1DV532 – Starting Out with Java

Lesson 4

Control Flow Statements

Dr. Nadeem Abbas

nadeem.abbas@lnu.se

Linnæus University

Control Flow Statements

- Control flow statement are used to control the flow of execution of a program
  - Program statements are generally executed in sequential order from top to bottom
  - Control flow statements may change the execution order, from one execution to another, based on certain *conditions* (*boolean expressions*) that evaluate to true or false.

Control Flow Statements

Java provides following types of the control flow statements:

1. **Selection Statements**
   1. `if`
   2. `if … else`
   3. `switch`
2. **Iteration Statements**
   1. `while`
   2. `do … while`
   3. `for`
3. **Branching Statements**
   1. `break`
   2. `continue`

Selection Statements

- The selection statements enable a program to *select* among a set of statements depending on the value of a *controlling expression*.

- Java support three type of selection statements
  1. **`if`**
  2. **`if-else`**
  3. **`switch`**

The `if` Statement

The `if` statement tells computer to execute a certain section of code only if a given condition, shown as *boolExpr* in the general syntax given below, evaluates to true.

General Syntax:

```
if ( boolExpr) {
    ... ; // Executed if boolExpr is true
}
```

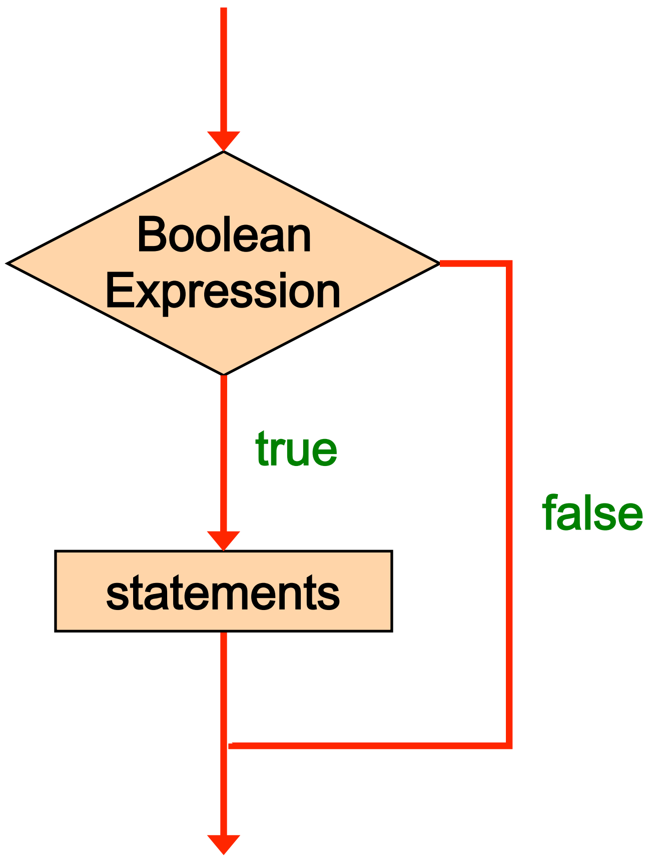
The expression **boolExpr** must evaluate to a **boolean** value *(**true** or **false**)*

Example:

```
int marks = 39;

if(marks >=40)
    System.out.print("Pass ");
```

The `if` Statement



The `if-else` Statement

The if-else statement tells computer to choose between two alternative blocks, if block and else block, based on value of a boolean expression.

General Syntax:

if (**boolExpr**) {

    ... ;

    // Executed if boolExpr is **true**

}

else {

    ... ;

    // Executed if boolExpr is **false**

}

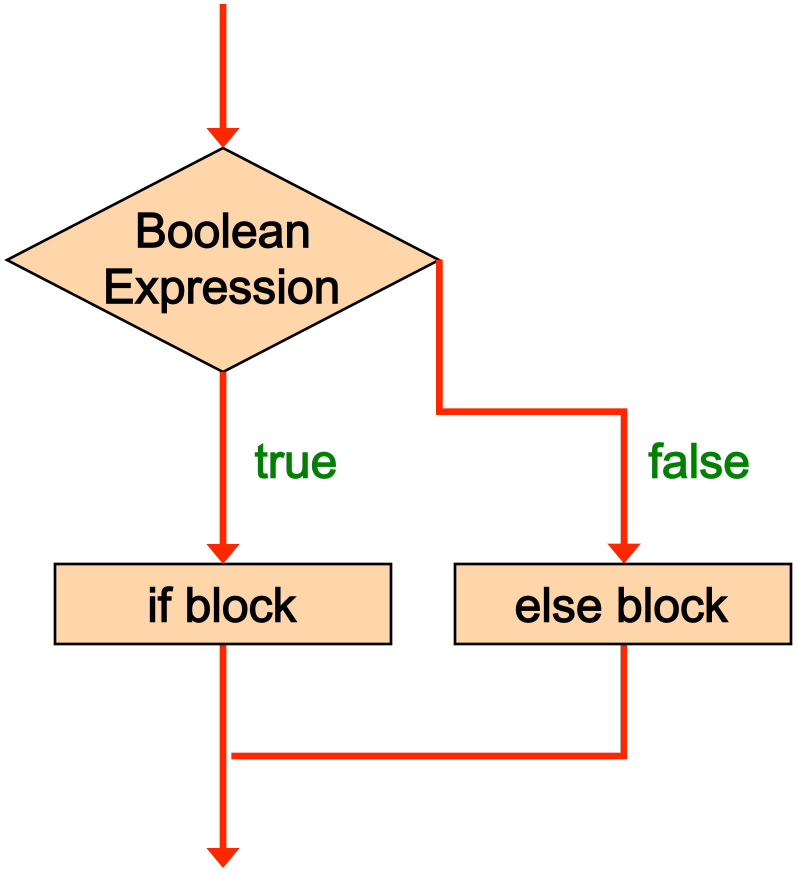
The expression **boolExpr** must evaluate to a **boolean** value ***(true*** *or* ***false)***

Example:

```
int marks = 39;

if(marks >=40)
    System.out.print("Pass ");
else
    System.out.print("Fail");
```

The `if-else` Statement



The if-else Statements - Example Program

```
class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 40;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

Nested if and if-else Statements

The statement executed as a result of an `if` or `else` statement could be another `if or if-else` statement

- Such statements are called *nested if or nested if-else statements*

Example:

```
if (testscore >= 90) {
     grade = 'A';
 } else if (testscore >= 80) {
     grade = 'B';
 } else if (testscore >= 70) {
     grade = 'C';
 } else if (testscore >= 60) {
     grade = 'D';
 } else {
     grade = 'F';
 }
```

Switch Statement

```
switch (expression) {
        case value1:
                // statement sequence
                break;
        case value2:
                // statement sequence
                break;
        .
        .
        .
        case valueN :
                // statement sequence
                break;
        default:
                // default statement sequence
}
```

- **Switch** statement enables a program to select from multiple choices (cases) based on a set of fixed values for a given *expression*.
- The ***Expressio*n** can be any expression of type integer, char, or String.
- The body of the switch acts like a series of ***if...else*** statements.
- The first case statement that fits is executed.
- **break** statement – moves the control out of the switch statement
- Each **case** clause should end with a break statement.
- The **default** acts like an else and catches anything that does not match with above cases.

Switch Statement – Example

```
int day = 1;
String dayString;
switch (day) {
   case 1:
        dayString = "Monday";
        break;
   case 2:
        dayString = "Tuesday";
         break;
   .
   .
   case 7:
        dayString = "Sunday";
         break;
   default:
        dayString = "Invalid day";
        break;
}
```

Iteration Statements

- The iteration statements enables a set of instructions or statements to be executed repeatedly for a specified number of time or until a condition is met.
  - These statements are often referred as **Loop** statements.

- Java support three type of iteration statements
  1. **`while`**
  2. **`do … while`**
  3. **`for`**

The while Loop

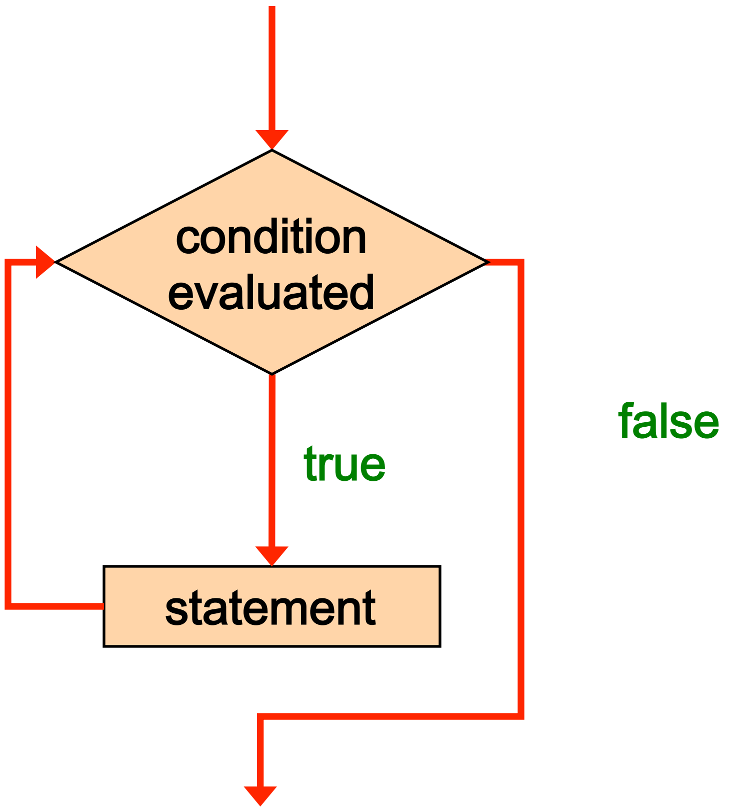
- A *while statement* has the following syntax:

```
while ( condition ){
    statement;

}
```

- If the `condition` is true, the `statement` is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

The while Loop



The while Loop

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
   System.out.println(count);
   count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Thus, the body of a `while` loop may execute zero or more times

Infinite Loops

- The body of a `while` loop eventually must make the condition false
  - If not, it is called an *infinite loop*, which will execute until the user interrupts the program.
  - This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
   System.out.println(count);
   count = count - 1;
}
```

- This loop will continue executing until interrupted.

The *do - while* Loop

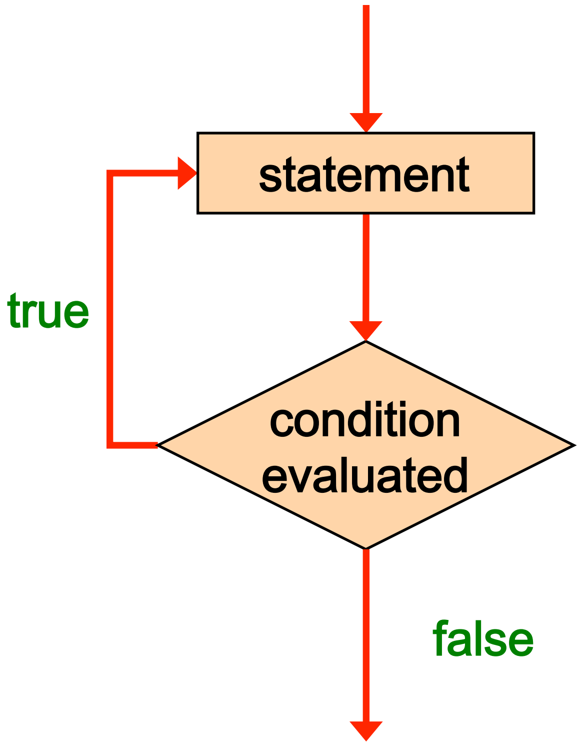
- A *do - while* loop has the following syntax:

```
do
{
    statement-list;
}
while (condition);
```

- The `statement-list` is executed once initially, and then the `condition` is evaluated

- The statement is executed repeatedly until the condition becomes false

The *do - while* Loop



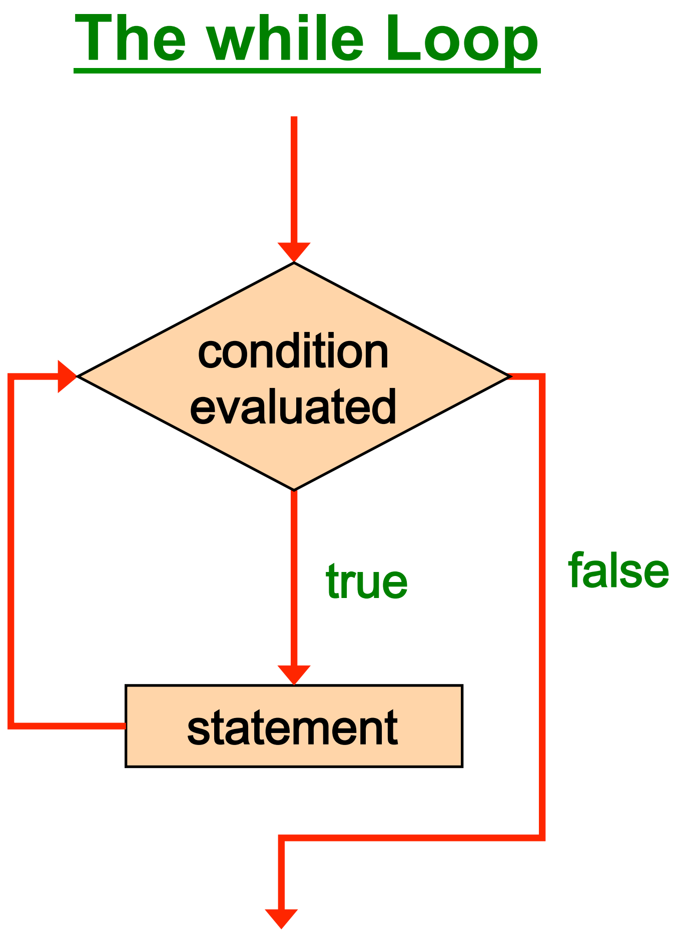
The *do - while* Loop

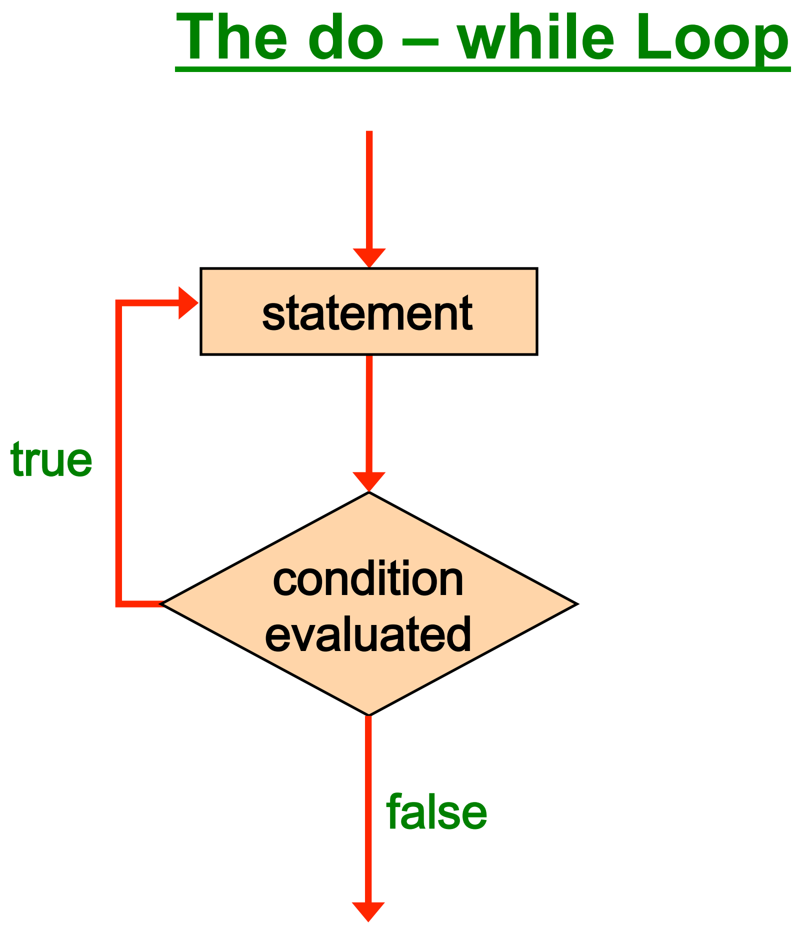
- An example of a `do - while` loop:

```
int count = 0;
do
{
    count++;
    System.out.println(count);
} while (count < 5);
```

- The body of a `do - while` loop always executes at least once, whether the condition is true or false.

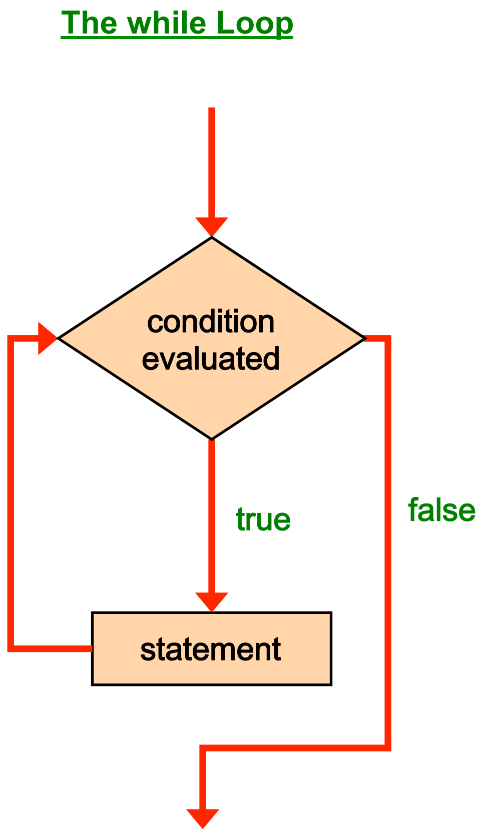
Comparing while and do – while loops

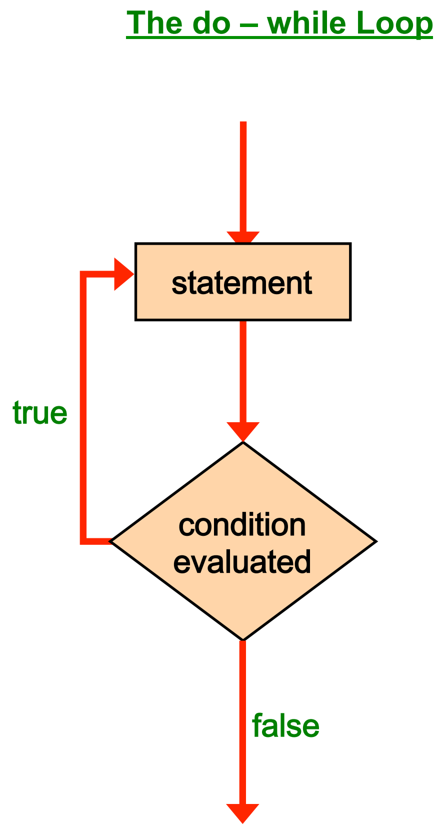




Comparing while and do – while loops

A **difference** between the two loops is that in the while loop, **condition is evaluated first** and the loop statements are executed only if the condition evaluate to true. On the other hand, in the do-while loop **condition is evaluated at the end**, after body of the loop, thus the loop statements are always executed at least once even if the condition evaluates to be false for the first iteration.





The for Loop

- A *for statement* has the following syntax:

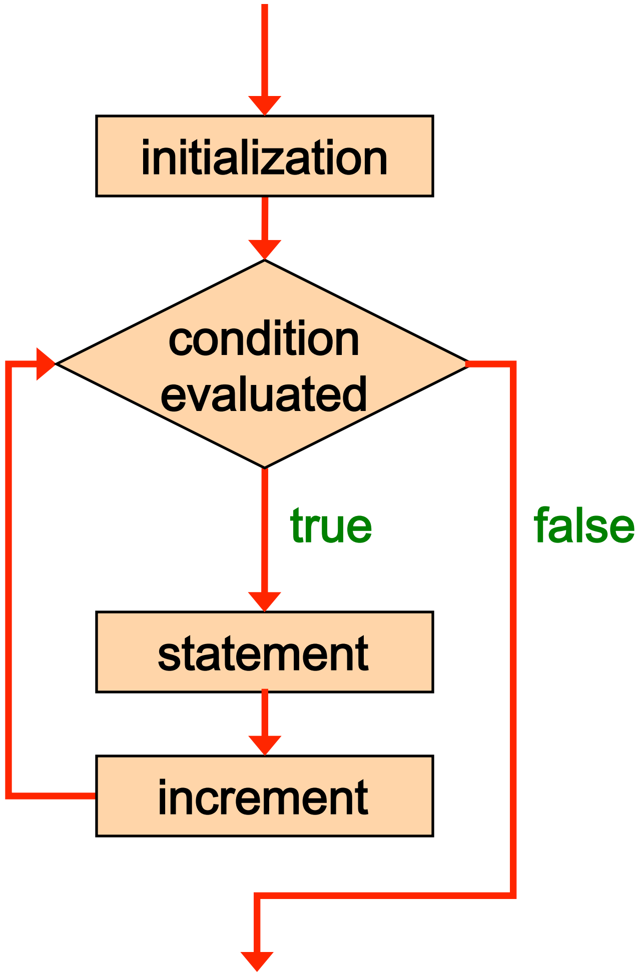
The ***initialization*** is executed once before the loop begins

```
for (initialization; condition; increment)
    statement; // body of the for loop
```

The ***statement*** is executed until the ***condition*** becomes false

The ***increment*** part is executed at the end of each iteration

The for Loop


initialization
condition evaluated
true
false
statement
increment

The for Loop

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while (condition)
{
    statement;
    increment;
}
```

The for Loop

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
   System.out.println(count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times

The for Loop

- The increment section can perform any calculation:

```
for (int num=100; num > 0; num -= 5)
   System.out.println(num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance

- An example program, IterationsDemo.java, provided with the lecture, demonstrates use of all the three iteration statement

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops – Example Program

```
class NestedForLoopDemo {
   public static void main(String[] args) {

      int rows = 5;

      for(int i = 1; i <= rows; ++i) //outer loop
      {
          for(int j = 1; j <= i; ++j) // inner loop
          {
             System.out.print(j + " ");
          }
          System.out.println("");
      }
   }
}
```

Branching Statements

- Branching statements, as the name indicates, are used to transfer control from one point to another point in the code.

- Java provides following branching statements
  - **`break`**
  - **`continue`**

The `break` and `continue` Statements

- The `break` statement consists of the keyword `break` followed by a semicolon ;
  - When executed, the `break` statement ends the nearest enclosing **switch** or **loop** statement
- The `continue` statement consists of the keyword `continue` followed by a semicolon ;
  - When executed, the `continue` statement ends the current loop body iteration of the nearest enclosing loop statement
  - Note that in a `for` loop, the `continue` statement transfers control to the ***increment*** part
- When loop statements are nested, remember that any `break` or **continue** statement applies to the innermost containing loop statement

The `break` and `continue` Statements – Example Program

```
class BreakContinueDemo {
   public static void main(String[] args) {
      for (int i = 1; i <= 20; ++i) {
         if (i == 10) { //break the loop when i is 10
            break;
         }
         if (i% 2 == 0) {  // continue the loop if i is
                           // an even number
             continue;
          }
         System.out.println(i);
      }
   }
}
```

Suggested Readings

- Absolute Java, Global Edition, 6/E by Walter J. Savitch, Chapter 3
- Introduction to Java Programming, Brief Version, Global Edition, 11/E Liang, Chapter 3 and 5
- Java Tuotrials:
  - **Control Flow Statements** https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html

Lnu.se