Java Object Oriented Programming

Core Java - Basics

Hiba ALQASIR

2021-2022

télécom saint-étienne
école d'ingénieurs / nouvelles technologies

Today

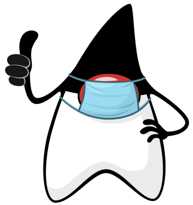
- **Goal**: learn the basics to do useful stuff in Java

Today

- When you see **Duke**
  → it is your turn: exercise!

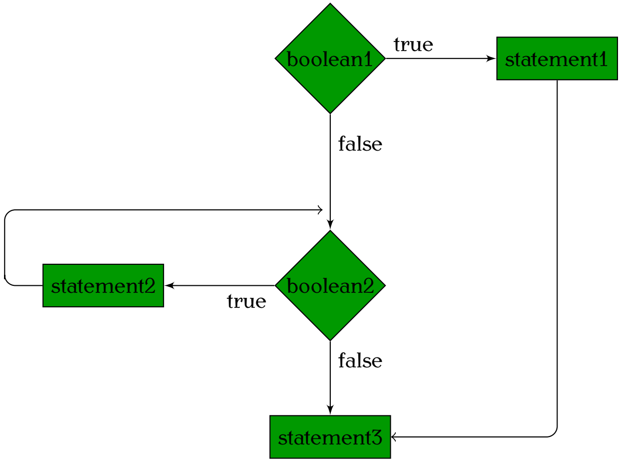
```java
public class TD2{
    public static void exercise1() {
        /* Write your code here */
    }
    public static void exercise2() {
        /* Write your code here */
    }
    // ....

    // The signature of the main method canNOT be modified
    public static void main(String[] args) {
        exercise1();
        exercise2();
        // ....
    }
}
```



Control flow statements

- Conditional statements
- Looping statements
- Jumping statements



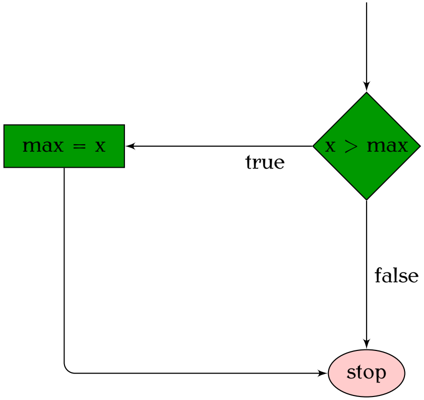
Conditional statements

- if
- if ... else
- switch

if statement

The **if** statement is used to execute a block of code only when a predefined condition is met.

- Evaluate a **boolean** expression.
  Ex: mathematical equation.
- If true, execute a statement.



if statement

Syntax

```java
if (condition) {
    // block of code to be executed if the condition is true
}
```

Example

```java
int max = 5;
int x = 9;
if (x > max) {
    max = x;
}
System.out.println("max value is: " + max);
```

Output

```
max value is:  9
```

Comparison operators

Relational

x > y	x is greater than y
x >= y	x is greater than or equal to y
x < y	x is less than y
x <= y	x is less than or equal to y

Equality

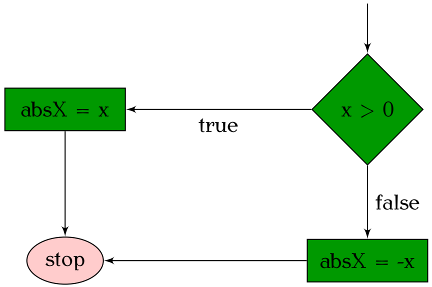
x == y	x and y are equal
x != y	x and y are not equal

```
/*
Be careful
    == is for equality
    =  is for assignment
*/
```

if ... else statement

The **if** statement is used to execute a block of code only when a predefined condition is met.
The **else** statement is used to execute a block of code, if the same condition is **not** met.

- Evaluate a **boolean** expression.
- If true, execute a statement.
- If false, execute a different statement.



if ... else statement

Syntax

```java
if (condition) {
    // block of code to be executed if the condition is true
} else {
    // block of code to be executed if the condition is false
}
```

Example

```java
float x = -9;
float absX; // absolute x
if (x > 0) {
    absX = x;
} else {
    absX = -x;
}
System.out.println("|" + x + "| = " + absX);
```

Output

```
|-9.0| = 9.0
```

shorthand **if ... else** statement

Syntax

```
variable = condition? value1 : value2
// <value1> is assigned to <variable> if <condition> is true
// <value2> is assigned to <variable> if <condition> is false
```

Example

```java
float x = -9;
float absX; // absolute x
absX = x > 0 ?  x :  -x;
System.out.println("|" + x + "| = " + absX);
```

Output

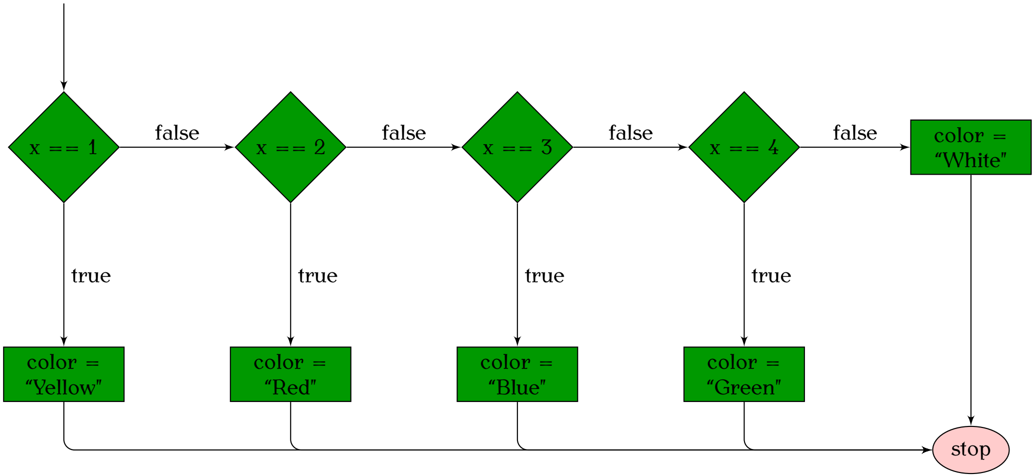
```
|-9.0| = 9.0
```

But what if we had many alternative blocks of code to be executed ?

```java
String color;
int x = 3;
if( x == 1){
    color = "Yellow";
}else if ( x == 2){
    color = "Red";
}else if ( x == 3){
    color = "Blue";
}else if ( x == 4){
    color = "Green";
}else {
    color = "White";
}
System.out.println(color);
```

Output

```
Blue
```


x == 1
false
x == 2
false
x == 3
false
x == 4
false
color = "White"
true
true
true
true
color = "Yellow"
color = "Red"
color = "Blue"
color = "Green"
stop

switch statement

We use **switch** to specify many alternative blocks of code to be executed.

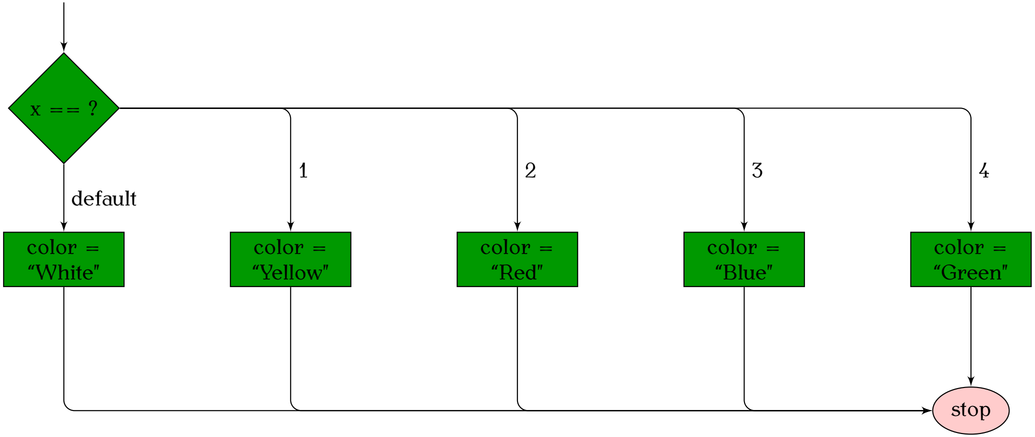
Syntax

```java
switch(expression) {
  case value1:
    // block of code to be executed when the result of the expression
    // matches value1
    break;
  ....
  case valueN:
    // block of code to be executed when the result of the expression
    // matches valueN
    break;
  default:
    // block of code to be executed when none of the values match
    // the value of the expression
}
```

switch statement

- The type of <expression> can be byte, short, char, int, String or enumerated.
- The [value1:valueN] must be of the same type as <expression>.
- If you drop **break** statement the program continues execution at the next <case>.

switch statement



switch statement

```java
String color;
int x = 3;
switch(x){
case 1:
    color = "Yellow"; break;
case 2:
    color = "Red"; break;
case 3:
    color = "Blue"; break;
case 4:
    color = "Green"; break;
default:
   color = "White";
}
System.out.println(color);
```

Output

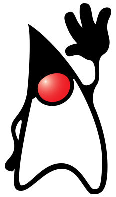
```
Blue
```

Exercise #1

What does **method1** do?

```java
public static void method1(int a, int b) {
    if (a > b) {
        int tmp = a;
        a = b;
        b = tmp;
    }
    System.out.println(a + " " + b);
}

public static void main(String[] args) {
    method1(33,77);
    method1(77,33);
}
```



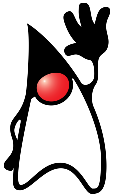
Output ?

Exercise #2

Add code to **method2** that puts a, b, and c in a descending order.

```java
public static void method2(int a, int b, int c) {
     /*  Write your code here */
    System.out.println(a + " " + b + " " + c);
}

public static void main(String[] args) {
    method2(55, 22, 77);
    method2(22, 55, 77);
    method2(77, 55, 22);
}
```



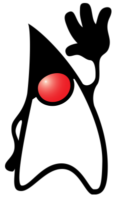
Output

```
77 55 22
77 55 22
77 55 22
```

Exercise #3

Rewrite the following snippet of code using switch statement.

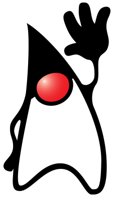
```java
if(x <= 0)
      System.out.println("Minimum value is 1");
else if(x == 1)
      System.out.println("Monday");
else if(x == 2)
      System.out.println("Tuesday");
else if(x == 3)
      System.out.println("Wednesday");
else if(x == 4)
      System.out.println("Thursday");
else if(x == 5)
      System.out.println("Friday");
else if(x == 6)
      System.out.println("Saturday");
else if(x == 7)
      System.out.println("Sunday");
else if(x >= 8)
      System.out.println("Maximum value is 7");
```



Exercise #4

Write the method **signSum** which takes two integers as parameters and which, *without calculating the sum*, returns 0 if the sum is zero, -1 if it is negative and 1 otherwise.

```
public static int signSum(int a, int b) {
    /*  Write your code here */
}
public static void main(String[] args) {
    System.out.println(signSum(2,3));
    System.out.println(signSum(2,-3));
    System.out.println(signSum(-2,3));
    System.out.println(signSum(2,-2));
}
```



Output

```
1
-1
1
0
```

Looping statements

Loops are used to execute a block of code repeatedly as long as a specified condition is true.

Advantages:

- save time
- reduce errors
- make code more readable

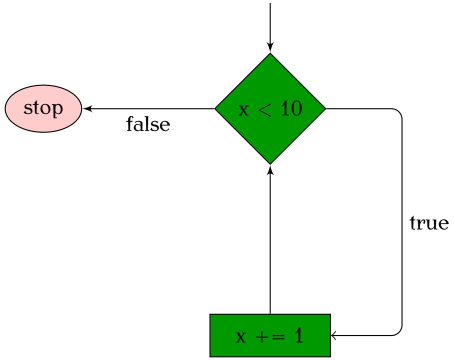
Looping statements

- while
- do ... while
- for

while loop

The **while** loop loops through a block of code as long as a specified condition is true.

- Evaluate a boolean expression.
- If true, execute a statement.
- Repeat.



while loop

Syntax

```java
while (condition) {
    // block of code to be executed
}
```

Example

```java
int x = 1;
while (x < 4) {
    System.out.println("#" + x);
    x = x + 1;
}
```

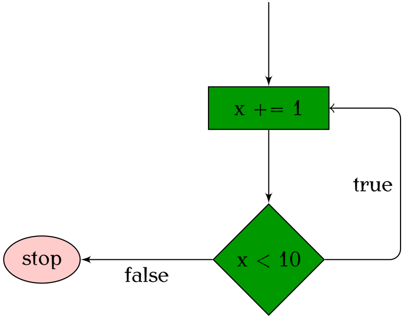
Output

```
#1
#2
#3
```

do ... while loop

The **do ... while** loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

- Execute a statement.
- Evaluate a boolean expression.
- If true, Repeat.



do ... while loop

Syntax

```java
do {
    // block of code to be executed
} while (condition);
```

Example

```java
int x = 1;
do {
    System.out.println("#" +x);
    x = x + 1;
} while (x < 4);
```

Output

```
#1
#2
#3
```

do ... while loop vs. while loop

```java
System.out.println("while loop");
int x = 5;
while (x < 4) {
    System.out.println("#" + x);
    x = x + 1;
}
```

Output

```
while loop
```

```java
System.out.println("do ... while loop");
int x = 5;
do {
    System.out.println("#" +x);
    x = x + 1;
} while (x < 4);
```

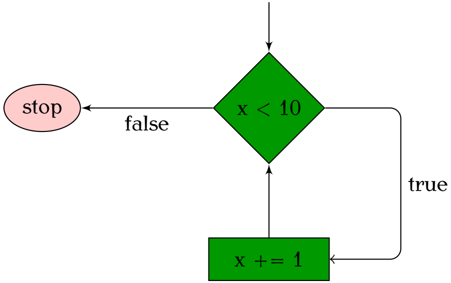
Output

```
do ...  while loop
#5
```

for loop

When you know exactly how many times you want to loop through a block of code, use the **for** loop.

- Execute an initialization statement.
- Evaluate a boolean expression.
- If true, execute a statement, then execute an update statement.
- Repeat.



for loop

Syntax

```java
for (statement 1; statement 2; statement 3) {
    // block of code to be executed
}
```

- statement 1: is executed only once as the loop begins.
- statement 2: defines the condition for executing the code block, when evaluates to false, the loop terminates.
- statement 3: is executed every iteration after the execution of the code block.

for loop

Example

```
for (int i = 1; i < 4; i++) {
    System.out.println("#" + i);
}
```

Output

```
#1
#2
#3
```

- statement 1: initialization

```
int i = 1;
```

- statement 2: termination

```
i < 4;
```

- statement 3: increment/decrement

```
i++;
```

for loop vs. **while** loop

for

```java
for (statement 1; statement 2; statement 3) {
    // block of code to be executed
}
```

while

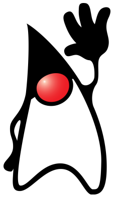
```java
statement 1;
while (statement 2) {
    // block of code to be executed
    statement 3;
}
```

Exercise #5

What is the wrong with the following code?

```java
int n = 5;
int v = 1;
int i = 0;
while (i <= n)
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
```

After fixing it, what does it do?



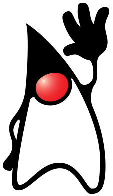
Exercise #6

What is the output of the following Java code?

```java
for(int i=0; 0 ; i++){
     i = i * 5;
     System.out.println(i);
}
```

What about the following one?

```java
for(int i=0; i > 0 ; i++){
     i = i * 5;
     System.out.println(i);
}
```

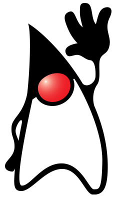


Exercise #7

Given the following code snippet:

```java
for (int i = 1; i < n; i = i * 4) {
    System.out.println("loop #1");
    for (int j = 1; j < i; j = j + 4) {
        System.out.println("loop #2");
    }
}
```

- If n = 7, what is the number of times “loop #1” is printed?
- If n = 17, what is the number of times “loop #2” is printed?



Exercise #8

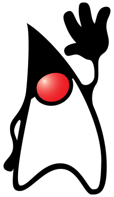
Write the method **primeNumbers** that takes as parameter an integer *max* and displays the prime numbers less than or equal to *max*.

```java
public static void primeNumbers(int max) {
    /*  Write your code here */
}

public static void main(String[] args) {
    primeNumbers(7);
    primeNumbers(13);
}
```

Output

```
2, 3, 5, 7
2, 3, 5, 7, 11, 13
```



Jumping statements

- break
- continue

break statement

break immediately exit a loop or a block of code, the control flow then transfers to the statement after the loop or the block.

Example

```
for (int i = 1; i < 5; i++) {
    if(i == 3)
        break;
    System.out.println("#" + i);
}

System.out.println("End of the loop");
```

Output

```
#1
#2
End of the loop
```

continue statement

continue skips the current iteration of a loop and proceeds directly to the next iteration

Example

```
for (int i = 1; i < 5; i++) {
    if(i == 3)
        continue;
    System.out.println("#" + i);
}

System.out.println("End of the loop");
```

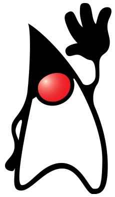
Output

```
#1
#2
#4
End of the loop
```

Exercise #9

What is the output of the following code?

```java
int count = 0;
for(int i = 0; i < 3; i++)
{
    count++;
    for(int j = 0; j < 3; j++)
    {
        count++;
        continue;
        // break;
    }
}
System.out.println(count);
```



What is the output if you replace continue; statement with break; statement?

Arrays

- An array is an indexed list of values, the index starts at zero and ends at length-1.
- An array holds elements of the same type, and that could be any type *int, double, String, boolean, etc..*

index	0	1	2	3	4	5	6	7	8	9
value	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'k'

index	0	1
value	"Hello"	"World"

index	0	1	2	3	4
value	105.1	222.2	13.7	89.8	99.9

Arrays

Declaration

Declaration

```
<type> [] oneDimensionalArray;          // OR <type> oneDimensionalArray [];
<type> [][] twoDimensionalArray;
<type> [][][] threeDimensionalArray;
```

Memory allocation

```
// you need to know the size of the array.
oneDimensionalArray = new <type>[<size1>];
twoDimensionalArray = new <type>[<size2>][<size3>];
```

Example

```
double [] array1;               // declaration
array1 = new double[10];        // memory allocation
int [][] array2 = new int[2][5]; // declaration & memory allocation
```

Arrays

Initialization

Initialization

```java
// Default initialization for numeric types
double [] numbers1 = new double[5];

// Curly braces can be used to initialize an array ONLY
// when you declare the array.
int [] numbers2 = {1, 7, 13, 0, 9, 2, 7};
```

index	0	1	2	3	4
value	0.0	0.0	0.0	0.0	0.0

numbers1

index	0	1	2	3	4	5	6
value	1	7	13	0	9	2	7

numbers2

Arrays

Access

Access the elements of an array

```
arrayName [ index ];
```

Example

```
System.out.println("The first element in the array is: "
                                      + numbers2[0]);
System.out.println("The last element in the array is: "
                                      + numbers2[numbers2.length-1]);
```

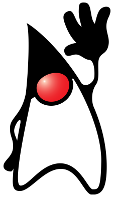
Output

```
The first element in the array is:  1
The last element in the array is:  7
```

Exercise #10

Given the following code snippet:

```
String[] coffee = {
            "Affogato", "Americano", "Cappuccino", "Corretto",
            "Cortado", "Espresso", "Frappucino", "Lungo",
            "Macchiato", "Marocchino", "Ristretto" };
String [] cake;
```



- What is wrong with the following line of code?

```
System.out.println("Today I'd like to drink " + coffee[11]);
```

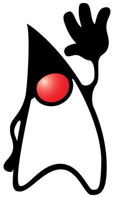
- Is there a problem with this line?

```
cake[0] = "donuts";
```

Exercise #11

What is the output of the following code snippet:

```java
String[] brands = {"Dior", "Gucci", "Givenchy", "",
                   "Fendi", "Chanel", "Prada", "Miu Miu"};
for(int i = 0; i < brands.length; i++)
{
    if(i > 3)
    {
        brands[i] = "Luxury";
    }
    else
    {
        brands[i] = "LV";
    }
}
System.out.println(brands[3]);
System.out.println(brands[4]);
```

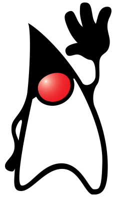


Exercise #12

Write a method that finds the minimum and maximum value of an array.

```java
public static void min_max(int arr[]) {
    int min;
    int max;
    /* Write your code here */
    System.out.println("Minimum value in the array = " + min);
    System.out.println("Maximum value in the array = " + max);
}

public static void main(String[] args) {
    int arr[] = {11,15,6,7,88,9,22,3,48};
    min_max(arr);
}
```



Output

```
Minimum value in the array = 3
Maximum value in the array = 88
```