

**Formål:** Vi repeterer **ALT** hvad vi har gennemgået!

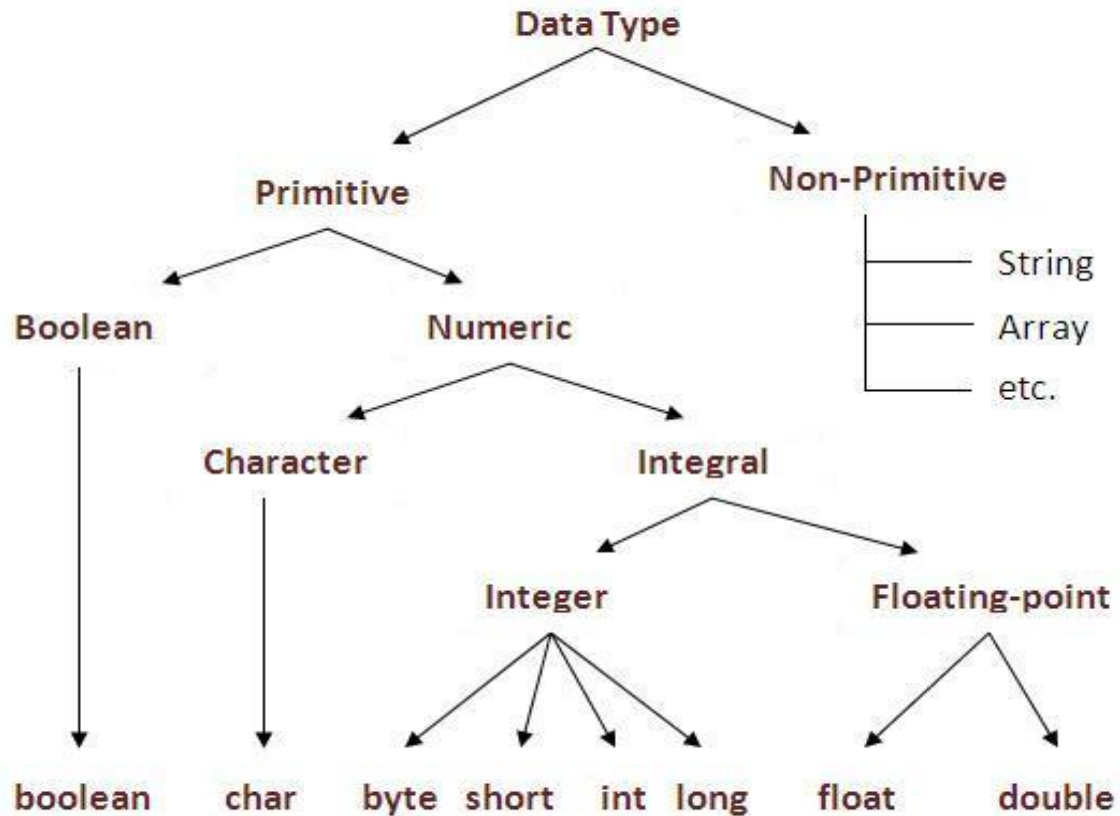
## Emner:

- Simple typer
- If..Then &&, ||, <=, >=, !=
- For..Next
- Klasser, metoder, konstruktører, parametre og returværdier, import
- Encapsulation, Private, **This**, Getter, Setter

**Litteratur:** Reges & Slides fra undervisningen

# Data Typer

- Hvad er en type?
- Hvad er primitive typer?
- Hvad er komplekse typer?



	Size	Default Value	Type of Value Stored
byte	1 byte	0	Integral
short	2 byte	0	Integral
int	4 byte	0	Integral
long	8 byte	0L	Integral
char	2 byte	'\u0000'	Character
float	4 byte	0.0f	Decimal
double	8 byte	0.0d	Decimal
boolean	1 bit (till JDK 1.3 it uses 1 byte)	false	True or False

**Type:** Javas typer angiver to ting: Dels får vi reserveret 4 bytes i hukommelsen, dels angiver vi hvordan bitmønstret skal læses!

**Int myNumber = 1;**

**Identifier:** Vores unikke navn til vores datafelt. Ved at kalde vores typer unikke navne kan vi altid referere til dem. Ligesom mennesker har unikke navne!

# IF..Then

```
if ( [type] [operator] [type] ){  
    <KODE der køres når IF er opfyldt>  
}  
else {  
    <KODE der køres når IF IKKE er opfyldt>  
}
```

Operatorer der kan bruges på Boolske udtryk

Operator	Result
&	Logical AND
	Logical OR
^	Logical XOR (exclusive OR)
	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to

Typen kan være **Primitiv** (Int,Short,Boolean...) Men kan også være en **String**, eller en **metodes return-parameter** eller et udtryk der er **boolsk**  
BEMÆRK: Typerne skal være af samme format på begge sider af operatoren.  
Et udtryk hvor man fx skriver:  
If( 1<3) && ("hest"!="hund") er fint fordi begge typer ender med at være boolean's 😊

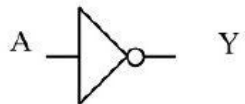
Operatorer der kan bruges på alle **typer**

Condition	Operator	Example
Is equal to (or "is the same as")	==	int i=1; (i == 1)
Is not equal to (or "is not the same as")	!=	int i=2; (i != 1)
Is less than	<	int i=0; (i < 1)
Is less than or equal to	<=	int i=1; (i <= 1)
Is greater than	>	int i=2; (i > 1)
Is greater than or equal to	>=	int i=1; (i >= 1)

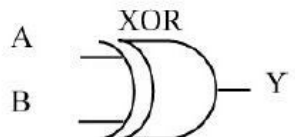
# Elektrisk & Logisk 😊

## Basic Logic Gates

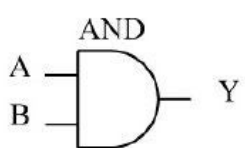
A	Y
0	1
1	0



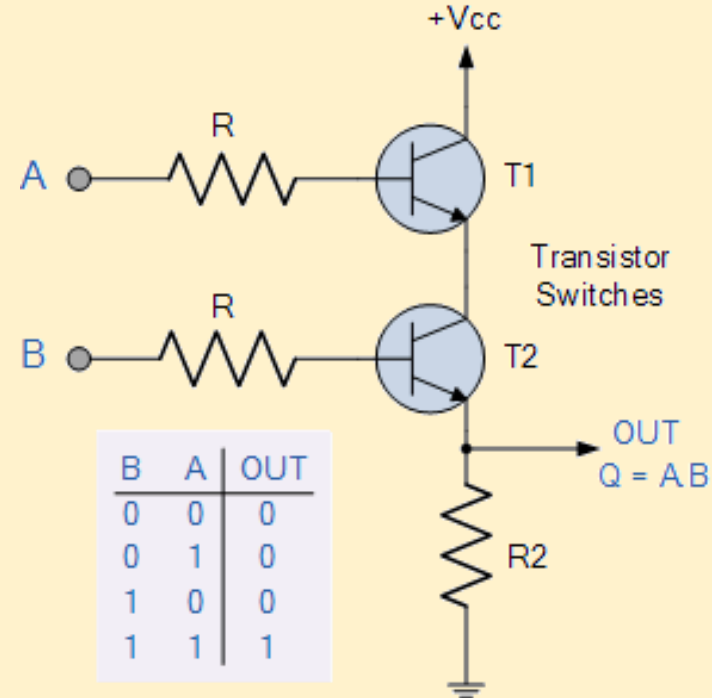
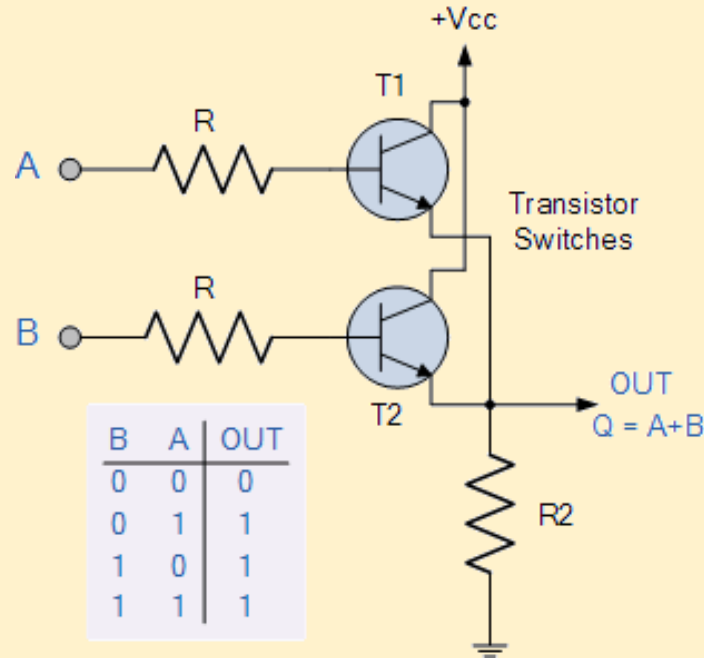
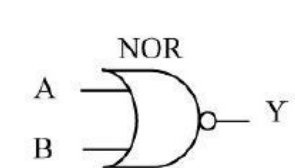
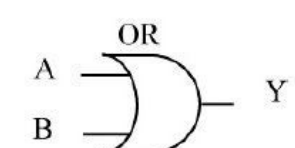
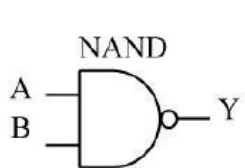
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

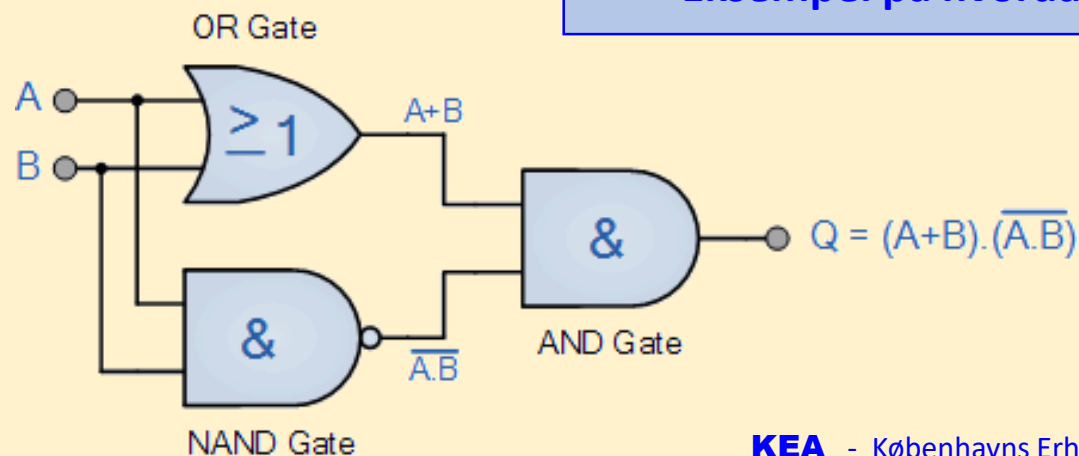


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



Eksempel på hvordan man elektrisk skaber en AND og OR !

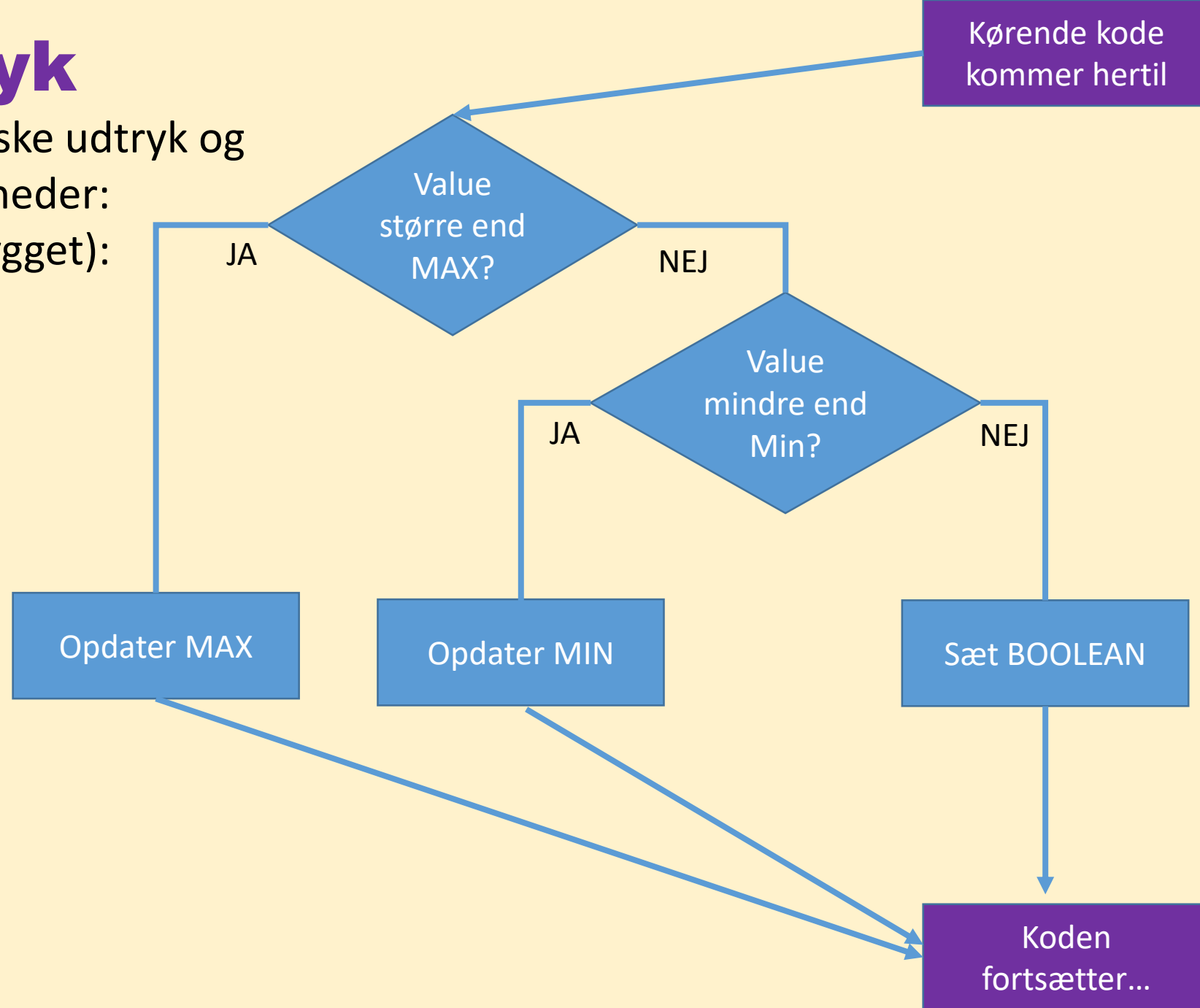
Eksempel på hvordan man bygger XOR ud fra AND og OR kredse !!!



# Komplekse udtryk

IF THEN skal ses som strengt logiske udtryk og kan bruges i et netværk af muligheder: I bogen er et eksempel (lidt udbygget):

```
If (value > max){  
    max = value;  
} else if (value < min) {  
    min = value;  
} else {  
    averageValue=true;  
}
```



# If..Then..Else

```
//udskriv hund
```

```
if (dog2.getAge() <= 1) {  
    System.out.println("Her er en spritny hund:");  
    System.out.println( dog2.getDogData() );  
} else {  
    System.out.println("hunden her er ikke en hvalp:");  
    System.out.println( dog2.getDogData() );  
}
```

Betingelsen er at metoden returnerer en alder mindre end eller lig et år og efterfølgende er tilføjet Else. Else udføres i **alle andre** tilfælde hvor IF ikke er udført.

# Simpelt For Loop

```
for ( [start] [slut] [tæller] ){  
    <KODE der køres når slut ikke er opfyldt>  
}
```

Vi opretter en tæller variabel og sætter en start værdi (senere ser vi på loops der bruger andre typer😊)

Her fortæller hvordan tælleren kører i hvert loop (fx: cnt++)

Når vores tæller variabel når slutværdien stopper løkken med at køre og der fortsættes i koden efter loopet.

# Eksempel: For løkker

```
import java.util.*; //Inkluder java's package! * Reges kap.3 :)
public class ForNext { //koden her er uden underklasse med objekter :-))
    public static void main (String[] args){
        int dice = -1; //Terningen er minus 1 så vi ved om den har været slået inden vi kigger på den :-)
        Random r = new Random();

        System.out.println("Tælleren cntX behoever ikke foelge cntX++ - her tæller vi 2 op ad gangen");
        for (int cntX=2; cntX <= 20; cntX=cntX+2){
            System.out.println("Her er cntX = "+ cntX);
        }
        for (int cntLoop1=1; cntLoop1<=10; cntLoop1++){ //Computeren prøver 10 gange at slå en sekser på 3 slag
            System.out.print(cntLoop1+" gang faar vi: ");
            for (int cntRollDice=1; cntRollDice<=3; cntRollDice++){
                dice = r.nextInt(6)+1; //tilfældigt tal mellem 1 og seks
                System.out.print(" slag#" +cntRollDice +": " +dice);
                if (dice==6 ){
                    System.out.println("<--- Her lykkedes det!!! :-)");
                    break; //vi behøver jo ikke flere slag vi har fået en sekser og forlader loopet ved at sætte tælleren til slutværdien
                } else {
                    if (cntRollDice>=3 & dice != 6){ //Vi har her en ifsætning hvor BEGGE udsagn skal være opfyldt
                        System.out.println("<---- Ingen sekser her :-(");
                    }
                }
            }
        }
    }
}
```

Mest simple løkke

Bemærk i stedet for cnt++ som man altid ser hopper vi to ad gangen (normalt bruger vi cnt++ og ganger bare tæller med de steps vi ønsker)

En nested konstruktion – en løkke i en løkke (bemærk at vi ændrer på den indre løkke for at komme ud – her ville man måske bruge en while opbygning – kommer vi til senere😊)



# Konstruktør / Constructor

Når et Objekt erklæres, vil det ofte være bekvemt at kunne oprette det i en **begyndelsestilstand**. Vi har tidligere set med eksempel at **variable erklæres med en forud defineret værdi**. Med en konstruktør kan objektet startes op med forskellige startværdier. fx gem et userID på den medarbejder der opretter en motorcykel i systemet.

- En konstruktør har eksakt samme navn som klassen!
- Der er altid en konstruktør! Hvis du ikke erklærer en vil den være der 'usynligt'!
- Det vil ofte være et ønske at benytte parametre i forbindelse med en konstruktør!

# Hovedprogrammet med constructor

```
//definer hundeklasse (Vi ser på PARAMETRE CONSTRUCT og OVERLOAD)
```

```
public class Dog{
```

```
String name = "<blank>";
```

```
String color = "<hollow>";
```

```
int age = -1;
```

```
double weight = -1;
```

```
String barking sound = "Silent";
```

```
//Konstruktøren oprettes public Hund = klassenavnet
```

```
public Dog(){
```

```
//konstruktøren overskriver name som er sat til at være <blank>
```

```
name = "Ikke oplyst";
```

```
}
```

```
// BEMÆRK : Her kommer der endnu en konstruktør? OVERLOADING
```

```
// Når vi ønsker at kunne bruge samme metode/konstruktør med forskellige parametre
```

```
// På denne måde kan vi oprette hunde-objekter både med og uden navn
```

```
public Dog(String newname){
```

```
name = newname;
```

```
}
```

# Encapsulation

Private – vores variable er nu indkapslede!!!

```
//definer hundeklasse (Vi ser på PARAMETRE CONSTRUCT og OVERLOAD samt PRIVATE)  
public class Dog{
```

```
    private String name = "<blank>";  
    private String color = "<hollow>";  
    private int age = -1;  
    private double weight = -1;  
    private String barkingSound = "Silent";
```

```
    //Konstruktøren oprettes public Hund = klassenavnet
```

```
    public Dog(){  
        //konstruktøren overskriver name som er sat til at være <  
        name = "Ikke oplyst";  
    }
```

## BEMÆRK:

Private er det nye reservede ord vi nu kan bruge.

Når vi skriver private foran en variabel betyder det at disse variable indkapsles – dette kaldes ENCAPULATION og gør at variablerne slet ikke kan ses uden for objektet.

Vi bruger nu udelukkende metoder til at hente eller sætte værdier for disse variable. Metoder der gør dette kaldes "Get'ere og Set'ere" fordi vi typisk vil navngive metoderne med getParamX eller setParamX.

# Kodeeksempel med parametre

```
//Metode til at sætte hundeparametre
public void setdogdata(String newname, String newcolor, int newage, double newweight, String newbarkingsound){
    name = newname;
    color = newcolor;
    age = newage;
    weight = newweight;
    barkingsound = newbarkingsound;
}

//metode til hundelyd
public void dogbarking(){
    System.out.println(name + "\t siger:\t" + barkingsound);
}

//hundelyd overloadet - nu med mulighed for at definer ny lyd
public void dogbarking(String newbark){
    barkingsound = newbark;
    System.out.println(name + "\t siger nu:\t" + barkingsound);
}
```

Bemærk:  
To metoder med SAMME navn  
Én med og en uden parametre.

# Returparameter

Når vi kalder en metode med returparameter vil vi på en enkelt måde kunne skrive og arbejde med kode. Fx. Kan en metode returnere en boolean.

Nogle i klassen sad og kodede et bookingsite for et flyselskab og ønskede en metode hvor man kunne få et logisk svar. Det har de nu! 😊

**Fx: If flightPassengerList.booked(flightName, passportNr) then.....**

***Altså en metode hvor vi kalder et objekt med metoden flypassagerliste med et flynr. Og et passnr. Og får en True/False retur der fortæller om personen er booked på flyet!***

- En metode der ikke er 'void' skal altid slutte med en return-kommando
- En metode med returværdi bruges som den variabel-type (attribut) den returnerer. Returneres en boolean bruges metoden som logisk udtryk, er det en int bruges den som et heltal osv.
- Med parametre og returværdi bliver vores metoder pludselig meget kraftfulde og i ser nu hvordan objekter kan styres udelukkende gennem deres metoder.

# Kodeeksempel med returparameter

```
public class Dog{
```

```
    String name = "<blank>";  
    String color = "<hollow>";  
    int age = -1;  
    double weight = -1;  
    String barkingSound = "Silent";
```

```
    public Dog(){  
        name = "Ikke oplyst";  
    }
```

```
    public Dog(String name){  
        this.name = name;
```

BEMÆRK: VIGTIG at kende **THIS**  
Den er normalt underforstået –  
men her skal den bruges for at  
sikre forskellen mellem  
parameteren i metoden og  
objektets egen variabel da de  
begge hedder name 😊

Metoden returnerer en streng  
BEMÆRK: Når kommandoen return  
anvendes er det afslutningen på  
metoden. Der kan altså ikke føjes flere  
kodelinier til metoden efter return er  
kørt!

//BEMÆRK: this bruges til at skelne objektets variabel name fra metoden parameter da de har samme navn!!!

```
    //metode til give en tekststreng af hundedata
```

```
    public String getdogdata(){
```

```
        return "Navn:" + name + "\n Farve:" + color + "\n Alder:" + age + " Aar \n Vaegt:" + weight + " kg \n";  
    }
```

```
    public int getage(){  
        return age;  
    }
```

# LIDT MERE OM THIS....

This kan referer til flere implicitte elementer indeni klassen:

- En variabel/field/identifiser: `this.field` (Det som vi kender allerede😊)
- Et metodekald `this.method(parameters);`
- Eller... en konstruktør  
der kalder en anden konstruktør: `this(parameters);`

VÆR NYSGERRIGE  
Husk at LÆSE - Husk at KODE  
Man lærer konstant nye ting. JAVA  
er som alle andre sprog –  
mangfoldigt og fuld af nye  
muligheder og syntakser 😊

Hvorfor bruge `this.method();` ?? Det bruger vi når vi arbejder med 'sen binding' dvs. vi kalder en metode der findes i flere objekter og under afviklingen ved vi ikke hvilket objekt vi arbejder med – `this` tillader os at vælge under kørslen 😊  
Vi kommer tilbage til begrebet Sen Binding / Dynamisk Binding 😊

# Konstruktør kalder konstruktør...

- Vi kigger lige på hvad der sker her...

```
public class Point {  
    private int x;  
    private int y;  
  
    public Point() {  
        this(0, 0); // kalder (x, y) constructor  
    }  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Vi ser:

To constructors (Overload)

Vi ser at Point kalder sin tvilling  
med this og parametrene angiver  
hvilken construtor vi kalder.

**BEMÆRK:** Metoder kan IKKE  
kalder constructors!



# Repetitions øvelse

**Etablér 6 grupper, præsenter jeres oplæg næste gang!**  
**(SKRIV JER OP PÅ EN AF GRUPPERNE!!!)**

1. Vis kode med **Klasser** og **Objekter** og forklar hvad der sker undervejs i koden – **new()**. Forklar også hvad en **metode** er, og hvad ideen med metoder er.
2. **Metoder:** Forklar hvad **Getter** og **Setter** er. Forklar hvordan **paramtre** og **returværdier** bruges og vis med kodeeksempel hvordan syntaksen er.
3. Hvad er en **Konstruktør**? Hvad er **Overloading**? Vis kodeeksempler hvor **new()** anvendes.
4. Forklar hvad **Encapsulation** går ud på. Hvad er **private**? Og hvorfor findes **this**. Og skal den altid bruges?
5. **Conditions:** Forklar hvad **If..Then..Else** gør i vores kode og gennemgå en forklaring. Kom ind på **boolske udtryk** – forskellen på fx **a=5** og **a==5** og **"|"** og **"||"** samt **"&"** og **"&&"**
6. **For-Loop'et:** Hvad er **For** for en konstruktion og vis en stump kode med et **nested loop**. Forklar kort hvad **Random** er og hvor vi får objektet fra - **import**. Hvad gør **break**?

# Øvelse-I (gem koden til næste gang)😊

Opret en klasse af typen Animal – et dyr har en artsbetegnelse (species) og en x-koordinat og en y-koordinat.

- 1) Opret en **konstruktør** der kaldes med arten som parameter og lad desuden konstruktøren vælge en vilkårlig koordinat mellem 1 og 100 for x og for y koordinaten.
- 2) Opret en **Getter-metode**, getSpecies, som returnerer arten på dyret.
- 3) Opret en **is-metode**, isAnimalHere, som har x og y som parameter og som returnerer en boolean hvis de to koordinater svarer til dyrets koordinater.

I hovedprogrammet - Jungle – køres følgende:

1. opret to-tre dyr ved hjælp af **konstruktøren** i Animal
2. Skriv en **For-Loop** struktur som gennem søger alle koordinater og udskriver dyrenes art og findested (koordinater)

- Opret en **konstruktør** med parameter
- Benyt **private** variable i klasserne
- Brug **this** foran jeres klasse-variable når de anvendes i metoderne
- Brug **IS**-metode
- Se hvor der kan anvendes **For-loop**
- Benyt **If-then**

HUSK: Brug metoder direkte i for-løkker, if-sætninger eller ved System.out.😊

Umi, 4

①

NADEA

Maria

Ali

amir

②  
TOBIAS, JAKOB<sup>C</sup> BRAVN  
JONATHAN, JONAS

③  
Emil  
Jakob B.  
JAKOB M.  
Jacob Bay  
~~Christian~~

Jasper P  
Christian J.

④

Cem  
Ghazal

Benjamin  
Theis  
Daniel  
mudi

⑤

Frederic T  
Kristoffer  
Magnus

⑥

Formål: At forankre vores kompetencer og skrive god kode 😊

Emner: Vi repeterer **ALT** og vi bruger det vi har lært

Litteratur: Reges: Reges!!!

**NÆSTE GANG  
i Modul 7&8**

*Vi arbejder med **klasser**, **objekter** og **metoder**. Vi bruger **parametre** og **returværdier** og husker **private**-variable og **this**-begrebet så vores metoder kan bruge samme navne i parameterlisterne. Vi initialiserer vores objekter med en **konstruktør** og vores kode rummer nu **if..then** (med logiske operatorer som **&&** og **||**) og løkker af typen **For..Next**.*