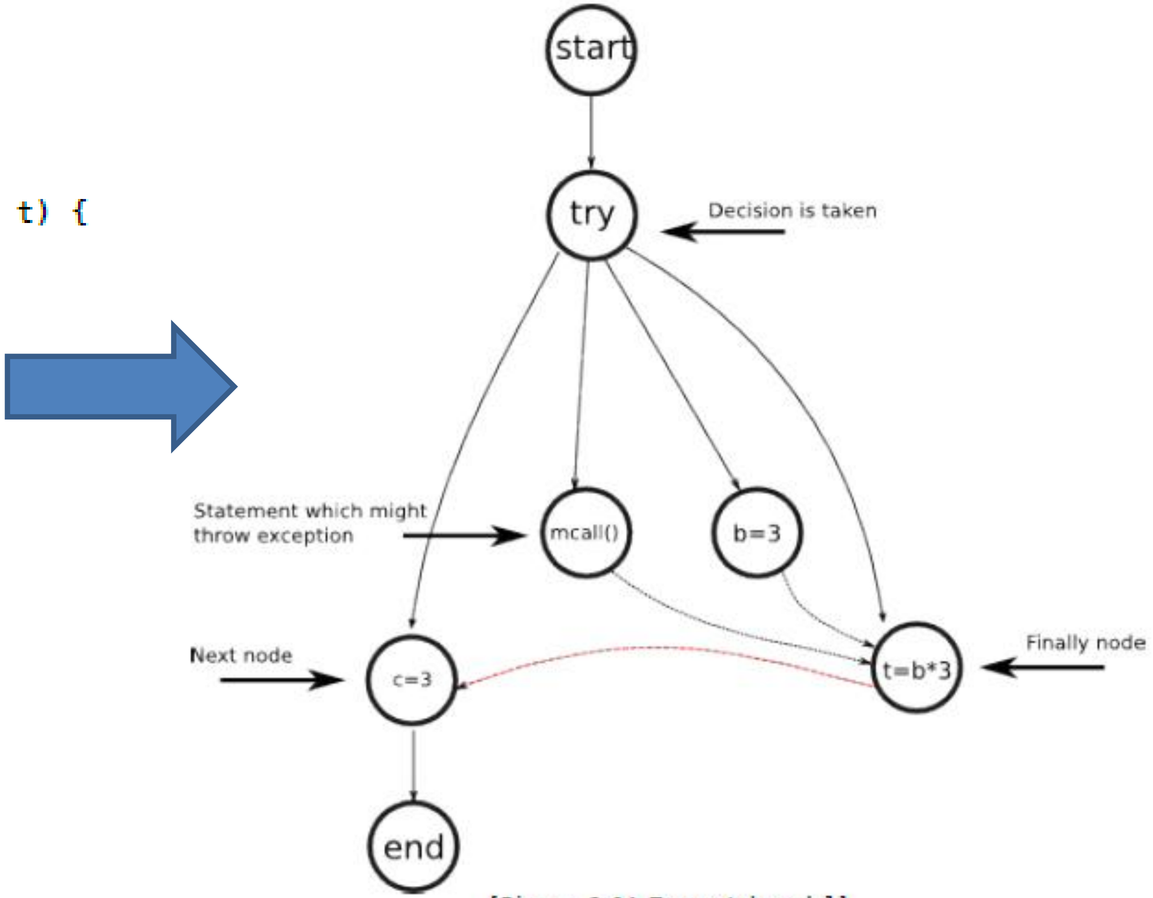
- If it has a problem than only one catch block is executed which turns it practically into a if-else similar form.

Try–catch statement

Example:

```
void tryCatchTestMethod(int b, int c, int t) {
      try {
            mightThrowAnException(b);
      } catch (Exception e) {
            b = 3;
      }finally {
            t = b*3;
      }
      c = 3;
}
```

[Code 6.9 Try-catch-finally example]



[Figure 6.14 Try-catch model]