Java refresher

August 19, 2025

1 Setup Java

Henrik har måske givet en guide til dette på Brightspace allerede, så kig der.
Du skal bruge Java JDK Standard Edition. Jeg tror 21 er den rigtige version (det var den da jeg tog kurset, og jeg tror ærligt ikke det gør en stor forskel).
Link to Java SE Downloads
Installations video

Download og installer IntelliJ comunity edition:
Link to intelliJ download
Installations video

Brug denne side til mere info (Jeg linker til nogle af de vigtige jeg ikke kommer ind på. Men du kan altid slå op her): java tutorials

2 First program

Næsten alt i Java er et objekt derfor skal vores main metode også være pakket ind i klasse. Lav et projekt med en Main classe (en Mail.java fil med nedenstående i).

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello World");

    }
}
```

Kør dette program og se den skrive "Hello World".

2.1

Modificer dette program så den skriver dit navn & alder. TIP: I IntelliJ kan man skrive sout og så trykke tab, så kommer hele "System.out.println();" op automatisk.

2.2 Primitive typer

I java er der primitive typer som ikke er objekter. De tre mest normale er: int, double(double precision float) & boolean.
Lav en variabel (vægt(double), alder(int), dreng(boolean)) af hver af disse typer og print disse variabel. For at lave en variabel skal man sige til Java hvilken type den er, dette kaldes "statisk typecheck" og er en af de store forskelle på Python og Java.

```
public class Main {
    public static void main(String[] args) {
        double v_gt = 97.8;
        System.out.println(v_gt);
```

```
        ...
    }
}
```

2.3 Objekter og klasser 1

I java starter ALLE klasser med stort, dette gør at det er nemt at holde styr på hvad er et objekt og hvad ikke er et objekt. Dette betyder også at når DU laver klasser skal de starte med stort, og når DU IKKE laver klasser skal de starte med småt.
For at lave en String skal man skrive

```
String navn = "...";
```

lav en string med dit navn og print det.

2.4 Objekter og klasser 2

Nu skal vi lave din egen klasse. Lav en ny fil med navn Student.java. Og lav en public class med navn Student.

```
public class Student {

}
```

For at man kan instantierer et objekt fra en klasse, skal klassen have en konstruktor. Dette gør man ved at skrive:

```
public class Student {
    public Student(){

    }
}
```

Denne konstruktor kan godt have parametre og bruges til at "starte et objekt op".

En klasse har også "feltværdier", som er en række variable som et objekt kan tilgå. Disse skal for det meste være "private" for at sikre hvem der kan tilgå dem direkte.

```
public class Student {
    private int alder;
    public Student(){

    }
}
```

NOTE: feltværdier plejer ikke at blive tildelt nogen værdi i deres deklaration I toppen. Dette plejer man at gøre i konstruktoren:

```
public class Student {
    private int alder;
    public Student(){
        alder = 24;
    }
}
```

Og dette kan selvfølgelig gøres mere generelt med parametre i konstruktoren. Her skal man være opmærksom på "shadowing", altså at man bruger det samme navn til flere variable. Man kan altid tilgå felværdier for et bestemt navn ved at skrive "this." foran, for at vise man bruge "dette objekts" variabel.

EKSEMPEL 1 (shadowing)
```
public class Student {
    private int alder;
    public Student(int alder){
        this.alder = alder;
    }
}
```

EKSEMPEL 2 (no–shadowing)
```
public class Student {
    private int alder;
    public Student(int age){
        alder = age;
    }
}
```

Færdiggør denne klasse med alder, navn & student-ID. (parameter listen bliver sepereret med kommaer)

2.5 Metoder & instantiering 1 -

Metoder er funktioner der tilhører en bestemt klasse. Man kalder en metode via "variabel.metode(argumenter)". Vi laver en toString() metode i Student. Den består af en adgangmodifikator (public/private/protected), en returtype (String), et metode navn (toString) og en parameter list (blank i dette tilfælde) I nedenstående kode bruger jeg aa i stedet for å, det er en LaTex fejl, java kan godt tage å.

```
public class Student {
    private int alder;
    private String navn;
    private String id;
    public Student(int alder, String navn, String id) {
        this.alder = alder;
        this.navn = navn;
        this.id = id;
    }

    public String toString(){
        return navn + ", " + alder + "aar: " + id;
    }
}
```

For at bruge et objekt skal vi instantierer det. Dette gøres ved at bruge keywordet "new":
Student bosch = new Student(24, "Bosch", "abc123-02");
Lav 3 forskellige students i main, og print deres værdier med toString metoden.

TIP: toString() er en metode ALLE objekter har, så man behøver ikke skrive bosch.toString() i print metoden, da Java selv bruger den for at lave objektet om til en String. Hvis man ikke selv giver en toString() metode kommer objektets lokation i memory ud (hvilket sjælendt er brugbart).

NOTE: Du så tidligere at for at instantierer en String skulle man ikke bruge new. Dette er noget speciel syntaks

2.6 Metoder 2 - settere og gettere

Lige nu er det kun objektet selv der kan tilgå feltvariablene, da de er private. Derfor er der et normal "pattern" i java der hedder "settere" og "gettere". Som er metoder der enten retunerer en feltværdi eller sætter en felt værdi. Nedenunder er eksempel på setter og getter metoder:

```
    public int getAlder(){
        return alder;
    }

    public void setAlder(int alder){
        this.alder = alder;
    }
```

Note: At lave gøre ens feltvariable "public" gør at man bare kan tilgå dem via "variabel.feltvariabel", her kan man både læse og skrive direkte til feltvariablen. Dette er for det meste dårlig praksis, og da du skal lærer om software arkitektur er det bedst at gøre feltvariable private og lave gettere/settere, når de skal bruges.

Lav setter og getter til alle feltvariablerne i Student. Prøv derefter at printe en feltvariabel i main, ændrer den og print den igen.
TIP: I IntelliJ kan du generer kode. Dette gøres via alt+insert, og de kan genrerer gettere, settere + mere.

2.7 Metoder 3 - opbygning af klasser

Rækkefølgen tingene kommer I har ingen forskel I en Java klasse, men det er god praksis at strukturerer sin klasse sådan:

1. Feltværdier
2. Konstruktor
3. settere og gettere
4. Resten af metoderne

2.8 Collections and loops

For at lave en liste af Students skal man importerer fra "java.util".
Det er god stil kun at importerer det man skal bruge, men hvis man skal bruge meget fra util, så kan man importerer det hele ved at skrive "import java.util.*"
Øverst I main, importer java.util.ArrayList. Derefter lav en arrayList, med Students i. Note: du behøver ikke lave variabler til dem alle, du kan gøre som til sidst nedenfor:

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Student> studentList = new ArrayList<Student>();
        Student anders = new Student(12, "Anders", "2025-8765");
        Student bertha = new Student(88, "Bertha", "1995-2641");
        Student cecilie = new Student(24, "Cecilie", "2022-7535");
        studentList.add(anders);
        studentList.add(bertha);
        studentList.add(cecilie);
        studentList.add(new Student(55, "Djon", "8888-1234"));
    }
}
```

Collections I Java har brug for at vide hvad de skal indeholde, derfor er der ¡Student¿ bag ved deklerationen. Det vil også sige at du ikke kan putte andre objekter ind i denne liste.

Lister bruges ofte i loops, her har du tre forskellige måder at lave loops på: for-in, for & while:

```
for (Student s : studentList){
    System.out.println(s);
}
```

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

```
int i = studentList.size()-1;
while (i >= 0 ){
    System.out.println(studentList.get(i));
    i--;
}
```

Prøv dem alle af i din main metode. While loops kommer du ikke til at bruge særligt meget. Men man kan nogle ret avanceret ting med while loops, som man ikke man med for loops.

2.9 Nedarvning 1

Lav en ny klasse "Teacher" der er en kopi af Sudent. Lav 3 Teacher objekter.
Giv Teacher en teach metode der bare printer "Teaching", giv student en study metode der bare printer "Studying". Dette er bare for at de gør noget forskelligt.

Nu står vi med to problemer,

1. Vi vil gerne kører en liste igennem med både teachers OG student
2. Vi har kode dublikation, der ikke burde værer (gettere settere og toString metoden på tværs af Student og Teacher)

Disse problemer kan løses med nedarvning!!!!

Lav en ny klasse Person, som indeholder alt fra Student, undtagen study metoden. Vi kan nu "genbruge" koden via nedarvning. Ændre Student til det følgende:

```
public class Student extends Person{

    public Student(int alder, String navn, String id) {
        super(alder, navn, id);
    }

    public void study(){
        System.out.println("Studying");
    }

}
```

"extends" keyworded bruger man til at nedarve metoder til en ny klasse. Dvs at alle public metoder som Person har, er noget Student har nu.
"super" keyworded minder lidt om "this", man refferer altså ikke til sig selv, men til den klasse man nedavre fra, altså super klassen. Note: den anden vej hedder sub klasse.

Lav også nedarvning fra Person til Teacher.

I main, kan du nu lave en ArrayListe med Person objekter. En liste kan indeholde klassen selv eller en hvilken som helst sub klasse. Dvs vi kan både have Teacher objecter og Student objekter i listen. Gør dette med alle dine objekter og print deres toString.

2.10 Nedarvning - Overriding metoder

Lav en ny klasse der hedder LazyStudent, der nedarver fra Student. TIP: IntelliJ kommer til at klage over at du ikke har en konstuktor når du nedarver. Du kan få IntelliJ til at løse mange af dine problemer ved at klikke der hvor der er rød, trykke alt+enter, og vælge en handling.

Giv den en ny study metode, der printer "not studying".
Nu har du overskrevet en super klasses metode, DVS at LazyStudent bruger dens egen metode og ikke super klassens metode.
Det er en god idé at sige til Java at det er formålet, fordi så kan Java hjælpe med at finde fejl hvis du nu sletter metoden i en super klasse, men ikke i sub klasserne.
Du fortæller java at det er din intention at overskrive via:

```
@Override
public void study(){
    System.out.println("Not Studying");
}
```

2.11 Interfaces

Der er en dårlig ting ved nedarvning. Du kan kun nedarve fra en klasse af gangen og nogle gange har man brug for at nedarve flere "egenskaber". Du kan tilgengæld nedarve fra flere interfaces. Et interface er noget "abstrakt" kode, som egentlig bare siger at et objekt der implementerer dette interface, har også implementeret disse metoder. F.eks. vil vi gerne have både Student og Teacher har uni-ID, men det skal Person klassen ikke have.
Giv både Student og Teacher en String feltvariabel ved navn uniID og en metode der hedder getUniID().

I main vi vil gerne kunne have en liste af Students og Teachers, hvor vi kan kalde getUniID(), på alle. Derfor laver vi et interface som nedenunder:

```
public interface UniversityIdentifiable {
    public String getUniId();
}
```

Og siger at Student og Teacher skal implementerer dem:

```
public class Student extends Person implements UniversityIdentifiable{
    public String uniID;

    public Student(int alder, String navn, String id) {
        super(alder, navn, id);
        uniID = "AU–" + id;
    }

    public void study(){
        System.out.println("Studying");
    }

    public String getUniID(){
        return uniID;
    }
}
```

I main, lav nu en liste af "UniversityIdentifiable" og kald deres getUniID metoder.

NOTE: EN klasse kan implementerer flere forskellige interfaces.

2.12 Metoder - Overloadning

Man kan det der hedder Overloading. Det betyder at man kan have flere metoder med samme navn, men med forskellige parameter lister. Dette kan både være i konstuktøren og i de normale metoder. Så disse metoder kan alle være der på samme tid i Student klassen:

```
    public Student(int alder, String navn, String id) {
        super(alder, navn, id);
        uniID = "AU-" + id;
    }
    public Student(int alder, String navn, String id, String uniId) {
        super(alder, navn, id);
        this.uniID = uniId;
    }

    public void study(){
        System.out.println("Studying");
    }

    public void study(String topic){
        Systen.out.println("I am studying " + topic);
    }
}
```

3 Extra

3.1 Nedarvning extra - type casting

Du ved i hvilken rækkefølge objekterne kommer i, i ArrayListen. Så prøv at kald det første objekts teach eller study metode (alt efter hvilket objekt det er) via `personList.get(0).study()`. Du vil kunne se at det kan du ikke selvom DU er sikker på det er en Student/Teacher. Det er fordi Java kun ved det er et Person objekt.
For at kalde metoden skal man lave type cast, altså fortælle Java hvilket objekt det rent faktisk er. Dette gøres med paranteser før objektet:

```
Student s = (Student )personList.get(0);
s.study();
```

Dette er ikke noget du skal bruge før sidst på kurset.

3.2 Ting du også skal vide

Du skal vide noget om pakkemanagement og imports: link
Exceptions: exceptions
Klasse variable: klasse variable
Klasse metoder
Måske abstrakte metoder/klasser, Jeg kan ikke huske om SWEA kom ind på noget af det: link