

# IDAT1001 Programmering 1

Objektorientert programmering

Mål for dagen:

Objekter og klasser

Sammenhengen mellom klasse og bruken av klassen

Referanser og objekter

Konstruktører og initiering av objektvariabler

Metoden toString()

Språkkjerne, klasser, forenklet utgave

Nye begreper

Surya Kathayat



**GitHub**  
Copilot

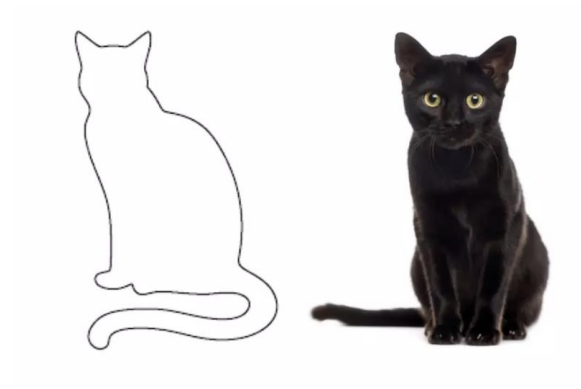
AI pair programmer

# Læringsutbytter, forelesning 3 - løkker

- Identifisere kontrollstrukturen **løkke** i en problembeskrivelse
- Illustrere kontrollstrukturen løkke ved hjelp av **aktivitetsdiagram**
- Identifisere de ulike delene av kontrollstrukturen: **initiering, oppdatering, løkkebetingelse, løkke kropp**
- Anvende **operatorene** ++, --, +=, -=, \*=, /= og %=
- Anvende riktig setning (**for-, while eller do-setningen**) for å programmere en løkke
- Sette opp testdata for å validere programkoden for løkke
- Programmere nøstede kontrollstrukturer (løkker og valg)
- Lese programmer der **break** og **continue** er brukt

# Læringsutbytter, forelesning 4


- Forklare forskjellen på objekter i et program og objekter i den virkelige verden
- Identifisere objekter i en problembeskrivelse
- Anvende ferdige klasser
- Programmere egne klasser med objektvariabler, konstruktører, toString() og metoder
- Forklare hvordan begrepene melding, klient og tjener avspeiles i programkode
- Forklare forskjellen på referansetyper og primitive datatyper
- Beskrive klasser gjennom UML-Klassediagram



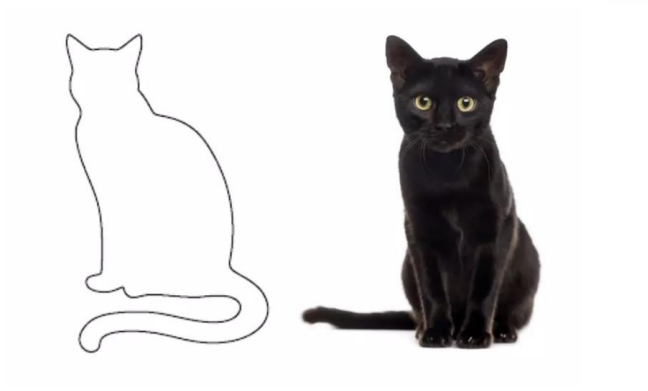
# Class and Object

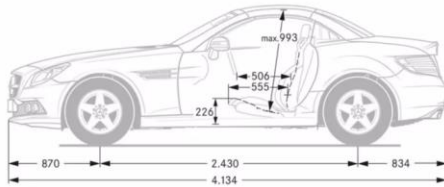
# Objekter og klasser

<https://www.youtube.com/watch?v=CPUaTT0Xoo4&list=PLYPWr4ErjcnzWB95MVvIKArO6PIfv1fHd>



My bike is a clean,  
red, racing bike



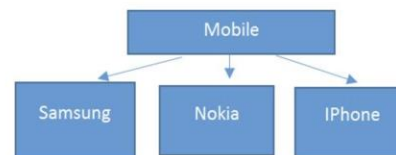
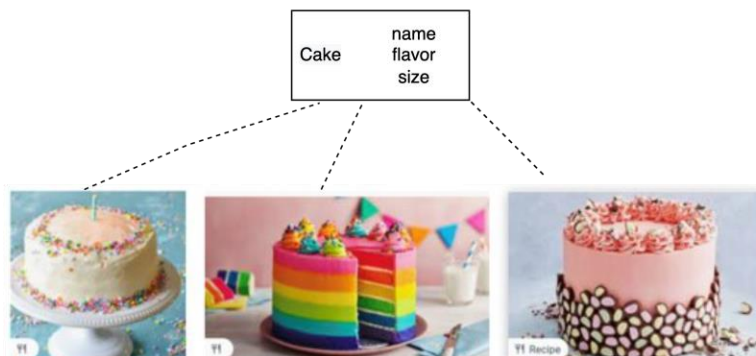
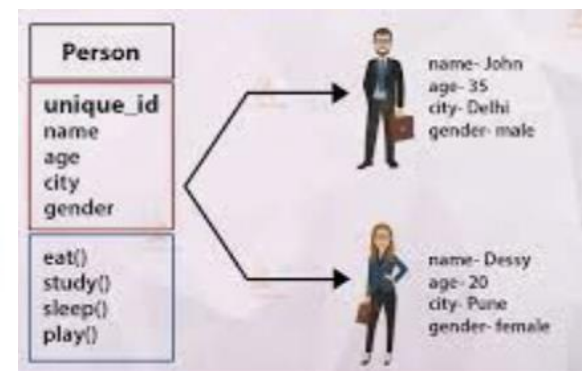
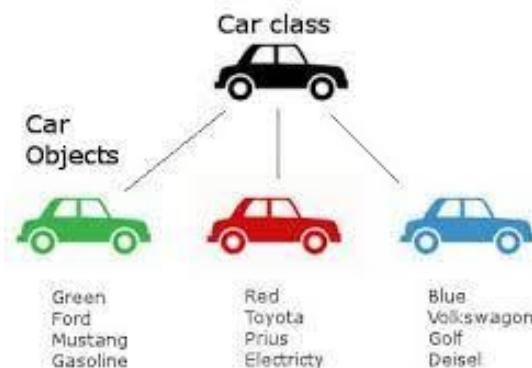
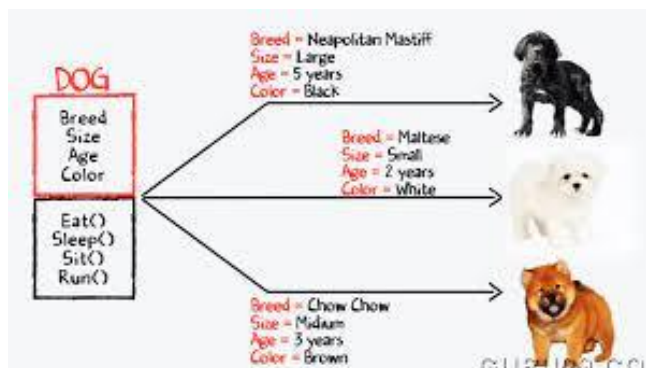


# Objekter og klasser - eksempler

Klasse – kategori eller type  
eller template eller blueprint

instance

Objekt – individuell  
item eller instance



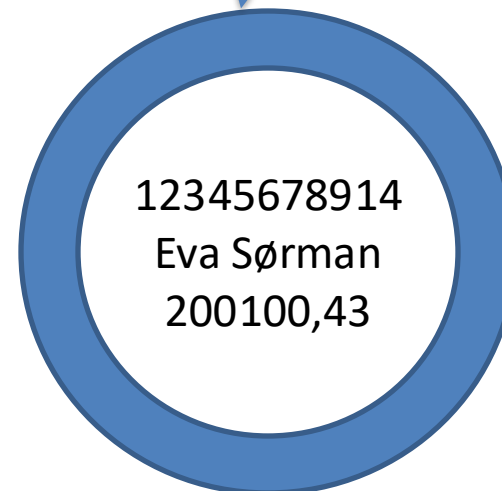
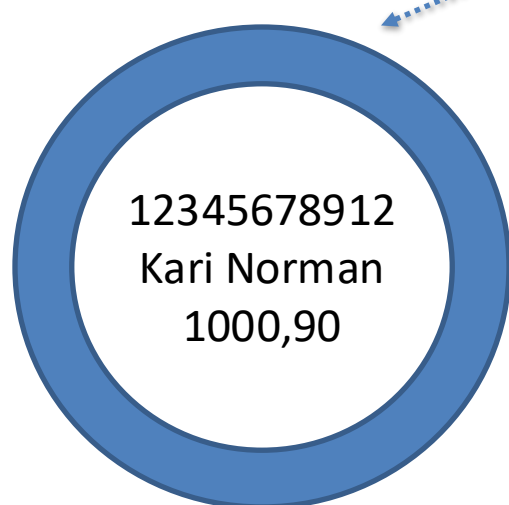


# Klasser i Bank ATM System?

# Objekter og klasser

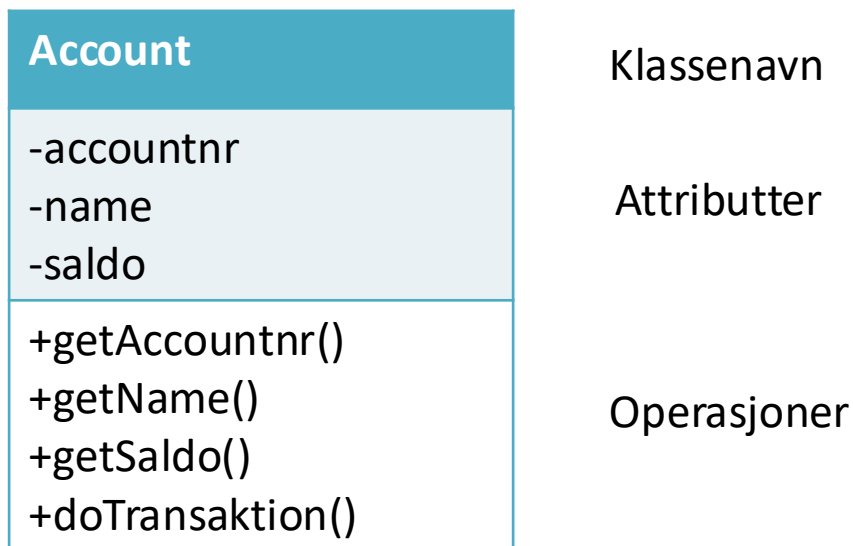
- Ulike kontoobjekter
  - Samme oppbygging
  - Forskjellig datainnhold

Account



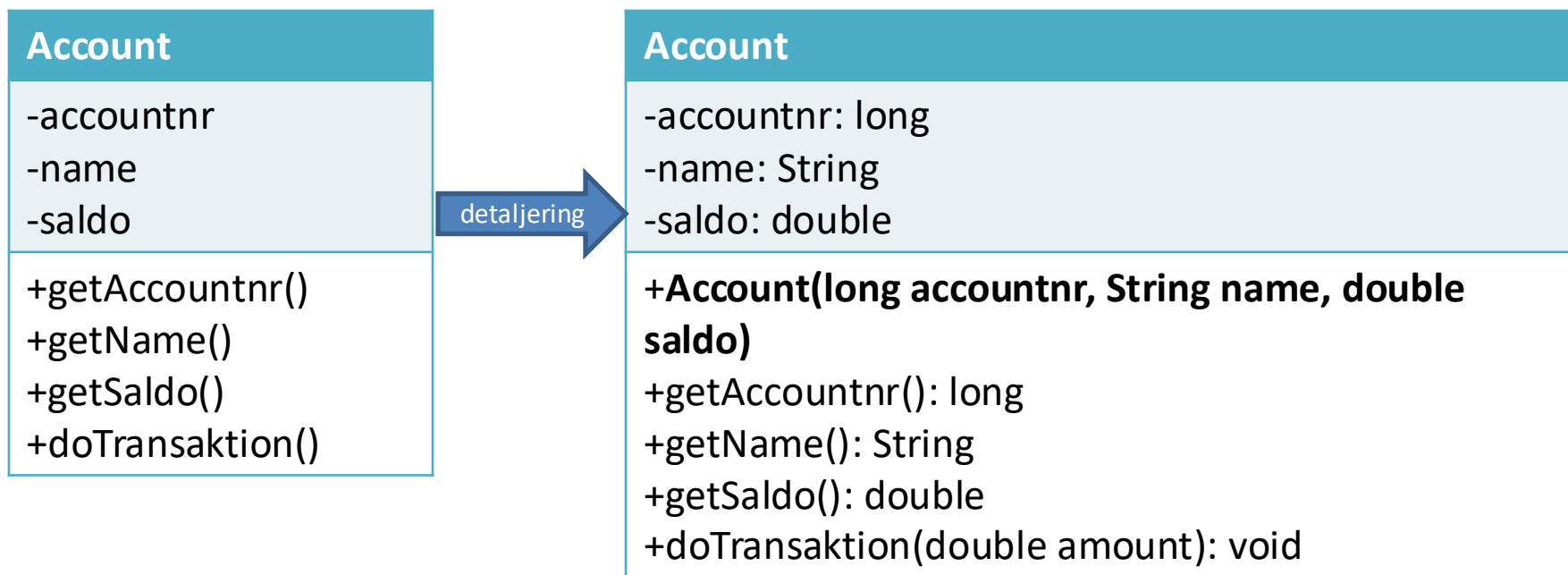
# Klassediagram

- En del av UML (unified modelling language)
- Ett skritt på veien i overgang fra tekstlig problembeskrivelse til programkode

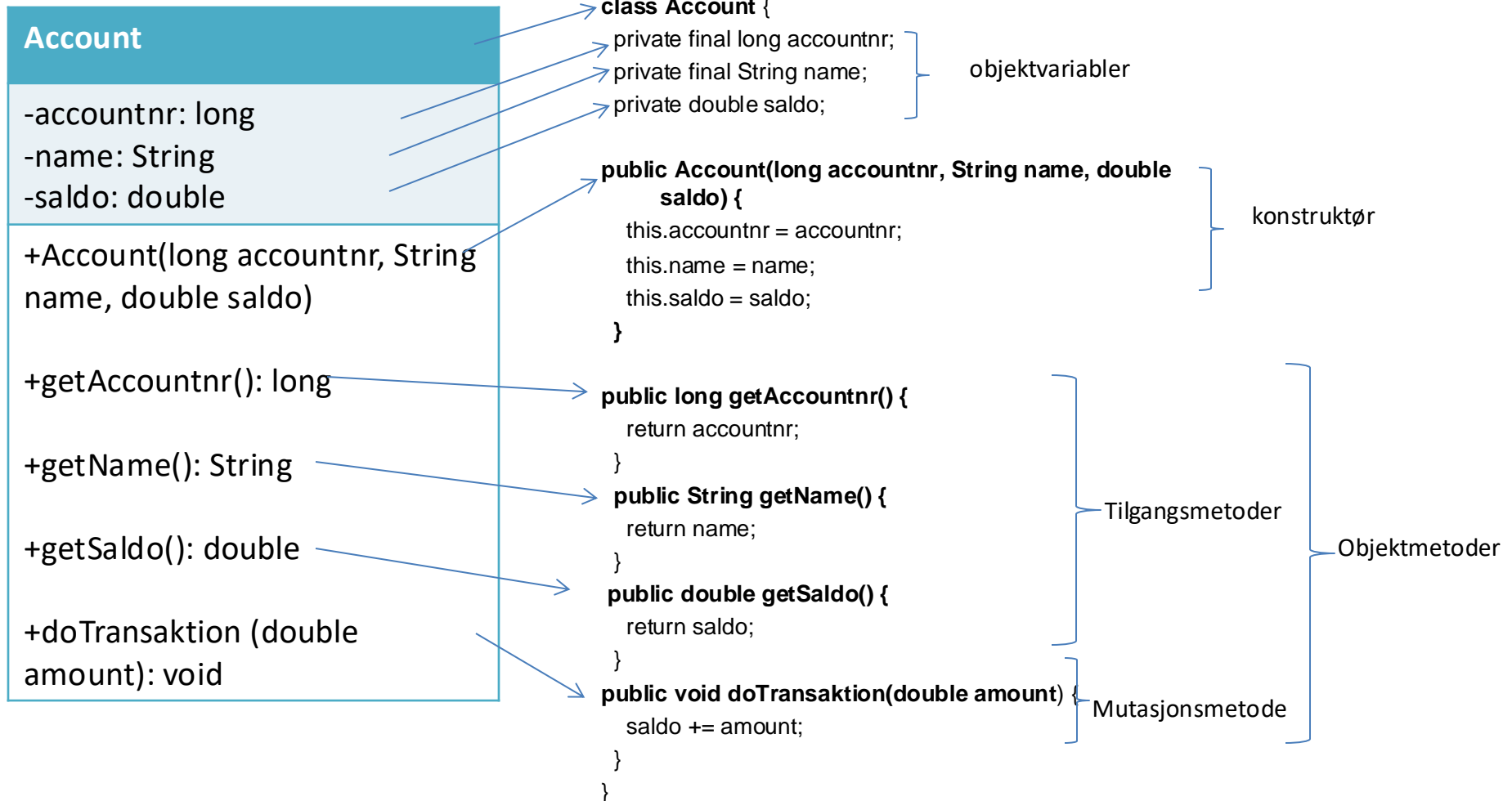


# Klassediagram - detaljering

- Detaljering av klassen Account iht. bruken av klassen (forrige foil)
  - For å lette overgangen til programkode (Java)



# Klassediagram – overgang til Java-kode



# Forstå - begreper

- Metodehode
  - `public double getSaldo()`
- Metodekropp / innhold
  - `return saldo;`
- Klassekropp
  - klasseblokka
- Mutabel/ Immutabel
  - objektet kan endres (eller ikke) etter at det er opprettet!

# Forstå - Tilgangsmodifikatorene

## private/private/protected

Tilgangsmodifikatorer styrer tilgjengeligheten til klasser, konstruktører og medlemmer (medlemmer = variabler + metoder).

- **Private (-) :**
  - kun tilgjengelig innenfra klassen, for eksempel fra alle metoder i klassen.
- **Public (+) :**
  - tilgjengelig fra overalt, for eksempel fra et klientprogram.
- ingen tilgangsmodifikator/ package
  - pakketilgang, kun tilgjengelig innenfor den pakken som klassen tilhører.

Privat tilgang

Pakketilgang (default)

```
class Account {  
    private long accountnr;  
    private String name;  
    private double saldo;
```

Offentlig tilgang

```
    public Account(long startAccountnr,  
                   String startName, double startSaldo)  
    {  
        accountnr = startAccountnr;  
        name = startName;  
        saldo = startSaldo;  
    }  
    .....  
}
```

Dersom tilgangen til klassen er strengere enn tilgangen til en konstruktør /medlem, vil klassetilgangen gjelde for konstruktøren/medlemmet. Ellers gjelder den tilgangen som konstruktøren/medlemmet har.

# Forstå - Return setningen

- ▶ Dersom returtypen er forskjellig fra void, må metoden inneholde minst én return-setning.
- ▶ En return-setning består av det reserverte ordet **return** etterfulgt av et uttrykk med samme type som returtypen, eller av en type som automatisk kan omformes til returtypen
- ▶ Dersom programkontrollen treffer på en return-setning, hopper den umiddelbart ut av metoden uavhengig av om det finnes flere setninger i metoden.
- ▶ En metode med returtype void kan også inneholde setningen:

*return;*

effekten er den samme som dersom en verdi returneres.



# Forstå - Skjuling av navn; this

- ▶ Hvorfor fungerer ikke følgende metode som forventet? Hva er feil?

```
public Account(long accountnr, String name, double saldo) {  
    accountnr = accountnr;  
    name = name;  
    saldo = saldo;  
}
```

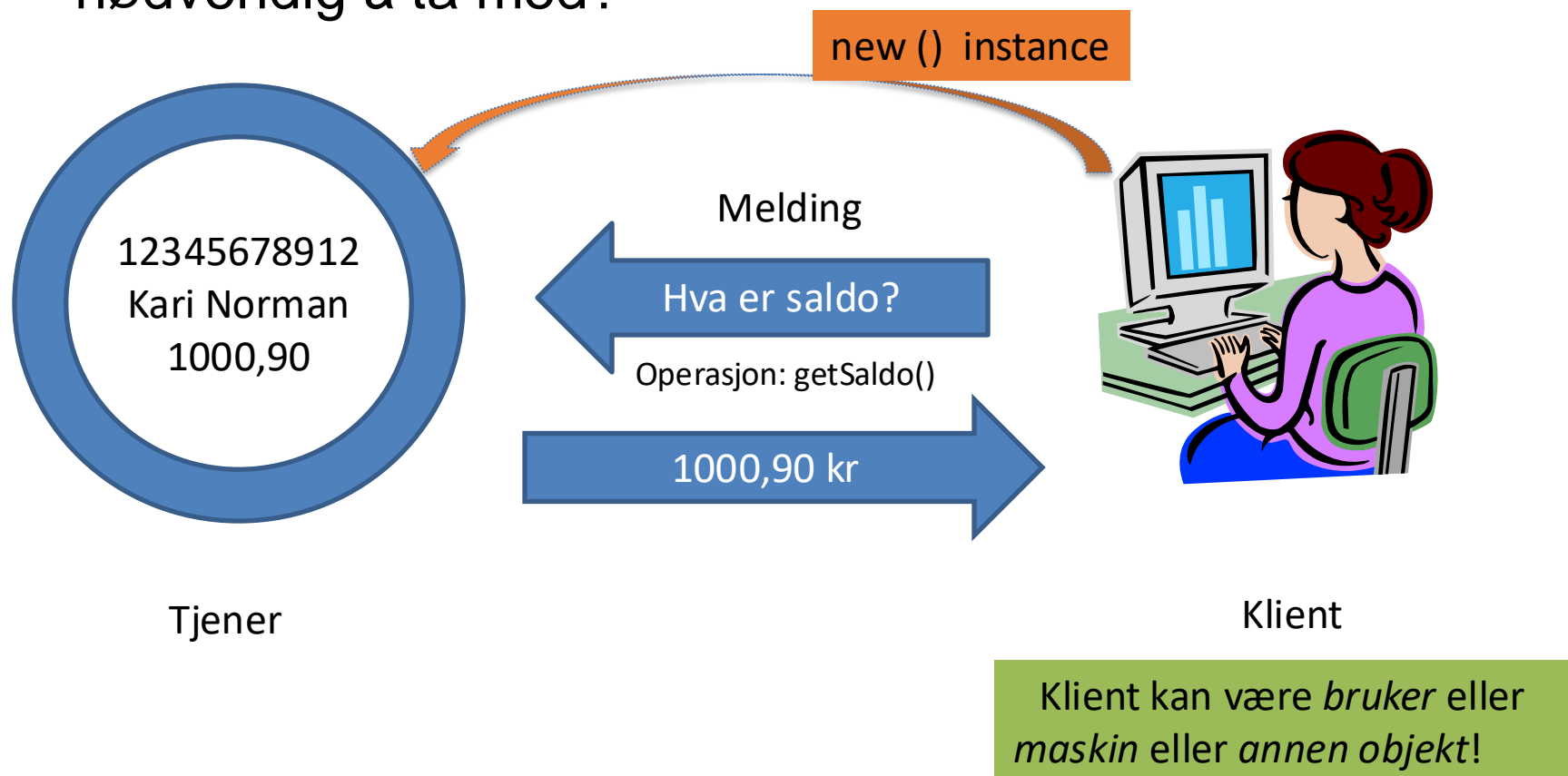
- ▶ Parameternavnene *skjuler* objektvariablene med samme navn.
- ▶ Vi må passe på at vi ikke gir samme navn til parametre og objektvariabler.
- ▶ Dersom vi absolutt ønsker å bruke samme navn, kan vi skille navnene fra hverandre ved å bruke det reserverte ordet *this*:

```
public Account(long accountnr, String name, double saldo) {  
    this.accountnr = accountnr;  
    this.name = name;  
    this.saldo = saldo;  
}
```

- ▶ *this* er en referanse som alltid peker til det objektet Java-tolkeren holder på med i øyeblikket.

# Forstå - Constructor, bruke av klasser

- Viktig å vite hva vi skal bruke klassen til – hva er nødvendig å ta med?



# Eks. på bruk av klassen Account (klient)

```
class AccountTest {  
    public static void main(String[] args) {  
        /* Oppretter et objekt av klassen Account. */  
        Account olesAccount = new Account(123456676756L, "Ole Olsen", 2300.50);  
  
        /* Setter inn 1000 kroner */  
        olesAccount.doTransaksjon(1000.0);  
  
        /* Spør objektet om den saldo */  
        double saldo = olesAccount.getSaldo();  
  
        /* Skriver ut nysaldoen */  
        System.out.println(" The new saldo is " + saldo);  
    }  
}
```

# Forstå - Metoden toString()

- Vi kan skrive ut innholdet i et Account-objekt slik (i klientprogrammet):

```
long accountnr = olesAccount.getAccountnr();
```

```
String name = olesAccount.getName();
```

```
double saldo = olesAccount.getSaldo();
```

```
System.out.println("Kontonr: " + accountnr + ", name: " + name + ", saldo:  
" + saldo);
```

- eller slik:

```
System.out.println("Før endring av dataene: \nAccountnr: " +  
olesAccount.getAccountnr() + ", name: " + olesAccount.getName() +  
", saldo: " + olesAccount.getSaldo());
```

- Dette er noe vi trenger ofte....*se neste side!*

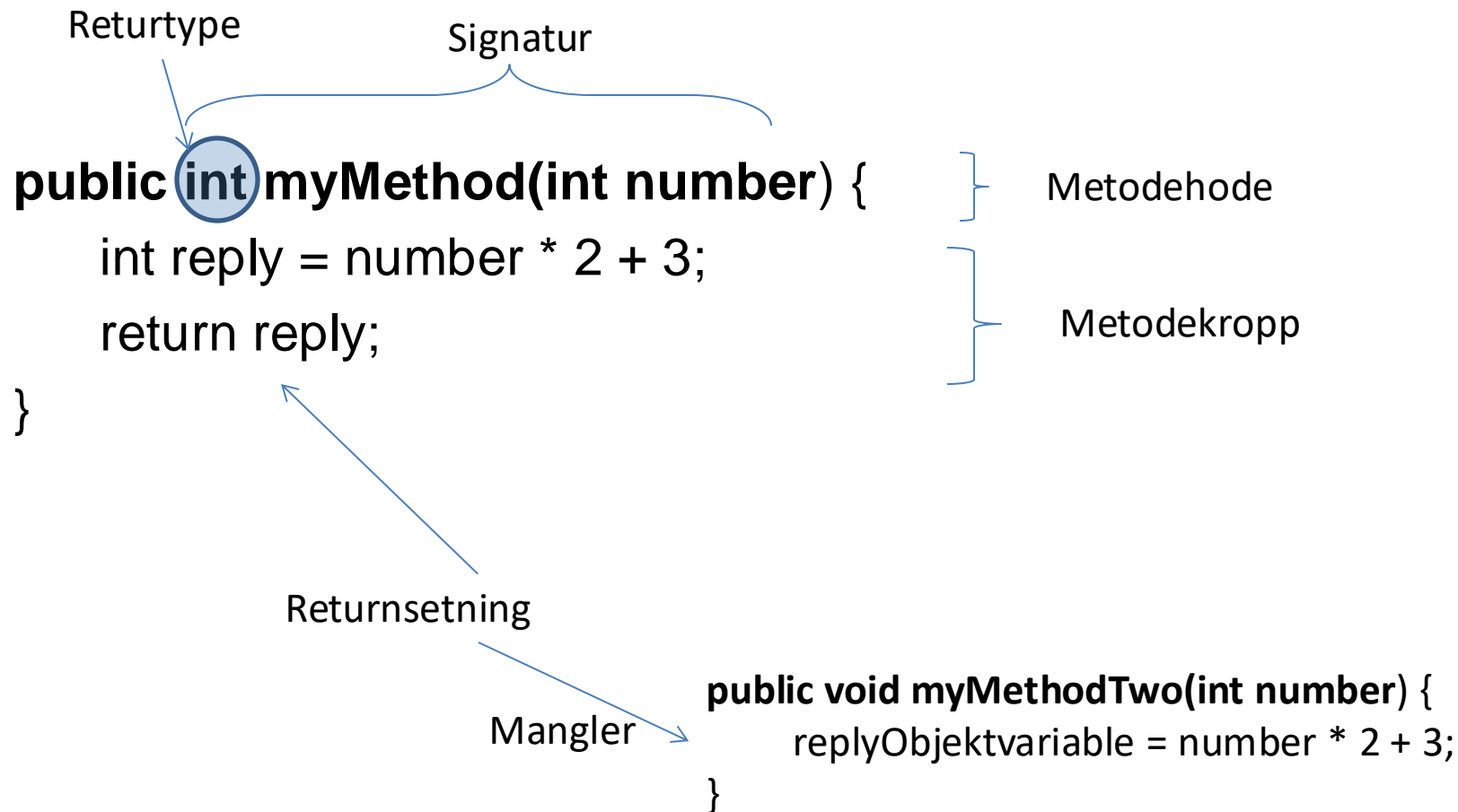
# Metoden toString()

- ▶ **Hvorfor ikke la *Klasse* selv lage den strengen som skal skrives ut?**

```
public String toString() {  
    return "Kontonr.: " + accountnr + "\nNavn: " + name + "\nSaldo: " + saldo;  
}
```

- ▶ Metodehodet er standardisert til *public String toString()*
- ▶ Utskriftsetningen fra forrige side kan nå se slik ut:  
`System.out.println(olesAccount.toString());`
- ▶ `toString()` er underforstått:
  - I `println()`-metoden:
    - `System.out.println(olesAccount);` // som om det hadde stått `olesAccount.toString()`
  - Ved skjøting av strenger:
    - `String result = "Dette er kontoen: " + olesAccount;`
- ▶ Standardutgaven av metoden `toString()` blir brukt dersom klassen ikke har sin egen utgave.
- ▶ Det er en god vane å lage `toString()`-metoder i alle klasser.

# Forstå - Navn overloading, en metode



# Navn overloading, en metode

- ▶ Flere metoder kan ha samme navn dersom de har forskjellig signatur

- *Signaturen* til en metode = metodenavnet, antall og type parametre.  
**(Returtypen og parameternavnene er ikke en del av signaturen)**

- ▶ Eksempel fra klassen String

- `int indexOf(int character)`
  - `int indexOf(int character, int fromIndeks)`
  - `int indexOf(String str)`
  - `int indexOf(String str, int fromIndeks)`

- ▶ Eksempel på bruk av disse metodene

```
String tekst = "Det er 9.august i dag. Om to uker er skolen i full gang.";
int pos1 = tekst.indexOf('.');           // pos1 blir 8
int pos2 = tekst.indexOf('9');           // pos2 blir 7
int pos3 = tekst.indexOf('.', pos1 + 1); // pos3 blir 21
int pos4 = tekst.indexOf("Om");          // pos4 blir 23
int pos5 = tekst.indexOf("Om", pos4 + 1); // pos5 blir -1 (=finnes ikke)
```

# Forstå - Navn overloading, en Konstruktører

- Også konstruktørnavnet kan overloades. En klasse kan ha flere konstruktører dersom de har forskjellig signatur.
- Dersom vi ikke lager noen konstruktører selv, lages det automatisk en *standardkonstruktør* (tom parameterliste/kropp).
- Dersom vi lager egne konstruktører, og vi ønsker at en konstruktør med tom parameterliste skal eksistere, må vi lage den.



# Eksempler på konstruktører for klassen Konto

```
public Account(long accountnr, String name, double saldo) {  
    this.accountnr = accountnr;  
    this.name = name;  
    this.saldo = saldo;  
}
```

```
public Account(long accountnr, String name) {  
    this.accountnr = accountnr;  
    this.name = name;  
    this.saldo = 0;  
}
```

```
public Account() {  
}
```

# Konstruktører og initiering av objektvariabler

- ▶ Vi kan initiere objektvariabler samtidig med deklarasjonen:
  - `private String name = "";`
  - `private double length = 5;`
- ▶ Dersom vi gir verdi til disse variablene i en konstruktør, vil verdiene i konstruktøren overstyre de initierte verdiene.
- ▶ En konstruktør kan, men må ikke, gi verdier til alle objektvariabler.

# Programmeringskonv. for klasser

***Dette er retningslinjer, ikke syntaksregler.***

- ▶ Klassenavn har stor forbokstav.
- ▶ Navn på variabler og metoder har liten forbokstav.
- ▶ Dersom navnet består av flere ord, benyttes stor forbokstav fra og med ord nummer to. Understrekingstegn og \$ benyttes ikke.
- ▶ Objektvariabler har tilgangsmodifikator *private*.
- ▶ Konstruktører har tilgangsmodifikator *public*.
- ▶ Metoder har tilgangsmodifikator *public* eller *private*.
- ▶ Klassevariabler og klassemetoder brukes sjelden.
- ▶ Tilgangs- og mutasjonsmetoder som har som oppgave å hente ut eller forandre på et attributt, har standardiserte navn på engelsk. Med tilpasning til norsk får vi følgende, vist ved eksempler:

```
public void setLength(double startLength)
```

```
public double getLength()
```

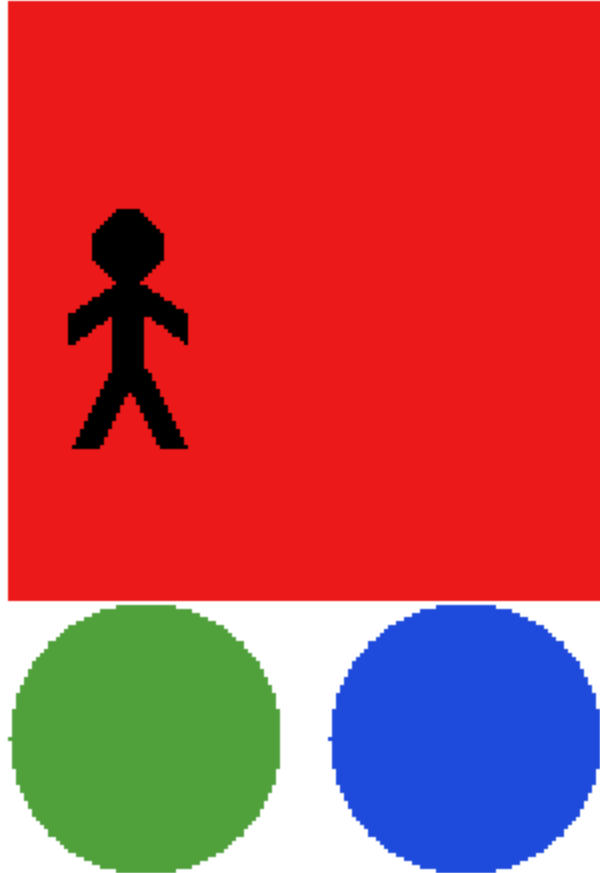
```
public boolean isFinished()
```

```
public void setFinished(boolean startFinished)
```

# Kahoot!

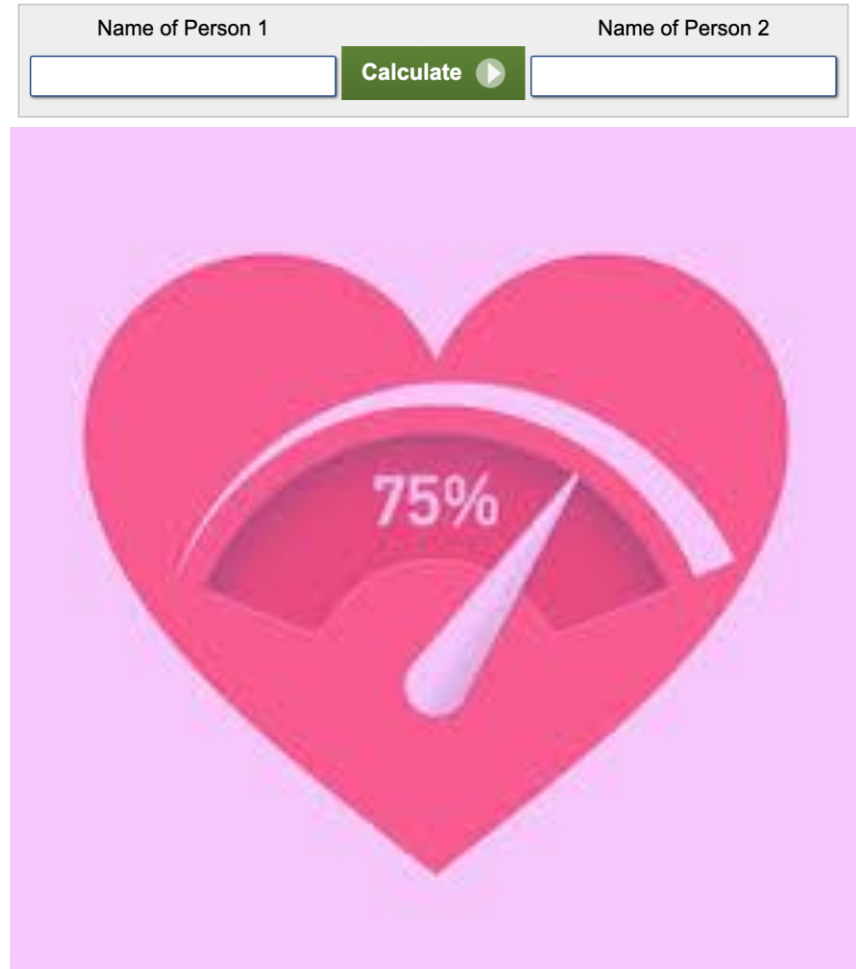
# Thank you!

# Oppgave - Tog



# Oppgave – Love Calculator

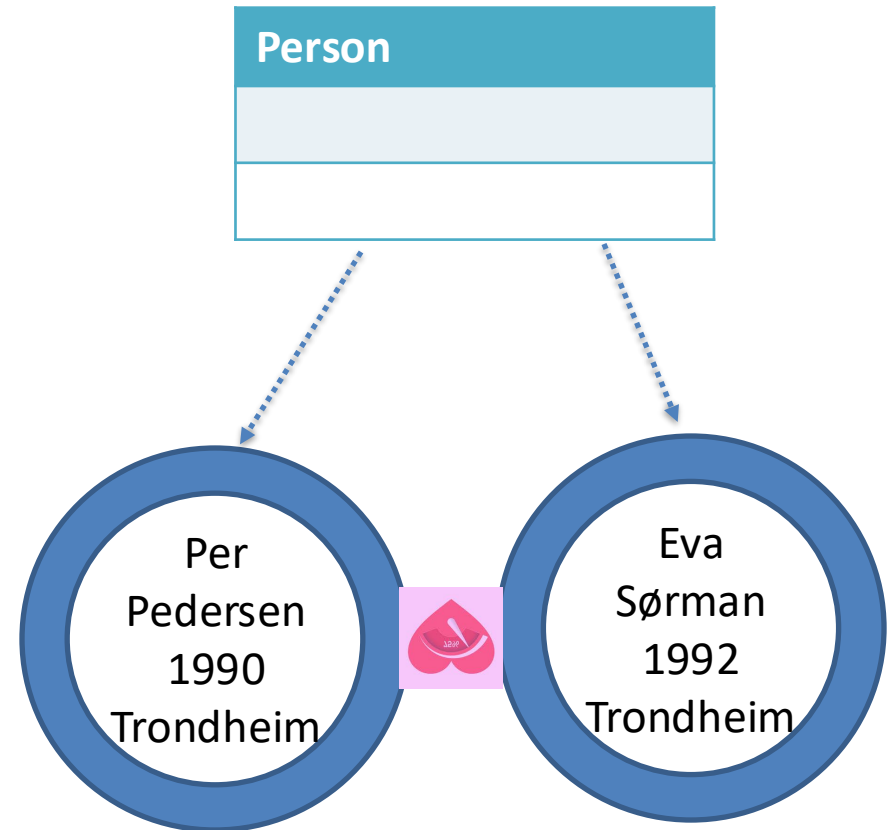
- Oppgave:
  - Lag **Person** klasse med objektvariabler, konstruktør og Objektmetoder
  - Finn forklaring på begrepene (basert på klassesdiagram!)
    - Private
    - Public



Picture from <https://www.lovecalculator.club/>

# Oppgave1 – Love Calculator (part 1)

- Person klasse:
  - Objektvariabler,
    - firstName
    - lastName
    - yearOfBirth
    - address
  - Konstruktør
  - Objektmetoder
    - getAge(): int
    - getFirstName(): String
    - getLastName(): String
    - getYearOfBirth(): String
    - getAddress(): String





# Oppgave2 – LoveCalculator (part 2)

- Lag klient for Person klasse:
  - LoveCalculator klasse
  - Objektmetoder
    - calculateLove(Person person1, Person person2): int
- Bruk LoveCalculator klasse fra Main klasse

LoveCalculator
+ calculateLove(Person person1, Person person2): int

# Ekstra Oppgave

- Lag en metode inn i Person klasse, slik
  - `int calculateLove (Person anotherPerson)`
- Lag en ny metode i LoveCalculator som har bare en parameter
  - `int calculateLove(Person person)`