Objektorienteret Programmering

Noter af Vivek Misra

Del 1 (Basics)

Vi præsentere de basale ting her.

Information

- Vi har besluttet os, at lære at programmering på Software Engineering Studiet.
- Den bedst måde, at sikre os at lære programmering er ved at starte med Java og Python, da det er basalt og nemt.
- Selvom vi er blevet bedt om, at starte fra Java – er det anbefalet at vi starte med Python og Thonny da det er god for dem som ikke har erfaring med Programmering overhovedet!
- Vi starter kort med Python, derefter går vi hurtigt til Java.
- Først gennemgår vi små teoriafsnit, med praktiske eksempler.

Hvordan starter vi praktisk med at kode? Java

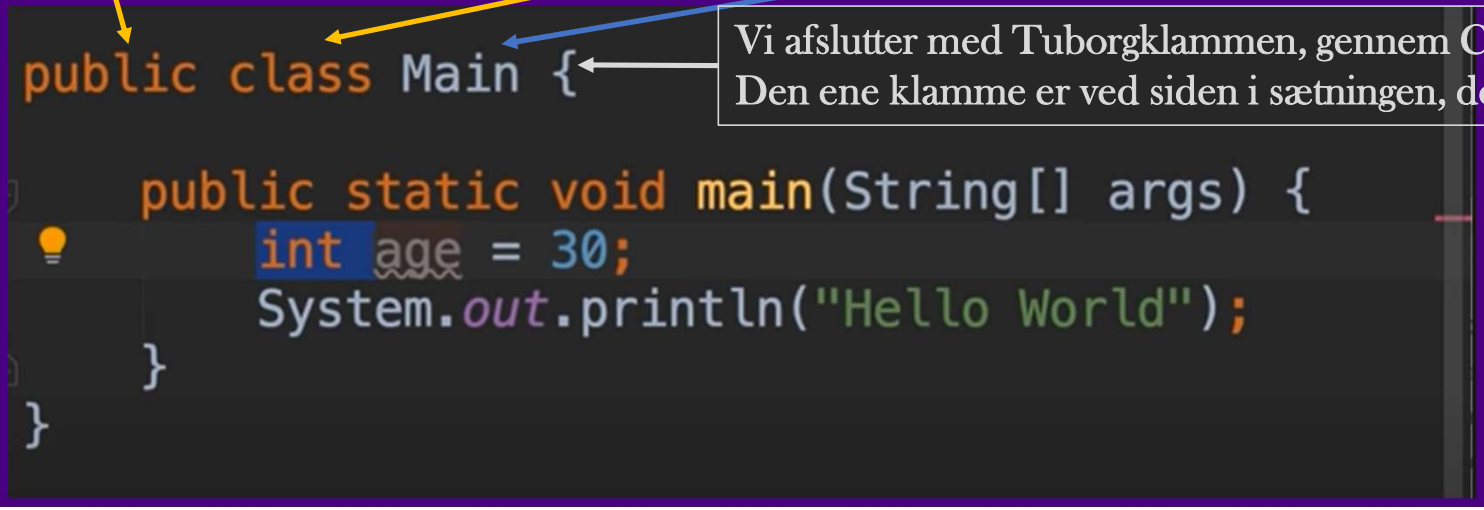
- Allerførst starter vi med, at skrive hvilken public class (filens navn).
  - Betydning: Det betyder, at vi offentligt (public) har en fil under klassen/navnet osv. Vi afslutter altid med tuborgklammen Ctrl+7, når vi starter.

Public, er filens offentligt tilstedeværelse.

Class, er filens offentlige navn.

Main, er eksempelvis filens navn i denne sammenhæng.

Vi afslutter med Tuborgklammen, gennem CTRL+7 og trykker enter. Den ene klamme er ved siden i sætningen, den anden er under (ikke vist)



Hvordan starter vi praktisk med at kode? Java

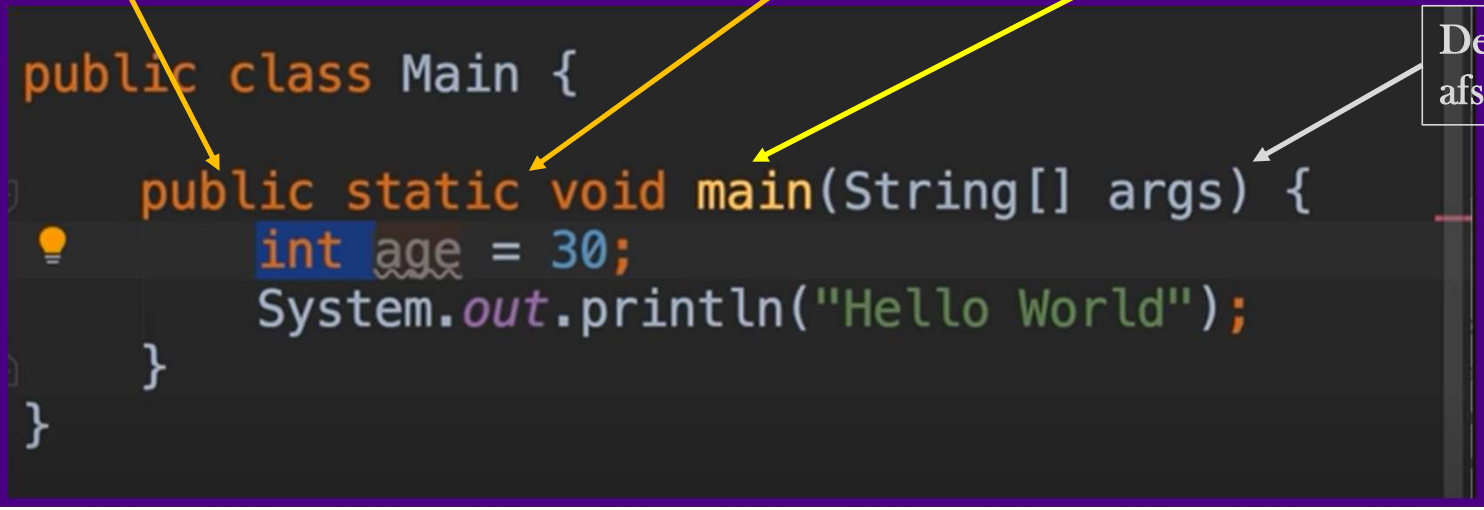
- Derefter skriver vi (public static void main(String[] args) {.
- {
  - Betydning: Vi prøver, at starte en offentligt hovedfunktion gennem en string og argument.

Public, er filens offentligt tilstedeværelse.

Static, void er også brugt til start

Main, er eksempelvis filens hovedfunktion.

Det er string og argumenter med afsluttende tuborgklamme.



Basale Begreber

- **Variabler:** Noget som vi definerer med.
  - Fra Gymnasiet og Grundskolen har vi kendt, at variabler er bogstaver som er defineret til noget.
  - I tilfældet her, er princippet lidt det samme. Forskellen er bare, at man har et navn hvor man oplagre data i.
- **Compiler:** Noget der oversætter fra det menneskelige sprog til Computersprog.
- **Interpreter:** En ”live” oversættelse af en igangværende tale.

Eksempel på brug af variabler (Python)

Birth er selve variablen, fordi den er defineret til at sige at man skal inputte et værdi for at kende alderen.

Det samme gælder for age, den er også kendt som en variabel.

Kigger man nærmere på selve ordet ”input”, betyder det at man vil gerne have en værdi indsat fra selve brugeren.

```
print("Hello, and welcome to Hello World")

print("We want you to type your age")

birth = input("Please, write your birth year below the message")

birth_year = input("Birth year: ")

age = 2022 - int(birth_year)

print(age)
```

Her i tilfældet kan det ses, at selve ”print” er der hvor man ønsker at udprinte/fremvise et besked. Dette betyder, at hvis vi skal skrive en besked til vores bruger, anvender vi print.

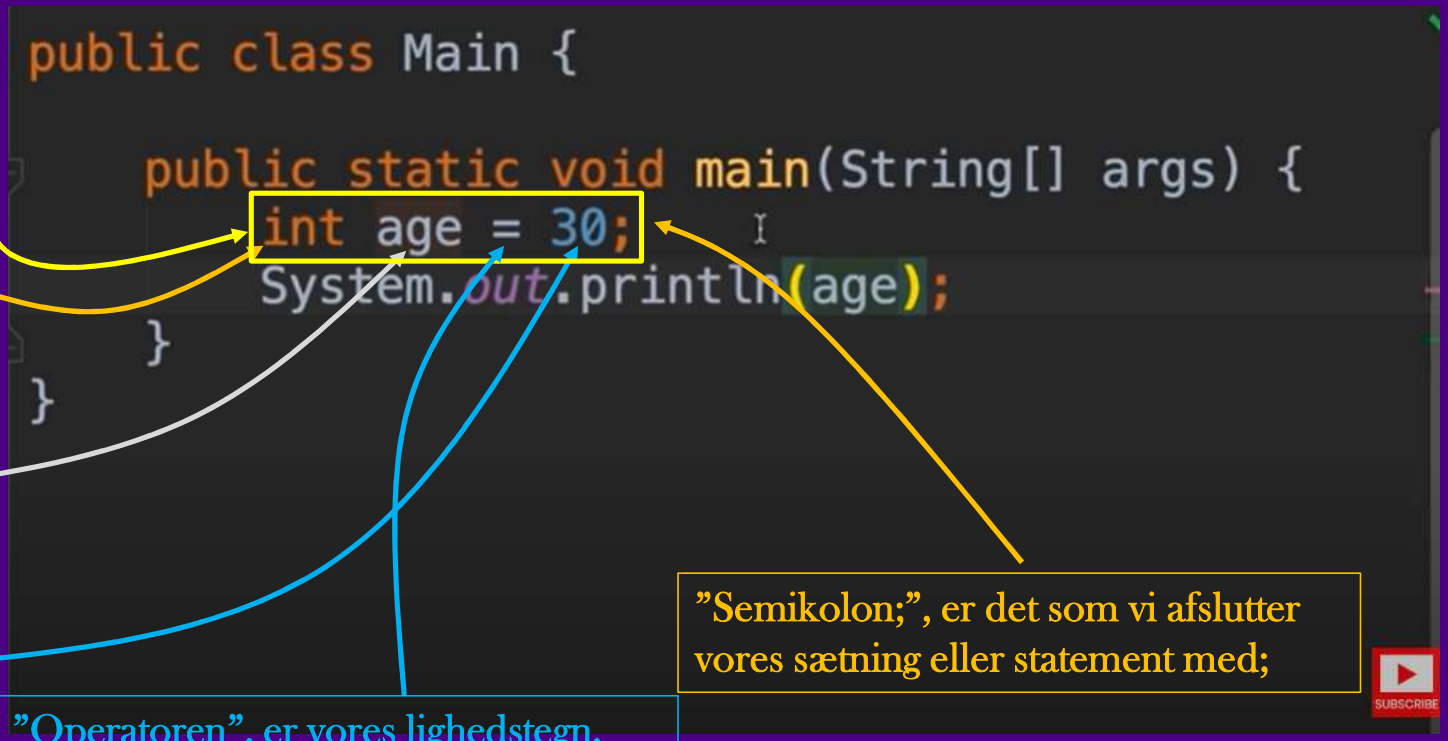
Eksempel på brug af variabler (Java)

Den markerede linje viser en defineret statement.

”Int” eller Integer er den tal som er defineret til at fortælle hvilken variabel type, der beskæftiges der med.

”age”, eller Alder er det som fortæller navngivet for vores variabel. Primært er det variablen.

30, er den alder som er blevet defineret til.



”Operatoren”, er vores lighedstegn.

”Semikolon;”, er det som vi afslutter vores sætning eller statement med;

Basale Begreber

Vi har to forskellige kategoriske typer i Java.

Primitive Typer

For at gemme simple værdier.

Referencetyper

For at gemme komplekse objekter.

Primitive Talværdier

- Sidste slide, kunne det ses at vi anvende integer til at beskæftige med taltypen.
- Nu skal vi lære om primitive tal, det er den måde tallene fortolkes i computeren på programmeringssprog.

Primitive Datatyper	Eksempler	Plads
Byte (Heltal)	-128 til 127	8 bit
Short (Heltal)	-32728 til 21474836647	16 bit
Integer (Heltal)	-2147483648 til 21474836647	32 bit
Long (Heltal)	-92233720368547758O8 til 9223372036854775807	64 bit
Floats	Decimaltal: 7 betydende cifre*	32 bit
Double	Decimalttal: 15 betydende cifre*	64 bit
Character	Tegn: A, i, ?, 2, 0, k, _, . Osv. Et unicode tegn*	16 bit
Boolean	Sandt / Falsk = 1 / 0	8 bit

Basale Begreber

Vi fandt ude af, at vi kan gemme følgende ting i Primitive Typer, men hvad kan vi gemme i Referencetyper?

Primitive Typer

Numre, Karakterer, Bolske tegn og Booleans.

Referencetyper

Tid, mail og besked.

String (Eksempel)

- Det er de grønne værdi med oppostrof.
  - Den optrædes som en variabel og en oppostrof på samme tid.

```
public class Main {

    public static void main(String[] args) {
        int age = 30;
        System.out.println("Hello World");
    }
}
```

”Hello World”, er en ”string” fordi den er med i variablen men optræder som en besked også.

Arrays

- Adskiller sig fra en variabel.
  - Variable: Er der, hvor vi ønsker at oplagre data i Computerens Hukommelse.
  - Arrays: Der, hvor vi oplagre lister af tal og værdier. De har fast længder.
  - Der er altid snak om, at konvertere en variabel om til en array.

```
public static void main(String[] args) {
    int number = 1;
}
```

Vi har variablen her, men vi ønsker at ændre det til array.

Vi indsætter firkantede klammer, efter (integer).

Array, er navngivet på forhånd.

```
public static void main(String[] args) {
    int[] number = new int[];
}
```

New "integer" indskriver og klammen skal angive hvad længde af tallet skal være.
Vi putter ind, hvilke tal vi ønsker.

Arrays

- Adskiller sig fra en variabel.
  - Variable: Er der, hvor vi ønsker at oplagre data i Computerens Hukommelse.
  - Arrays: Der, hvor vi oplagre lister af tal og værdier. De har fast længder.
  - Der er altid snak om, at konvertere en variabel om til en array.

```
public static void main(String[] args) {
    int number = 1;
}
```

Vi har variablen her, men vi ønsker at ændre det til array.

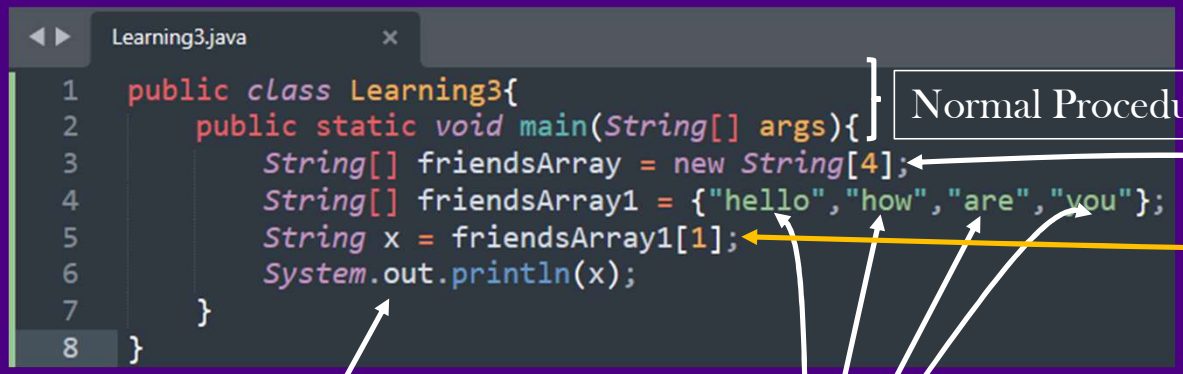
Vi indsætter firkantede klammer, efter (integer).

Array, er navngivet på forhånd.

```
public static void main(String[] args) {
    int[] number = new int[];
}
```

New "integer" indskriver og klammen skal angive hvad længde af tallet skal være.
Vi putter ind, hvilke tal vi ønsker.

Eksempel på Array



Normal Procedure

Definere et variabel og indskriver den værdi som er i array-længden.

Vi kan se, at vi har System.out.println(x)

Dette er (0,1,2,3,4)

Opskrives for, at definere længden af Array.

Loops

- En metode til at gentage funktioner på flere måder.
  - Der findes to former for loops: For or While

Learning2.java

```
1  public class Learning2{
2      public static void main(String[] args){
3          int[] nums = {2,3,5,4,6,6};
4          for (int i=0; i <= nums.length ; i++){
5              System.out.println(i);
6          }
7      }
8  }
```

Dette er vores Interval eller Arrayliste.

Startpunkt

Længden af Array

Optælling

```
C:\Users\vivek\OneDrive\Personal Folder\Dokumenter\Oevelser 1_1>javac Learning2.java

C:\Users\vivek\OneDrive\Personal Folder\Dokumenter\Oevelser 1_1>java Learning2
0
1
2
3
4
5
6

C:\Users\vivek\OneDrive\Personal Folder\Dokumenter\Oevelser 1_1>
```

Vi kan se, at når vi beskæftiger os med for-loops, så kan vi se at intervallet begynder med plads 0 osv.

Når vi starter med et tal, så kan det ses at tallet står på rækkelisten, men den fortsætter indtil optælling når sit mål på den sidste længde af array som er 6-tallet.

Metoder

- Informationer kan blive videre passeret som parametre.
- Parametre er variabler inde i et metode.
- Man kan se på billede, at metoden tager en streng som er kaldt fname.
  - fname er en parameter.
- Når en metode er kaldt, passeres den første navn, som er brugt inde i metoden til at printe den fulde navn.
- Se på næste side.

Metoder

String er en datatype.

```java
public class Learning1{
    static void myMethod(String name){
        System.out.println(name + "Misra");
    }
    public static void main(String[] args){
        myMethod("Vivek");
        myMethod("Vaishnavi");
        myMethod("Abhishek");
    }
}
```

```
C:\Users\vivek\OneDrive
es>cd "C:\Users\vivek\Or
Practices"

C:\Users\vivek\OneDrive
es>javac Learning1.java

C:\Users\vivek\OneDrive
es>java Learning1
VivekMisra
VaishnaviMisra
AbhishekMisra

C:\Users\vivek\OneDrive
es>
```

Billedet viser en metode.

Void betyder, at vi ønsker ikke noget i retur.

myMethod er navnet på metoden.

name er indirekte defineret for de variabler og navne som skal bruges.

Ny metode bruges

Navne er argumenterne, som er defineret gennem metoden. Nu printes de!

Java Methods og Parametre

- Kig på eksemplet igen nedenfor, med alders ændringer.

```java
public class Learning1{
    static void myMethod(String name){
        System.out.println(String name + "is" + int age + "years old");
    }
    public static void main(String[] args){
        myMethod("Vivek",19);
        myMethod("Vaishnavi",22);
        myMethod("Abhishek",21);
    }
}
```

```
//Vivek is 18 years old
//Vaihsnavi is years 22 old
//Abhishek is years 21 old
```

Static betyder, at den tilhører main klassen og ikke til selve objektet myMethod.

Vi starter med, at kalde metoden på følgende måde.

1. Vi skriver metodens navn med paranteserne() og semikolon;

2. MyMethod er brugt til at printe en tekst i aktion, når det er kaldt.

Java Methods og Parametre

- Kig på eksemplet nedenfor.

```
1  public class Learning1{
2      static void myMethod(String name){
3          System.out.println(String name + "is" + int age + "years old");
4      }
5      public static void main(String[] args){
6          myMethod("Vivek",19);
7          myMethod("Vaishnavi",22);
8          myMethod("Abhishek",21);
9      }
10 }
```

Static betyder, at den tilhører main klassen og ikke til selve objektet myMethod.

System.out.println(name + ”is” + age + ”years old”); - Her printer vi navnet (vi vælger selv navnet), og skriver en String ”is”, hvor vi vil gerne have udprintet et alder med tekst ”år gammel”.

Main-metoden er ”crafting table” og myMethod er det som vi ønsker at få udprintet.

Under myMethod skrives navnet og alderen.

Så refereres det til myMethod igen og det gør at vi får udskrevet vores værdier.

(Husk, at vi har adgang til metoderne, fordi der står public og ikke private).

Java Methods og Parametre

- (Når) en parameter er passeret videre til en metode er det en argument.

```
public class Learning1{
    static void myMethod(String name){
        System.out.println(name + "Misra");
    }
    public static void main(String[] args){
        myMethod("Vivek");
        myMethod("Vaishnavi");
        myMethod("Abhishek");
    }
}
```

Fname er en parameter, fordi det under myMethod er defineret som navn (attribut).

Navnene er argumenterne, fordi den ovenstående parameter er passeret videre til metoden, som så her er blevet udskrevet.

Der kan blive tilføjet flere parametre, som eksempelvis fname + ”is” + age + ”old”.

Return Values

- Som sagt betyder void at metoden skal ikke returnere et værdi.
  - Hvis man ønsker, at metoden skal aflevere et værdi skal man anvende primitive datatype som er (int, char osv.) og derved bruge return keyword inde i metoden.

```
public class Main {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

5Steps-Metoder

1. Access (Modifier)

2. På klassen eller.....

3. Return

4. Name

5. Input

Metode	Public	Static	Void	Main	( )
X	1	2	3	4	5

Metode	Public	~~Static~~	Void	Name	( )
X	1	~~2~~	3	4	5

Klasse	Public	Class	Name	~~( )~~
Y	1	2 / 3	4	~~5~~

Konstru	Public	~~Static~~	~~Void~~	C-Name	( )
Y	1	~~2~~	~~3~~	4	5

I billedet ovenpå, kan det ses hvordan forskellen mellem metoder, klasser og konstruktører er.

Del 2 (Objektorienteret)

Vi præsentere de objektorienterede ting her.

Klasser, Objekter og Konstruktører

- Klasser = Filen
  - Man skal se en klasse som en skabelon til objekter.
- Objekter = Genstande
  - Et objekt er en forekomst af en klasse.
  - Når de enkelte objekter oprettes, arver de alle variabler og metoder fra klassen.
- Konstruktører = Genstandsdel

Flere Objekter

- Vi har oprettet flere objekter af Main, ved at bruge den samme metode fra sidst.

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

  public static void main(String[] args) {
    Main myObj1 = new Main();
    Main myObj2 = new Main();
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

```
5
5
```

Når vi opretter en objekt (VIGTIG SYNTAKS)

- Billedet viser, at vi har oprettet en main-klasse-
- Derefter har vi lavet en main-metode, hvor parametrene siger at vi har nogle datatyper i form af strings og argumenter.
- Objektet er oprettet ved at vi skriver klassens navn først, derefter skriver vi navnet på metoden. Så skriver vi ”lig med”, skriver vi new Main();

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

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

Attributter – Hvad er det?

- ”Variable” er et andet udtryk for (attributter).
  - Næste gang, der bliver spurgt det så ved du hvad der snakkes om.
  - Vi viser et eksempel, hvor vi har en main-klasse og hvor vi har to attributter x og y.

```
public class Main {
  int x = 5;
  int y = 3;
}
```

Adgang til Variabler

- I det tidligere eksempel kunne vi se, at vi havde en variabel som var x.
- Her kan vi se, at vi kan referere til en x-værdi og derved printe objektet.
- Vi kan også definere den udprintet værdi efter erklæring af Objektet

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

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

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

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

Adgang til Variabler

- Samme metode kan anvende for flere objekter og derved ”strings”.

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

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

```java
public class Main {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

Metoder

- Vi starter som sagt med, at skrive ”static void”.
- Metoderne printer den aktion, som er kaldt.
- Vi skriver metodens navn og sætter parentesen med semikolon;

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "Hello World!"
```

Adgang til metoder gennem Objekter

Example explained

1) We created a custom `Main` class with the `class` keyword.

2) We created the `fullThrottle()` and `speed()` methods in the `Main` class.

3) The `fullThrottle()` method and the `speed()` method will print out some text, when they are called.

4) The `speed()` method accepts an `int` parameter called `maxSpeed` - we will use this in **8)**.

5) In order to use the `Main` class and its methods, we need to create an **object** of the `Main` Class.

6) Then, go to the `main()` method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).

7) By using the `new` keyword we created an object with the name `myCar`.

8) Then, we call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program using the name of the object (`myCar`), followed by a dot (`.`), followed by the name of the method (`fullThrottle();` and `speed(200);`). Notice that we add an `int` parameter of **200** inside the `speed()` method.

Create a Car object named `myCar`. Call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program:

```
// Create a Main class
public class Main {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Main myCar = new Main();   // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Indkapsling

- Klasser = Filen
  - Man skal se en klasse som en skabelon til objekter.
- Objekter = Genstande
  - Et objekt er en forekomst af en klasse.
  - Når de enkelte objekter oprettes, arver de alle variabler og metoder fra klassen.
- Konstruktører = Genstandsdel

Get og Set

- Get-Metode returnerer en variabel værdi.
- Set-Metode returnerer en værdi.
- Hvad der afhentes og afsættes er den navn som står sammensat med get og set.

```
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

Referer til selve String name.

getName returner en navn værdi fra private String name;

setName tager name-værdien fra String name og derefter sætter i lig med instansvariablen til newName.

Indkapsling

- Nu ser vi en eksempel med Indkapsling, formuleret anderledes.

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.setName("John");
    System.out.println(myObj.getName());
  }
}
```

```
John
```

Pakker

- Vi skal kende til Importering af Pakker
  - For at lave en pakke skal vi sige ”class MyPackageClass”.

Example

```
└── root
    └── mypack
        └── MyPackageClass.java
```

To create a package, use the `package` keyword:

MyPackageClass.java

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

Arv

- Arv, som vi kender betyder Inheritance på Engelsk.
  - Vi plejer, at referere til Subklasse og Superklasse.
    - Subklassen er underklassen, hvorimod superklassen er overklassen.
    - Begge klasser har barn og værge forhold.
- Subklasse: (Barnet) som arver fra en anden klasse
- Superklassen: (Værge) som klassen der bliver arvet fra.

Arv

- Vi skriver keywords, når vi arver fra en klasse til det andet.
  - Vi skriver ”extends” som betyder at vi udvider os til et andet klasse.

```java
class Vehicle {
  protected String brand = "Ford";         // Vehicle attribute
  public void honk() {                     // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}

class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Klassen Car arver variabler fra Klassen Vehicle

Arv

- Hvis du ikke ønsker, at arve noget er der noget som er modsat.
  - Det er keywordet ”final”.

```
final class Vehicle {
  ...
}

class Car extends Vehicle {
  ...
}
```

```
Main.java:9: error: cannot inherit from final Vehicle
class Main extends Vehicle {
                   ^
1 error)
```

Vi kan se, at ”final” forhindrer Car-klassen i at arve fra Vehicle-Klassen.

Vi kan se, at der opstår ”Error”.

Polymorphism

- Poly betyder flere og morphism betyder former.
  - Polymorphism betyder ”flere former”.
  - Polymorphism lader de arvede metoder, at udøve forskellige handlinger.
  - Vi kan med andre orde udøve en handling i forskellige måder.

Indre Klasser og Ydre Klasser

- Indre Klasser
  - Det er også muligt, at lave indre klasser inde i en klasse.
- Ydre Klasser
  - Dem som befinder sig udenfor Indre Klassen.

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Andre ting som relatere indre klasser

- Man kan erklære dem private, for at hindre andre udestående objekter at få adgang til indre klasser. (private class Innerclass){
- Static betyder, at man kan få adgang til objektet gennem ydre klassen.

One advantage of inner classes, is that they can access attributes and methods of the outer class:

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```

Abstrakt Klasse og Metoder

- Man ønsker, at gemme særlige detaljer gennem essentiel information
  - Abstraktion kan fås gennem abstract classes eller interfaces.
- Abstract-Keyworder en non-access modifier som er brugt for klasser og metoder.
  - Abstrakt Klasser er begrænset klasser som kan ikke være brugt til at danne objekter.
  - Abstrakt Metoder kan være i abstrakte klasser, som ikke har en body.

Eksempel

```java
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Vi kan se, at der står extends som resulterer at klassen Pig arver fra Klassen Animal.

Main metoden laves som "crafting table" og her kan vi se, at der laves en objekt.

myPig objekt laves fordi, der står - new Pig - her.

myPig er koblet sammen med AnimalSound og derefter kan det ses at der er lavet en input tegn.

Interfaces

- Kan bruges ligesom ”arv”, med keyword ”interface”.
  - Brugen af keywordet, minder dog meget om Abstract-måden.
  - Interfaces kan ikke oprette objekter
  - Interfaces har ingen ”body”, men kan få det (implementeret) igennem.
    - Implements tillader Interfaces i at få en ”body”.

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

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).

Interfaces Eksempler

- Vi kan se, at vi bruger ordet ”implements” og den optræder som ”extends”.

Vi bruger interface ved klassegivning.

Implements gør, at vi klassen Pig arver fra den ydre interface-klasse Animal.

Vi kan se, at der bliver oprettet 3 metoder. I Main-metode bruges den samme fremgang med at oprette et objekt med navnet myPig.

Den henviser til de to tidligere metoder.

```java
// 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");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Enums

- En special klasse, som kan repreæsentere en gruppe af konstanter.
  - De kan indeholde uforanderlige variabler som eksempelvis (final-variables).
  - Man bruger keywordet ”enum” i stedet for en interface/class og adskiller med komma.

Example

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

You can access `enum` constants with the **dot** syntax:

```
Level myVar = Level.MEDIUM;
```

Eksempler med ENUM

Example

```
public class Main {
  enum Level {
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```

The output will be:

```
MEDIUM
```

```
public class Main {
  enum Level {
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {
    Level myVar = Level.LOW;
    System.out.println(myVar);
  }
}
```

```
LOW
```

Switch-Statement

```
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;
    }
  }
}
```

The output will be:

```
Medium level
```

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.

Loop Through an Enum

The enum type has a `values()` method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum:

Example

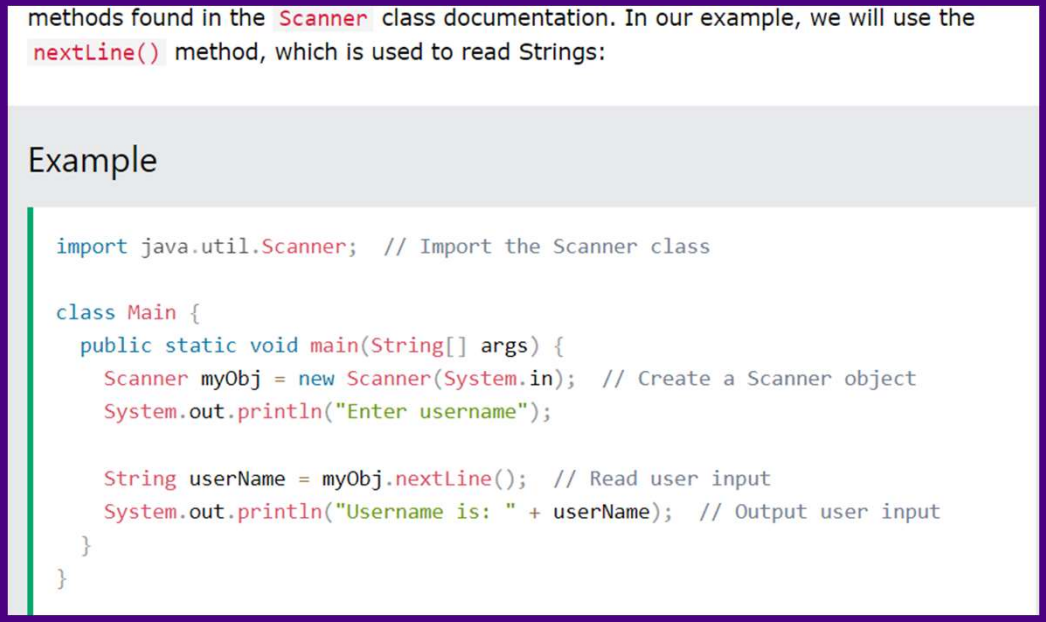
```
for (Level myVar : Level.values()) {
  System.out.println(myVar);
}
```

The output will be:

```
LOW
MEDIUM
HIGH
```

Scanner

- Scanner class er brugt til at få brugerinput, og er fundet i java.util pakke.
- Scanner-klassen er brugt til at lave objekt af en klasse gennem tilgængelige metoder fundet i scanner class.



methods found in the `Scanner` class documentation. In our example, we will use the `nextLine()` method, which is used to read Strings:

Example

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Input Typer

- Her er givet Eksempler fra W3-Schools.

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

ArrayList

- Man kan i denne sammenhæng danne en ArrayList, nemt ikke?
  - Om lidt viser vi et eksempel, hvor man kan tilføje ting i listen.

```java
import java.util.ArrayList;

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");
    System.out.println(cars);
  }
}
```

```
[Volvo, BMW, Ford, Mazda]
```

To access an element in the `ArrayList`, use the `get()` method and refer to the index number:

Example

```java
cars.get(0);
```

```
Volvo
```

Change an Item

- For at kunne ændre en element, skal vi anvende set() og derefter referer til index nummeret. Her kan vi se, at vi har følgende.
  - Gennem følgende eksempel har vi lige sat en nyt array op.

```
cars.set(0, "Opel");
```

  - Det samme er også med remove-eksemplet.

```
cars.remove(0);
```

Vi kan fjerne alle elementerne gennem følgende eksempel.

```
cars.clear();
```

Arrayist Size

- For, at kunne finde ud af ArrayListens længde har vi følgende eksempel.

```
cars.size();
```

```
import java.util.ArrayList;

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");
    System.out.println(cars.size());
  }
}
```

Sorter i ArrayList

- Man har en pakke som hedder Collection Class, hvor man kan sortere.

```java
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

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");
    Collections.sort(cars); // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

```
BMW
Ford
Mazda
Volvo
```

Collections.sort(cars) er den måde vi sortere i ArrayListen på.

Vi kan se, at for-lykken er lavet og derefter er der lavet en udprint.

Del 1 (Programudvikling)

Vi præsentere de udviklingsmæssige ting her.

Exceptions

- Hvad er Exceptions?
- Hvornår bruges de?
- Hvad har de af arbejde?

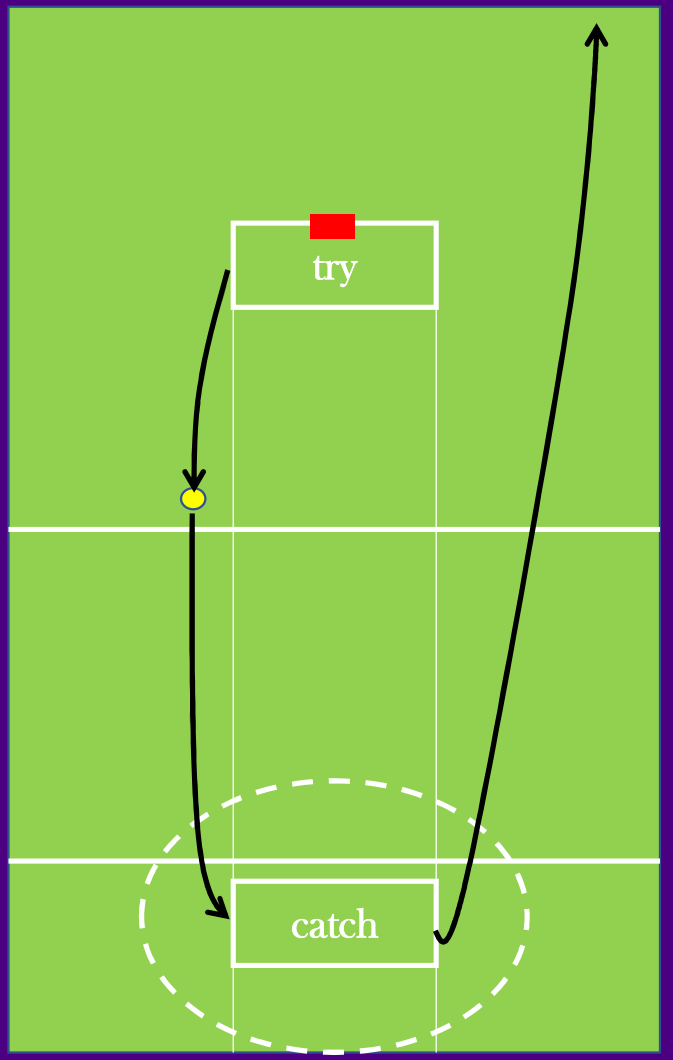
Exceptions (Cricket Eksempel)

- Se på figuren til højre ☺
  - Try er vores bowler, som kaster bolden.
    - Bolden er vores problem.
  - Catch er vores batter, som teknisk får bolden.

Forstået på programmeringskode:

Try er kode som kan potentielt indeholde fejl.

Catch er der hvor man gør noget ved fejlen!



Exceptions

- Når du ønsker, at køre et problem igennem, skal du bruge try.
- Catch skal analysere koden igennem og løse problemet.
- Final er det som ”overruler/overrider” begge statements igennem.
  - Teknisk set er Final Dommeren her ☺

- Vi skal se eksempel på Kode
  - Kommer snart efter opdatering i Præsentationen!