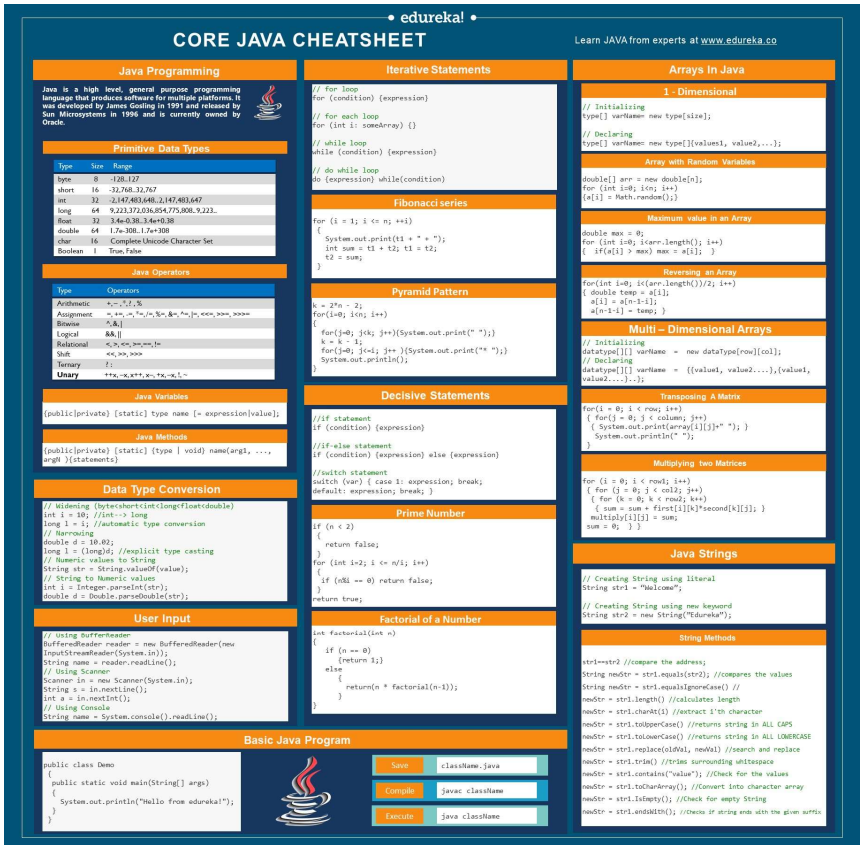
Java

23. august 2022 11:02



Java Tutorial

Husk dette om Java Syntax:

- Every line of code that runs in Java must be inside a `class`
- The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.
- The `main()` method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

- The curly braces `{}` marks the beginning and the end of a block of code.
  - De bruges hver gang man laver Tab

Husk dette om Java Output:

- You can use the `println()` method to output values or print text in Java
- There is also a `print()` method, which is similar to `println()`. The only difference is that it does not insert a new line at the end of the output

Husk dette om Java Comments:

- Single-line comments start with two forward slashes (`//`).
- Multi-line comments start with `/*` and ends with `*/`

Husk dette om Java Variables:

- In Java, there are different **types** of variables, for example:
  - `String` - stores text, such as "Hello". String values are surrounded by double quotes
  - `int` - stores integers (whole numbers), without decimals, such as 123 or -123
  - `float` - stores floating point numbers, with decimals, such as 19.99 or -19.99
  - `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
  - `boolean` - stores values with two states: true or false
- **Syntax:**

```
type variableName = value;
```

- Final Variables:
  - If you don't want others (or yourself) to overwrite existing values, use the `final` keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):
- Anderledes ved floats:
  - En float skal deklareres med tallet som normalt, efterfuldt af et "f". Ellers bliver værdien en double.
- Flere variabler i samme linje:

Declare Many Variables

To declare more than one variable of the **same type**, you can use a comma-separated list:

Example

Instead of writing:

```
int x = 5;
int y = 6;
int z = 50;
System.out.println(x + y + z);
```

You can simply write:

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

Try it Yourself »

One Value to Multiple Variables

You can also assign the **same value** to multiple variables in one line:

Example

```
int x, y, z;
x = y = z = 50;
System.out.println(x + y + z);
```

Husk dette om Java Data Types:

- `String` skal skrives med stort, alt andet med småt.
  - Data types are divided into two groups:
    - Primitive data types - includes `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` and `char`
    - Non-primitive data types - such as String, Arrays and Classes (you will learn more about these in a later chapter)

Numbers

Primitive number types are divided into two groups:

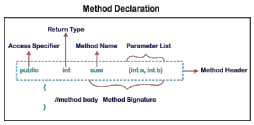
Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `byte`, `short`,

Java Methods

Husk dette om Java Methods:

- **Samme som funktioner i Python**
- A method must be declared within a class.

- **Syntax:**



Access Specifiers:

- public/private - Der kan også skrives andre modifiers:
  - https://www.w3schools.com/JAVA/java_modifiers.asp

Return types:

- Datatype (int, String, osv) eller void

Husk dette om Java Method Parameters:

- Ligesom i python, bortset fra at man skal skrive data-type på en parameters.

Husk dette om Java Method Overloading:

- **Til når man skal have en method som kan tage imod flere forskellige datatyper.**
- With **method overloading**, multiple methods can have the same name with different parameters:

Example

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Husk dette om Java Scope:

- Samme som python.

Husk dette om Java Recursion:

- Samme som python.

Java Classes

Husk dette om Java OOP:

- Afsnit med forklaring af og fordele ved objekt orienteret programmering.

Husk dette om Java Classes/Objects:

- Ingen `__init__()` - i stedet skrives disse ting blot i scopet for ens class. SE UNDER "Java Constructors"!
- Man laver en .java-fil til HVER CLASS.

Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the `main()` method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

- Main.java
- Second.java

Main.java

```
public class Main {
  int x = 5;
}
```

Second.java

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

- Data types are divided into two groups:
  - Primitive data types - includes `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` and `char`
  - Non-primitive data types - such as String, Arrays and Classes (you will learn more about these in a later chapter)

Numbers

Primitive number types are divided into two groups:

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `byte`, `short`, `int` and `long`. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: `float` and `double`.

Even though there are many numeric types in Java, the most used for numbers are `int` (for whole numbers) and `double` (for floating point numbers). However, we will describe them all as you continue to read.

Husk dette om Java Type Casting:

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

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 = 9.78d;
    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	x < 5 && x < 10	Try it »
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4	Try it »
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)	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

Second.java

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Husk dette om Java Class Attributes:

- Man kan ændre attributes som i python. Dog kan man låse en attribute ved at definere den med "final"-keyword:

Example

```
public class Main {
  final int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

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"

Husk dette om Java Encapsulation:

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as `private`
- provide public **get** and **set** methods to access and update the value of a `private` variable

Husk dette om Java Packages & API:

- Indbyggede packages
  - import xxx
- Selvlavede packages
  - package xxx

Husk dette om Java Inheritance:

- **En god del af dette lært ved oversættelse af python-program til Java**

Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

- 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:

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:

Husk dette om Java Inheritance:

- **En god del af dette lært ved oversættelse af python-program til Java**

Java Inheritance (Subclass and Superclass)

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from

To inherit from a class, use the `extends` keyword.

In the example below, the `Car` class (subclass) inherits the attributes and methods from the `Vehicle` class (superclass):

Did you notice the `protected` modifier in Vehicle?

We set the **brand** attribute in **Vehicle** to a `protected` access modifier. If it was set to `private`, the Car class would not be able to access it.

Why And When To Use "Inheritance"?

- It is useful for code reusability: reuse attributes and methods of an existing class when you create a new class.

Tip: Also take a look at the next chapter, Polymorphism, which uses inherited methods to perform different tasks.

Husk dette om Java Polymorphisms:

- Overskrivning af methods i child classes:

Example

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Husk dette om Java Exceptions:

Syntax:

Java try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Syntax

```
try {
  // Block of code to try
}
catch(Exception e) {
  // Block of code to handle errors
}
```

```
} finally {
  System.out.println("The 'try catch' is finished.");
}
```

The throw keyword

The `throw` statement allows you to create a custom error.

The `throw` statement is used together with an **exception type**. There are many exception types available in Java: `ArithmeticException`, `FileNotFoundException`, `ArrayIndexOutOfBoundsException`, `SecurityException`, etc:

Husk dette om Java Interface:

- **Bruges til privathed, til at vise udefrakommende hvad en klasse kan gøre (hvilke methods kan kaldes gennem et interface)**
- Generalisering af de methods som en klasse indeholder.

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
```

Lidt ligesom child class / parent class - men kun med navne på methods.

Notes on Interfaces:

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

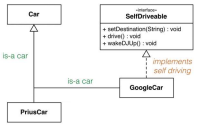
Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).

Denne video ses: Java Interfaces Explained - 040

Implementering - ikke is-a relation



GoogleCar.java

```
public class GoogleCar implements SelfDrivable {
    ...
}
```

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);
}
```

GoogleCar.java

```
public class GoogleCar implements SelfDrivable {
    ...
}
```

Husk dette om Java Enums

- En slags klasse med uskiftelige variabler.

Enums are often used in `switch` statements to check for corresponding values:

Example

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
        System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

Difference between Enums and Classes

An `enum` can, just like a `class`, have attributes and methods. The only difference is that enum constants are `public`, `static` and `final` (unchangeable - cannot be overridden).

An `enum` cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

Why And When To Use Enums?

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

Husk dette om Java ArrayList

- Tættere på en python list, end et normalt array. Det har nogle af de samme methods, men kan stadig kun have én datatype i sig.

Example

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Try it Yourself »

Husk dette om Java Hashmap:

- Fungerer ligesom dictionary i Python, med key/value-par.

Example

Create a `HashMap` object called **capitalCities** that will store `String` **keys** and `String` **values**:

```
import java.util.HashMap; // import the HashMap class

HashMap<String, String> capitalCities = new HashMap<String, String>();
```

- **Common methods:**
  put()
  get()
  remove()
  clear()
  size()

  keySet()
  values()

Husk dette om Java Iterator:

- Bruges til at iterere gennem samlinger af værdier (lister, hashmaps).

```
// Make a collection
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");

// Get the iterator
Iterator<String> it = cars.iterator();

// Print the first item
System.out.println(it.next());
```

- Bruges med f.eks. while loops, til at agere som for loops, bare mere simpelt.

Example

```
while(it.hasNext()) {
  System.out.println(it.next());
}
```

Try it Yourself »

Husk dette om Java Lampda:

- I Java kan man bruge Lampda-funktioner, ligesom man kan i python. Er essentielt en method, som kan skrives inde i en blok kode.

Syntax

The simplest lambda expression contains a single parameter and an expression:

```
parameter -> expression
```

To use more than one parameter, wrap them in parentheses:

```
(parameter1, parameter2) -> expression
```

Expressions are limited. They have to immediately return a value, and they cannot contain variables, assignments or statements such as `if` or `for`. In order to do more complex operations, a code block can be used with curly braces. If the lambda expression needs to return a value, then the code block should have a `return` statement.

```
(parameter1, parameter2) -> { code block }
```

Eksempel med Lampda og Interface:

Example

Create a method which takes a lambda expression as a parameter:

```
interface StringFunction {
  String run(String str);
}

public class Main {
  public static void main(String[] args) {
    StringFunction exclaim = (s) -> s + "!";
    StringFunction ask = (s) -> s + "?";
    printFormatted("Hello", exclaim);
    printFormatted("Hello", ask);
  }
  public static void printFormatted(String str, StringFunction format) {
    String result = format.run(str);
    System.out.println(result);
  }
}
```