

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
`byte -> short -> char -> int -> long -> float -> double`
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
`double -> float -> long -> int -> char -> short -> byte`

Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

```
Example

public class Main {
    public static void main(String[] args) {
        int myInt = 9;
        double myDouble = myInt; // Automatic casting: int to double

        System.out.println(myInt); // Outputs 9
        System.out.println(myDouble); // Outputs 9.0
    }
}
```

Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

```
Example

public class Main {
    public static void main(String[] args) {
        double myDouble = 9.78;
        int myInt = (int) myDouble; // Manual casting: double to int

        System.out.println(myDouble); // Outputs 9.78
        System.out.println(myInt); // Outputs 9
    }
}
```

Husk dette om Java Operators:

- Der findes ingen exponent-operator.

Java Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example	Try it
&&	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>	Try it »
	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>	Try it »
!	Logical not	Reverse the result, returns false if the result is true	<code>!(x < 5 && x < 10)</code>	Try it »

Husk dette om Java Strings:

- Der findes String methods - ligesom i Python. Disse kan læses herinde:
 - https://www.w3schools.com/java/java_strings.asp

Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

```
String txt = "We are the so-called "Vikings" from the north.";
```

- The solution to avoid this problem, is to use the **backslash escape character**.

The backslash (\) escape character turns special characters into string characters:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

The sequence \" inserts a double quote in a string:

Other common escape sequences that are valid in Java are:

Code	Result	Try it
\n	New Line	Try it »
\r	Carriage Return	Try it »
\t	Tab	Try it »
\b	Backspace	Try it »
\f	Form Feed	

Husk dette om Java Math:

- Nogle matematiske ting, som skal bruges med Math.XXXX, som i python bliver bruges ved .xxxx
- For eksempel:

Math.max(x,y)

The **Math.max(x,y)** method can be used to find the highest value of *x* and *y*:

Example

```
Math.max(5, 10);
```

Husk dette om Java Booleans:

- Ligesom Python

Husk dette om Java If...Else:

- **Forskel** - Alt i if-statement skal stå i parentes (!):

The if Statement

Use the **if** statement to specify a block of Java code to be executed if a condition is **true**.

```
Syntax

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

Note that **if** is in lowercase letters. Uppercase letters (**If** or **IF**) will generate an error.

In the example below, we test two values to find out if 20 is greater than 18. If the condition is **true**, print some text:

```
Example

if (20 > 18) {
    System.out.println("20 is greater than 18");
}
```

- else if, i stedet for elif:

The else if Statement

Use the **else if** statement to specify a new condition if the first condition is **false**.

```
Syntax

if (condition1) {
    // block of code to be executed if condition1 is true
} else if (condition2) {
    // block of code to be executed if the condition1 is false and condition2 is true
} else {
    // block of code to be executed if the condition1 is false and condition2 is false
}
```

- Short Hand If...Else

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Example

```
int time = 20;
String result = (time < 18) ? "Good day, " : "Good evening, ";
System.out.println(result);
```

Husk dette om Java Switch

- Udfør kode til ét af mange forudbestemte scenarier:

Husk dette om Java Class Methods:

- Ligesom du har lært tidligere om methods.

Static vs. Non-Static

You will often see Java programs that have either **static** or **public** attributes and methods.

In the example above, we created a **static** method, which means that it can be accessed without creating an object of the class, unlike **public**, which can only be accessed by objects:

Example

An example to demonstrate the differences between **static** and **public methods**:

```
public class Main {
    // Static method
    static void myStaticMethod() {
        System.out.println("Static methods can be called without creating objects");
    }

    // Public method
    public void myPublicMethod() {
        System.out.println("Public methods must be called by creating objects");
    }

    // Main method
    public static void main(String[] args) {
        myStaticMethod(); // Call the static method
        // myPublicMethod(); This would compile an error

        Main myObj = new Main(); // Create an object of Main
        myObj.myPublicMethod(); // Call the public method on the object
    }
}
```

Husk dette om Java Constructors:

- Hvis man vil angive variabler specifikt til objekt (parameters til class):

Example

```
public class Main {
    int modelYear;
    String modelName;

    public Main(int year, String name) {
        modelYear = year;
        modelName = name;
    }

    public static void main(String[] args) {
        Main myCar = new Main(1969, "Mustang");
        System.out.println(myCar.modelYear + " " + myCar.modelName);
    }
}

// Outputs 1969 Mustang
```

Husk dette om Java Modifiers:

Access Modifiers

For **classes**, you can use either **public** or **default**:

Modifier	Description	Try it
public	The class is accessible by any other class	Try it »
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »

For **attributes, methods and constructors**, you can use the one of the following:

Modifier	Description	Try it
public	The code is accessible for all classes	Try it »
private	The code is only accessible within the declared class	Try it »
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »
protected	The code is accessible in the same package and subclasses . You will learn more about subclasses and superclasses in the Inheritance chapter	Try it »

Non-Access Modifiers

For **classes**, you can use either **final** or **abstract**:

Modifier	Description	Try it
final	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)	Try it »
abstract	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)	Try it »

For **attributes and methods**, you can use the one of the following:

Modifier	Description
final	Attributes and methods cannot be overridden/modified
static	Attributes and methods belongs to the class, rather than an object
abstract	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example abstract void run() . The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters
transient	Attributes and methods are skipped when serializing the object containing them
synchronized	Methods can only be accessed by one thread at a time
volatile	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Example

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    case 4:
        System.out.println("Thursday");
        break;
    case 5:
        System.out.println("Friday");
        break;
    case 6:
        System.out.println("Saturday");
        break;
    case 7:
        System.out.println("Sunday");
        break;
}
// Outputs "Thursday" (day 4)
```

- Man kan også have "case default", til når ens switch-værdi ikke er en af de givne cases.

Husk dette om Java While Loop:

- Normalt while-loop:

Syntax

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

- 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.

Syntax

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

The example below uses a **do/while** loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Husk dette om Java For Loop:

Syntax

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

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

The example below will print the numbers 0 to 4:

Example

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

For-Each Loop

There is also a **"for-each"** loop, which is used exclusively to loop through elements in an **array**:

Syntax

```
for (type variableName : arrayName) {
    // code block to be executed
}
```

The following example outputs all elements in the **cars** array, using a **"for-each"** loop:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
    System.out.println(i);
}
```

Husk dette om Java Break/Continue:

- Break er som i Python - Stopper alle iterationer i loop
- Continue stopper nuværende iteration i loop, men fortsætter til næste.

Husk dette om Arrays:

- Array = List

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

- Fungerer i høj grad som lists i Python - Samme syntaks til at få adgang til indeks i array, samt array i array. F.eks:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Loop Through an Array

You can loop through the array elements with the **for** loop, and use the **length** property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
    System.out.println(cars[i]);
}
```

[Try it Yourself >](#)

Loop Through an Array with For-Each

There is also a **"for-each"** loop, which is used exclusively to loop through elements in arrays:

Syntax

```
for (type variable : arrayname) {
    ...
}
```

The following example outputs all elements in the **cars** array, using a **"for-each"** loop:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
    System.out.println(i);
}
```