Kapittel 6:
Arv

Redigert av:

Khalid Azim Mughal (khalid@ii.uib.no)

Kilde:

Java som første programmeringsspråk (3. utgave)

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen
Cappelen Akademisk Forlag, 2006.
ISBN: 82-02-24554-0
http://www.ii.uib.no/~khalid/jfps3u/
(NB! Boken dekker opptil Java 6, men notatene er oppdatert til Java 7.)

Emneoversikt

Sentrale begreper:
- Superklasse og subklasse
- Generalisering - spesialisering
- Arv av medlemmer
- Arvhierarki

Utvidelse av egenskaper og atferd
- Initialisering vha. `super()` og `this()`
- Klienters bruk av subklassen

Bruk av arvede medlemmer
- I subklassen og i klienter
- Instans- og statiske medlemmer

Superklassereferanser og subklasse-objekter
- Forskjell på super- og subklassereferanser

Overkjøring av instansmetoder
- Bruk av overkjørt instansmetode
- `super` nøkkelord
- Aksess av overskygget felt

Endelige klasser og medlemmer:
- `final` nøkkelord
- Endelige klasser
- Endelige metoder og variabler

Konvertering av referanser
- Implisitt (automatisk)
- Konverteringsoperator (*cast*)
- `instanceof` operator
- Superklasse-subklasse relasjon (*er-en* relasjon): `sub` *er-en* `sup`

Eksempel: Brus-automater

Arv: motivasjon og sentrale begreper

- En nyttig mekanisme for *gjenbruk av kode*.
- Utledning av nye klasser fra en gammel klasse.
- Den *deriverte/utvidede* (sub)klassen arver medlemmer fra *basis* (super)klassen.
  - Alle medlemmer som er *tilgjengelige* utenfor superklassen arves.
  - Arvede medlemmer kan brukes som om de var definert i subklassen selv.
  - Denne forelesningen vil kun deklarere tilgjengelige medlemmer (som arves).
- Arv brukes for *utvidelse* av en klasse (*spesialisering*):
  - Vi kan tilføye nye metoder, dvs. ny *atferd*.
  - Vi kan tilføye nye (instans)variabler, dvs. nye *egenskaper*.
  - Dessuten kan atferd fra superklassen endres ved å *overkjøre* en *superklassemetode*.
- En subklasse har én superklasse, også kalt *foreldreklasse/basisklasse*.
- En subklasse kan også være forelder til andre subklasser, osv. Denne relasjonen er *transitiv*.
  - Slik kan klasser ordnes i et *arvhierarki*, med to eller flere nivåer.
- Klasse `Object` er (implisitt) superklasse til alle klasser, og er roten til arvhierarkiet.
  – Alle klasser, og dermed alle objekter, arver felles oppførsel fra `Object`-klassen.

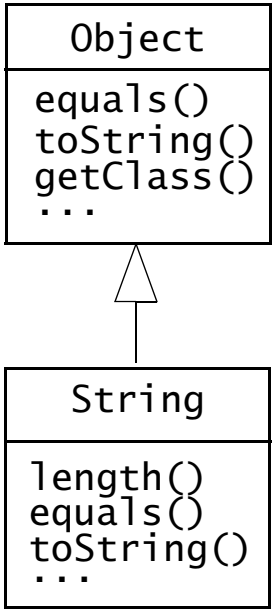
Arv: viktige begreper

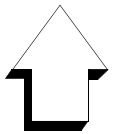
Superklasse

Superklasse/basisklasse: mest generelle klasse

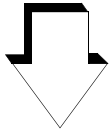
Subklasse

Subklasse/derivert klasse: mest spesialiserte klasse
- arver egenskaper og oppførsel til superklassen
- kan definere nye egenskaper
- kan endre og/eller utvide oppførsel





Generalisering



Spesialisering

Generalisering fører til mer abstrakte klasser.

Spesialisering fører til mer skreddersydde klasser.

Arv brukes til å implementere superklasse-subklasse ("is-a") klasserelasjon.

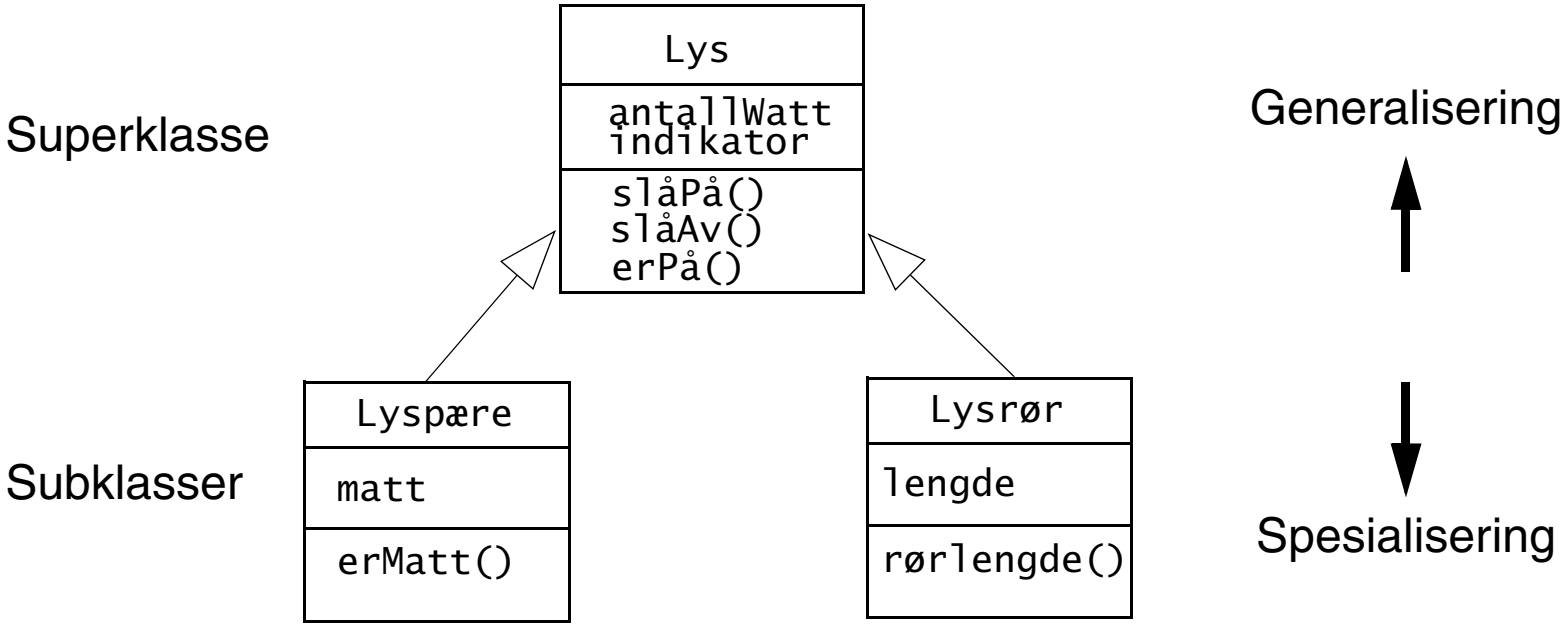
Et første eksempel på bruk av arv

```
public class OOPKlient {
  public static void main(String[] args) {
    String str = new String("aha");          // String utvider Object

    System.out.println(str.getClass());      // arvet fra Object klasse

    System.out.println(str.length());        // definert i String klasse

    System.out.println(str.equals("hihaw")); // redefinert (overkjørt) i String klasse
  }
}
```

- *Arv av medlemmer fra superklassen*: subklassen `String` arver metoden `getClass()` fra superklassen `Object` (men dette er likegyldig for klassen `OOPKlient`, som kan bruke `getClass()`-metoden som om den var definert i `String`-klassen).
- *Utvidelse av subklassen*: subklassen `String` har en ny metode `length()` (som superklassen `Object` ikke har).
- *Metodeoverkjøring*: subklassen `String` redefinerer metoden `equals()` (mer om dette senere i forelesningen).

Utvidelse av egenskaper og atferd

- Superklassen `Lys` danner grunnlag for mer spesialiserte subklasser.
- *Tilgjengelige* medlemmer arves i subklassen.
- Unntak: Konstruktører kan *ikke* arves.



Superklassen: Lys

```
public class Lys {
    int antallWatt;    // lysstyrken
    boolean indikator; // av == false, på == true
    String lokasjon;   // hvor lyset er plassert
    // ...
    Lys(int antallWatt, boolean indikator, String lokasjon) {
        this.antallWatt = antallWatt;
        this.indikator  = indikator;
        this.lokasjon   = lokasjon;
    }
    public void slåPå() { indikator = true; }
    public void slåAv() { indikator = false; }
    public boolean erPå() { return indikator; }

    @Override
    public String toString() {
        return "[Lys: " + antallWatt + " Watt, " + "indikator "
               + indikator + ", plassering: " + lokasjon + "]";
    }
}
```

Utvidelse av egenskaper og atferd (forts.)

- Subklassen `Lyspære` utvider superklassen `Lys`, og er mer spesialisert.

```
public class Lyspære extends Lys {
    boolean matt;      // true == matt, false == klar

    Lyspære(int antallWatt, boolean indikator, String lokasjon, boolean matt) {
        super(antallWatt, indikator, lokasjon);
        this.matt = matt;
    }
    @Override public String toString() {
        return "[Lyspære: " + antallWatt + " Watt, " + "indikator "
                + indikator + ", plassering: " + lokasjon + ", matt: " + matt + "]";
    }
    public boolean erMatt() { return matt; }
}
```

- Hver lyspære *er-et* lys.
- Nøkkelordet `extends` markerer superklasse-subklasserelasjonen.
- Subklassens konstruktør bruker `super()` til å initialisere arvede felter.
- Metoden `toString()` endrer atferden ved å overkjøre superklassens metode.
- Ny metode i subklassen, `erMatt()`, utvider atferden.
- Subklassen `Lyspære` har derfor mer funksjonalitet enn superklassen `Lys`.

Superklasse-konstruktører

- Subklasse-konstruktør kaller superklasse-konstruktør for å initialisere arvede instansvariabler.
- Dersom en subklasse-konstruktør *ikke* kaller en superklasse-konstruktør eksplisitt, blir den *implisitte standard superklasse-konstruktøren*, `super()`, kalt automatisk.

```
class Lyspære extends Lys {
    //...
    Lyspære() { }
    //...
}
```

Den standardkonstruktøren i klassen `Lyspære` er ekvivalent med

```
Lyspære(){ super(); }
```

- Dersom en superklasse har *kun ikke-standard* konstruktører spesifisert, vil kompilatoren gi en feilmelding dersom *standard superklasse-konstruktør* blir kalt eksplisitt fra subklassen.
- Kall til superklasse-konstruktøren er den første setningen i subklasse-konstruktøren.
- Tips: KALL ALLTID SUPERKLASSE-KONSTRUKTØR EKSPLISITT!

Bruk av `this()` og `super()` i konstruktører

- Konstruktører kan forenkles: `this()` kaller andre konstruktører i samme klasse.

```
class Lys {
    int antallWatt;      // lysstyrken
    boolean indikator;   // av == false, på == true
    String lokasjon;     // hvor lys er plassert

    Lys() {                          // 1. (eksplisitt) standardkonstruktør
        this(0,false,"X");           // 3. ikke-standardkonstruktør blir utført
    }
    Lys(int watt, boolean ind) {     // 2. ikke-standardkonstruktør
        this(watt,ind,"X");          // 3. ikke-standardkonstruktør blir utført
    }
    Lys(int watt, boolean ind, String lok) {  // 3. ikke-standardkonstruktør
        super(); // fører til at standard superklasse-konstruktør blir utført.
        antallWatt = watt;
        indikator = ind;
        lokasjon = new String(lok);
    }
    // ...
}
```

Klienters bruk av subklasse-objekter

- Klienter trenger ikke bry seg om hvorvidt et medlem er deklarert i subklassen eller arvet fra dennes superklasse.

```
public class Belysning {

    public static void main(String[] args) {
        // Opprett en lyspære.
        Lyspære pære = new Lyspære(40, false, "stuen", true);

        pære.slåPå(); // arvet metode
        System.out.println("Lyspæren i stuen er på " +
                           pære.antallWatt + " Watt"); // arvet feltvariabel

        if (pære.erMatt())                             // subklassens metode
            System.out.println("Pæren er matt.");
        else
            System.out.println("Pæren er klar.");

        if (pære.erPå())                               // arvet metode
            System.out.println("Stuelyset er på.");
        else
            System.out.println("Stuelyset er av.");
```

```
        System.out.println(pære);

        // Opprett et vanlig lys.
        Lys utelys = new Lys(75, true, "utenfor døren");

        utelys.slåAv();
        System.out.println("Utelyset er på " + utelys.antallWatt + " Watt");

        if (utelys.erPå())
            System.out.println("Utelyset er på.");
        else
            System.out.println("Utelyset er av.");

        System.out.println(utelys);

    }
}
```

- Utskrift fra kjøring:

```
Lyspæren i stuen er på 40 Watt
Pæren er matt.
Stuelyset er på.
[Lyspære: 40 Watt, indikator true, plassering: stuen, matt: true]
Utelyset er på 75 Watt
Utelyset er av.
[Lys: 75 Watt, indikator false, plassering: utenfor døren]
```

Bruk av arvede medlemmer

```
public class Rektangel {
    double bredde;
    double lengde;

    Rektangel(double bredde, double lengde) {
        this.bredde =  bredde;
        this.lengde = lengde;
    }
    public double beregnAreal() { return bredde*lengde; }
    @Override public String toString() {
        return "[" + getClass() + ": bredde=" + bredde +
               ", lengde=" + lengde + "]";
    }
}
```

- Klassen `Rektangel` har bare tilgjengelige medlemmer.
- Disse arves i en subklasse og kan dermed refereres til ved navnet i subklassen.
- Klassen `Rektangel` er selv en subklasse (til `Object`) og kan kalle den arvede metoden `getClass()` direkte ved metodenavnet.
- Enhver klasse som ikke eksplisitt angir at den utvider en annen klasse, er (implisitt) en subklasse til `Object`-klassen.

Bruk av arvede medlemmer innad i subklasser

- En subklasse kan referere *direkte* til alle arvede medlemmer vha. *medlemsnavn*:

```
public class Kube extends Rektangel {
    double høyde;

    Kube(double bredde, double lengde, double høyde) {
        super(bredde, lengde);  // bruker superklassens konstruktør
        this.høyde = høyde;
    }
    public double beregnOverflate() {
        return 2 * ((bredde*lengde)       // aksess vha. feltnavn
                   +(this.bredde*høyde)   // aksess vha. this-referansen
                   +(this.lengde*høyde));
    }
    public double beregnVolum() {
        return beregnAreal() * høyde;     // aksess vha. metodenavn
    }
    @Override public String toString() {
        return "[" + getClass()           // aksess vha. metodenavn
               + ": bredde=" + bredde + ", lengde=" + lengde
               + ", høyde=" + høyde + "]";
    }
}
```

Bruk av arvede medlemmer i en klient

```
public class VolumRegning {
    public static void main(String[] args) {
        Kube klosse = new Kube(3, 4, 5);
        System.out.println("Areal av grunnflaten: " + klosse.beregnAreal());
        System.out.println("Overflate av alle sidene: " + klosse.beregnOverflate());
        System.out.println("Volumet av klossen: " + klosse.beregnVolum());
        System.out.println("Klosse: " + klosse);
    }
}
```

- Ingen forskjell for klienten om medlemmet er arvet (som `beregnAreal()`) eller deklarert i subklassen selv (f.eks. `beregnVolum()`).
- Klienten kunne også skrevet ut f.eks. bredden direkte ved `klosse.bredde`.
- Utskrift fra kjøring:

```
Areal av grunnflaten: 12.0
Overflate av alle sidene: 94.0
Volumet av klossen: 60.0
Klosse: [class Kube: bredde=3.0, lengde=4.0, høyde=5.0]
```

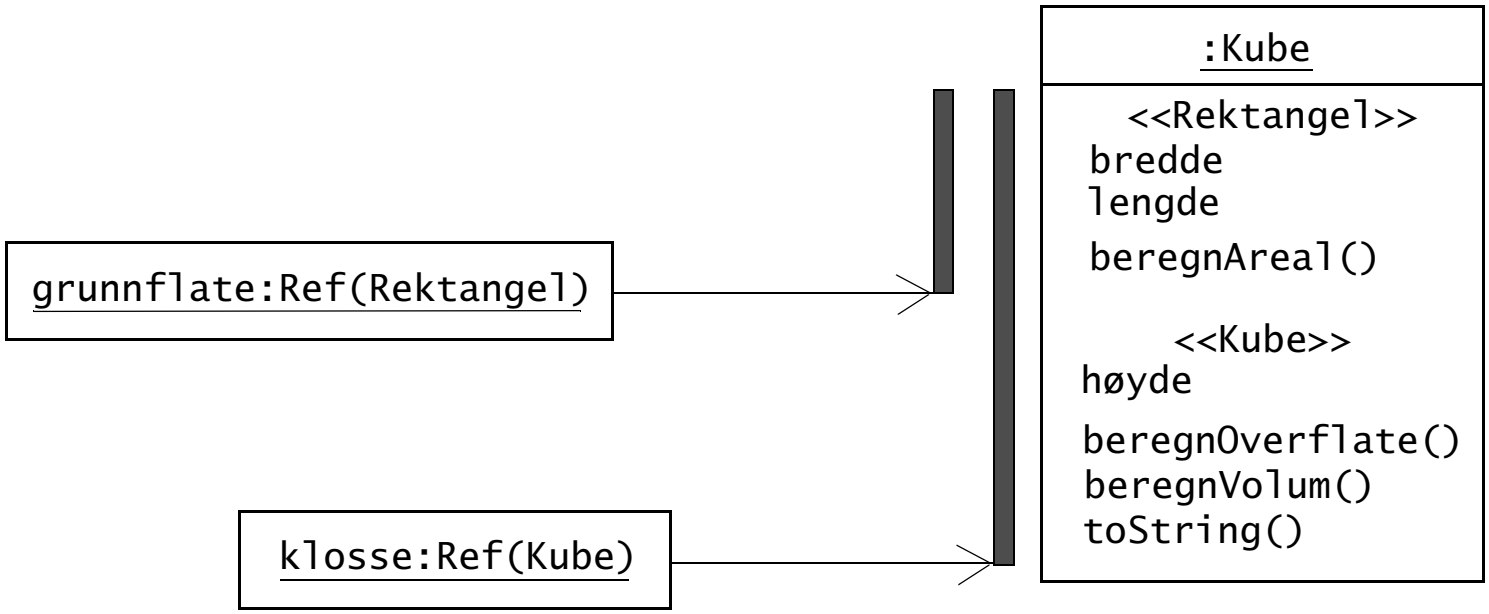
Bruk av arvede medlemmer (forts.)

- *Statiske medlemmer* kan aksesseres gjennom *klassenavn* (superklasse eller subklasse) eller via en *referanse til et subklasseobjekt*:

```
class AnsattV0 { // Fra Program 6.1
    ...
class ArbeidsLederV0 extends AnsattV0 { // Fra Program 6.2
   ...
class BrukArv { // Fra Program 6.4
    public static void main(String[] args) {
        // Oppretter et leder-objekt
        ArbeidsLederV0 enLeder = new ArbeidsLederV0("Ole","Olsen",325.0);
        ...
        System.out.printf("Normal arbeidsuke er %.1f timer%n",
                          enLeder.NORMAL_ARBEIDSUKE);                  // (1)
        if (50.5 > ArbeidsLederV0.NORMAL_ARBEIDSUKE)                   // (2)
            System.out.printf("Lederen jobbet %.1f timer overtid%n",
                              (50.5-AnsattV0.NORMAL_ARBEIDSUKE));      // (3)
        else
            System.out.println("Lederen jobbet ikke overtid denne uken");
    }
}
```

Superklassereferanser og subklasseobjekter

- En superklassereferanse kan referere til et objekt av en subklasse - men denne referansen kan kun brukes til å referere til *arvede* medlemmer.




```
Kube klosse = new Kube(3, 4, 5);
Rektangel grunnflate = klosse;  // aliaser
System.out.println("Bredde: " + grunnflate.bredde);  // ok
grunnflate.beregnAreal();         // ok
grunnflate.beregnOverflate();     // Ikke tillatt - metoden er deklarert i subklassen
```

Overkjøring av instansmetoder

- Overkjøring betyr redefinering av en *instansmetode* fra superklassen.
- Metoden må ha *samme signatur* og *samme returtype* i både superklassen og subklassen.
- Metodedeklarasjonen i subklassen *overkjører* metodedeklarasjonen i superklassen.
- Adgang fra subklassen til den overkjørte metodedeklarasjonen i superklassen er via nøkkelordet `super`, som vist i linje (5) under.

```
class AnsattV0 { // Fra Program 6.1
    ...
    double beregnUkelønn(double antallTimer) {
        ...
    }
    ...
}
class ArbeidsLederV2 extends AnsattV0 { // Fra Program 6.8
    ...
    @Override  // Bruk denne annotasjonen foran den overkjørte metoden.
    double beregnUkelønn(double antallTimer) {                          // (4)
        ...
        double fastUkelønn = super.beregnUkelønn(antallTimer);          // (5)
        ...
    }
}
```

Overskygging av feltvariabler

- Deklarasjon av et felt i subklassen med samme navn som i superklassen *overskygger* feltet i superklassen, slik at vi må bruke `super` for å få fatt i superklassens felt.

```
public class Sparepære extends Lys {
    int antallWatt;      // overskygger felt fra superklassen
    Sparepære(int antallWatt, boolean indikator, String lokasjon, boolean matt) {
        super(0, indikator, lokasjon);
        this.antallWatt = antallWatt;  // subklassens felt
    }
    @Override public String toString() {
        return "[Sparepære: " + antallWatt + " Watt, tilsvarer " + super.antallWatt
          + " Watt, " + "indikator " + indikator + ", plassering: " + lokasjon + "]";
    }
    public double beregnEffekt() {
        switch(antallWatt) {  // subklassens felt
            case  7: super.antallWatt = 40; break; // superklassens felt
            case 11: super.antallWatt = 60; break;
            case 15: super.antallWatt = 75; break;
            default: super.antallWatt =  0; break;
        }
        return super.antallWatt; // superklassens felt
    }
}
```

Endelige klasser og medlemmer (`final`)

- Endelige klasser (`final` *classes*)
  - klasser som ikke kan spesialiseres, dvs. som ikke kan utvides.

```
public final class Finito { ... }

public final class String { ... } // i java.lang pakken

public final class EnhetsKube extends Kube {
    EnhetsKube() { super(1, 1, 1); }    // bruker superklassens konstruktør
    public double beregnOverflate() { return 6; } // 2*((1*1)+(1*1)+(1*1))
    public double beregnVolum()     { return 1; } // 1*1*1
}
```

  - merk at det er tilstrekkelig å merke klassen med `final`.
  - ingen av metodene trenger å være endelige.

Endelige klasser og medlemmer (`final`) (forts.)

- Endelige metoder (`final` *methods*)
  - er instansmetoder som ikke kan overkjøres eller statiske metoder som ikke kan overskygges (men kan overlastes).
  - *Kompilatoren kan generere mer effektiv kode.*

```
class Pensjon { public final double beregnPensjon(){...} ... }
class Avgift  { private static final double innkrevAvgift(){...}... }
class Drivhus { public final void startAlarm(){...} ...}
```

- Endelige medlemsvariabler eller *klassekonstanter*
  - Slike medlemsvariabler kan ikke modifiseres etter at de er initialisert.

```
class Pensjon { private final int PENSJONSALDER = 67; ... }

class Avgift  { public static final double MOMS = 23.5; ... }
```

Konvertering av referanser

- Konvertering av referanser fra subklasse til superklasse skjer *implisitt*, f.eks.

```
Lys pære = new Sparepære(7, true, "entreen"); // når Sparepære utvider Lys
```

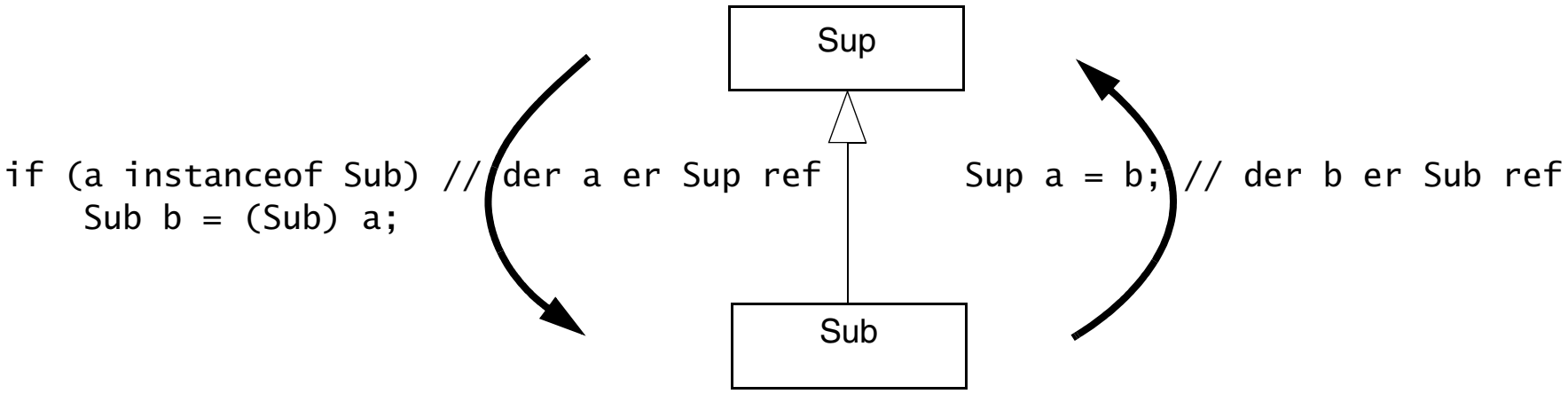
- Konvertering motsatt vei, dvs. fra superklasse til subklasse, må gjøres *eksplisitt* vha. *konverteringsoperator* (**cast**), f.eks.

```
Sparepære nyPære = (Sparepære) pære; // når pære er av referansetype Lys
```

- En slik konvertering kan imidlertid skape problemer.
  - hvordan kan programmet vite at det *er* et subklasseobjekt av riktig type som betegnes av superklassereferansen?
- Løsningen er å bruke operatoren `instanceof` for å sjekke at objektet som betegnes har riktig type:

```
if (pære instanceof Sparepære) {
    Sparepære nyPære = (Sparepære) pære;
    System.out.println("En sparepære på " + antallWatt + " Watt tilsvarer en "
        + " vanlig pære på " + beregnEffekt() + " Watt.");
}
```

Sub *er-en* Sup



Et subklasse-objekt kan oppføre seg som et superklasse-objekt, og kan derfor brukes som om det var et superklasse-objekt. Derfor tillater kompilatoren upcasting:

```
Sup a = b; // der b er Sub ref
```

Derimot er det ingen garanti for at en superklasse-referanse betegner et subklasse-objekt av riktig type. Derfor må vi alltid sjekke vha. `instanceof` at det faktisk er et subklasse-objekt av riktig type før vi foretar downcasting:

```
if (a instanceof Sub) // der a er Sup ref
    Sub b = (Sub) a;
```

Eksempel: Brus-automater

- Super- og subklasse-relasjon
  - `Automat` - en enkel automat som kan selge varer med samme pris
  - `BrusAutomat` - en mer "avansert" automat som håndterer forskjellige priser
- Kall av `super()`
- Metodeoverkjøring
- Overskygget felt
- Utvidet atferd
- Husk å kjøre med `java -ea`
  - Vi bruker `assert`-setninger til å verifisere parameteren til en privat metode.
- Klienter gjøre bruk av super- og subklasse-objekter
  - Klienter kan kalle alle tilgjengelige metoder
  - Hvis en subklasse overkjører en metode i superklassen, kan klienter som bruker subklasse-objekter kun kalle den overkjørte metoden i subklassen.

Eksempel: Brus-automater (forts.)

```
public class Vare {
  String varenavn; // navn på det som kan kjøpes
  int beholdning; // antallet som er ikke solgt

  Vare(String varenavn, int beholdning) {
    if (beholdning < 0) {
      System.out.println("Feil beholdning: " + beholdning +
                         "\nDen må være mer enn 0 varer.\n" +
                         "Satt til 0.");
      beholdning = 0;
    }
    this.varenavn = varenavn;
    this.beholdning = beholdning;
  }

  public String hentVarenavn() {
    return this.varenavn;
  }

  public int hentBeholdning() {
    return this.beholdning;
  }
```

```
  public void settBeholdning(int beholdning) {
    if (beholdning < 0) {
      System.out.println("Feil beholdning: " + beholdning +
                         "\nDen må være mer enn 0 varer.");
      return;
    }
    this.beholdning = beholdning;
  }

  @Override
  public String toString() { // overkjører toString() fra Object-klassen
    return "\nVarenavn: " + this.varenavn + "\tBeholdning: " + this.beholdning;
  }
}
```

```
public class Automat {
  Vare[] varer; // varene som kan kjøpes fra automaten
  double pris; // felles pris for alle varer
  double mottattBeløp; // penger brukeren har lagt på automaten
  double vekslePenger; // penger automaten kan gi igjen som veksel
  double kasse; // penger som ikke kan gis tilbake som veksel
  private Display display; // display-vindu der informasjon kan vises

  Automat(Vare[] varer, double pris, double veksel) {
    this.varer = varer;
    this.pris = pris;
    vekslePenger = veksel;
    mottattBeløp = 0.0;
    kasse = 0.0;
    display = new Display();
  }

  private void giMelding(String melding) {
    display.visMelding(melding);
  }

  public double finnPris(int vareValg) { // NB. Parameteren er overflødig.
    return pris;
  }
```

```
public void mottaPenger(double beløp) {
  mottattBeløp += beløp;
  giMelding("Mottatt: " + mottattBeløp + " kr");
}

private boolean vareFinnes(int vareValg) {
  assert vareValg >= 0 && vareValg < varer.length : "Varenummer må være mellom 0 og "
      + (varer.length - 1);
  return (varer[vareValg].hentBeholdning() > 0);
}

private void leverVare(int vareValg) {
  // her vil automaten plassere valgt vare i mottakerluken
  giMelding("Varen " + varer[vareValg].hentVarenavn() + " ligger i luken");
}

private void giVekslePenger(double veksel) {
  // her vil automaten gi tilbake vekslepenger i returluken
  giMelding("Vekslepenger " + veksel + " ligger i pengeluken");
  vekslePenger -= veksel;
}

private void oppdaterKassen(double beløp) {
  kasse += beløp;
```

```
}

public void velgVare(int vareValg) {
  if (vareValg < 0 || vareValg >= varer.length) {
    giMelding("Feil vareslag angivelse: " + vareValg);
    return;
  }
  if (vareFinnes(vareValg)) {
    double pris = finnPris(vareValg);
    if (mottattBeløp >= pris) {
      leverVare(vareValg);
      varer[vareValg].settBeholdning(varer[vareValg].hentBeholdning() - 1);
      giVekslePenger(mottattBeløp - pris);
      oppdaterKassen(mottattBeløp);
      mottattBeløp = 0.0;
      giMelding("Takk for handelen og velkommen tilbake!");
    } else { // mottatt beløp mindre enn pris
      giMelding("For lite penger innkast. " + varer[vareValg].hentVarenavn()
          + " koster " + pris + " kr");
    }
  } else { // tom for denne varen
    giMelding("Dessverre tom for " + varer[vareValg].hentVarenavn());
  }
}
```

```
  public void avbryt() {
    giVekslePenger(mottattBeløp);
    giMelding("Vareutlevering er avbrutt.");
    mottattBeløp = 0.0;
  }

  public void visUtvalg() {
    String str = "Automaten inneholder varene:";
    str += this;
    str += "\nHver vare koster: " + pris;
    str += "\nVekslepenger: " + vekslePenger;
    str += "\nKasse: " + kasse;
    str += "\nInnkast: " + mottattBeløp;
    System.out.println(str);
  }
  @Override
  public String toString() { // overkjører toString() fra Object-klassen
    String str = "";
    for (int i = 0; i < varer.length; i++) {
      str += "\nVarenummer: " + i;
      str += varer[i];
    }
    return str;
  }
}
```

Eksempel: Brus-automater (forts.)

```
public class Display {
  // denne klassen bruker System.out for å vise meldinger, og trenger derfor
  // ikke noen feltvariabler
  public void visMelding(String melding) {
    System.out.println(melding);
  }
}
```

Eksempel I: Brus-automater (forts.)

- Klient med enkel test av automaten:

```
public class AutomatKlient {
  public static void main(String[] args) {
    Vare[] brus = { new Vare("Coca cola", 10), new Vare("Pepsi", 10),
        new Vare("Solo", 2), new Vare("Farris", 5) };
    Automat automat = new Automat(brus, 20.0, 100.0);
    automat.mottaPenger(20.0);
    automat.velgVare(1);
    automat.mottaPenger(50.0);
    automat.velgVare(2);
    automat.mottaPenger(17.0);
    automat.velgVare(2);
    automat.mottaPenger(5.0);
    automat.velgVare(2);
    automat.mottaPenger(20.0);
    automat.velgVare(2);
    automat.avbryt();
    automat.visUtvalg();
  }
}
```

Eksempel I: Brus-automater (forts.)

- Kjøring av `AutomatKlient`:

```
Mottatt: 20.0 kr
Varen Pepsi ligger i luken
Vekslepenger 0.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
Mottatt: 50.0 kr
Varen Solo ligger i luken
Vekslepenger 30.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
Mottatt: 17.0 kr
For lite penger innkast. Solo koster 20.0 kr
Mottatt: 22.0 kr
Varen Solo ligger i luken
Vekslepenger 2.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
Mottatt: 20.0 kr
Dessverre tom for Solo
Vekslepenger 20.0 ligger i pengeluken
Vareutlevering er avbrutt.
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola Beholdning: 10
```

```
Varenummer: 1
Varenavn: Pepsi Beholdning: 9
Varenummer: 2
Varenavn: Solo  Beholdning: 0
Varenummer: 3
Varenavn: Farris  Beholdning: 5
Hver vare koster: 20.0
Vekslepenger: 48.0
Kasse: 92.0
Innkast: 0.0
```

Eksempel II: Brus-automater (forts.)

- Klient der brukeren får velge vare fra automaten:

```
import java.util.Scanner;

public class AutomatKlientII {
  public static void main(String[] args) {
    Vare[] varer = { new Vare("Coca cola 0.5l", 20),
        new Vare("Pepsi 0.5l", 10), new Vare("Solo 0.5l", 5),
        new Vare("Farris 0.5l", 5), new Vare("Coca cola 0.3l", 10),
        new Vare("Pepsi 0.3l", 5) };
    Automat automat = new Automat(varer, 20.00, 200.0);
    Scanner tastatur = new Scanner(System.in);
    String svar;
    do {
      automat.visUtvalg();
      System.out.print("Oppgi beløp: ");
      automat.mottaPenger(tastatur.nextDouble());
      System.out.print("Velg brus nummer: ");
      automat.velgVare(tastatur.nextInt());
      System.out.print("Vil du kjøpe mer brus (j/n)? ");
      svar = tastatur.next();
    } while (svar.charAt(0) == 'j' || svar.charAt(0) == 'J');
    System.out.println("Status for automaten:");
    automat.visUtvalg();
```

```
    }
}
```

- Utskrift fra kjøring: (brukerens inndata er understreket.)

```
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 10
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 5
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 10
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5
Hver vare koster: 20.0
Vekslepenger: 200.0
Kasse: 0.0
Innkast:0.0
Oppgi beløp: 23
Mottatt: 23.0 kr
Velg brus nummer: 4
Varen Coca cola 0.3l ligger i luken
Vekslepenger 3.0 ligger i pengeluken
```

```
Takk for handelen og velkommen tilbake!
Vil du kjøpe mer brus (j/n)? j
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 10
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 5
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 9
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5
Hver vare koster: 20.0
Vekslepenger: 197.0
Kasse: 23.0
Innkast:0.0
Oppgi beløp: 25
Mottatt: 25.0 kr
Velg brus nummer: 2
Varen Solo 0.5l ligger i luken
Vekslepenger 5.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
```

```
Vil du kjøpe mer brus (j/n)? n
Status for automaten:
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 10
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 4
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 9
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5
Hver vare koster: 20.0
Vekslepenger: 192.0
Kasse: 48.0
Innkast: 0.0
```

Eksempel III: Brus-automater (forts.)

- Utvidet vareklasse som håndterer pris per enhet:

```
public class VareMedEnhetspris extends Vare {
  double enhetspris;

  VareMedEnhetspris(String varenavn, int beholdning, double pris) {
    super(varenavn, beholdning);
    enhetspris = pris;
  }
  public double hentEnhetspris() { return enhetspris; }
  @Override
  public String toString() { // overkjører toString() fra Vare-klassen
    return super.toString() + "\tEnhetspris: " + enhetspris;
  }
}
```

  - utvider egenskaper (nytt felt `enhetspris`)
  - utvider atferd (ny instansmetode `hentEnhetspris()`)
  - endre atferd (overkjører instansmetoden `toString()` fra `Vare`-klassen)
  - arver alle medlemmer fra `Vare`-klassen
  - bruker superklassens konstruktør
  - bruker superklassens `toString()`-metode vha. nøkkelordet `super`

Eksempel III: Brus-automater (forts.)

- En subklasse for automater som tillater varer med forskjellige priser:

```
public class BrusAutomat extends Automat {

  BrusAutomat(VareMedEnhetspris[] varer, double veksel) {
    super(varer, 0.0, veksel);
  }

  @Override
  public double finnPris(int vareValg) {
    return ((VareMedEnhetspris) varer[vareValg]).hentEnhetspris();
  }

  @Override
  public void visUtvalg() {
    String str = "Automaten inneholder varene:";
    str += super.toString();
    str += "\nVekslepenger: " + super.vekslePenger;
    str += "\nKasse: " + super.kasse;
    str += "\nInnkast:" + super.mottattBeløp;
    System.out.println(str);
  }
}
```

– Utvider atferd (overkjører instansmetoden `finnPris()` som utnytter arvhierarkiet for varer til å finne enhetspris for valgt vare og også `toString()`-metoden)

- Klient som bruker disse subklassene:

```
import java.util.Scanner;

public class BrusAutomatKlient {
  public static void main(String[] args) {
    VareMedEnhetspris[] varer = {
        new VareMedEnhetspris("Coca cola 0.5l", 20, 20.00),
        new VareMedEnhetspris("Pepsi 0.5l", 10, 20.00),
        new VareMedEnhetspris("Solo 0.5l", 5, 20.00),
        new VareMedEnhetspris("Farris 0.5l", 5, 20.00),
        new VareMedEnhetspris("Coca cola 0.3l", 10, 12.00),
        new VareMedEnhetspris("Pepsi 0.3l", 5, 12.50) };
    BrusAutomat automat = new BrusAutomat(varer, 200.0);
    Scanner tastatur = new Scanner(System.in);
    String svar;
    do {
      automat.visUtvalg();
      System.out.print("Oppgi beløp: ");
      automat.mottaPenger(tastatur.nextDouble());
      System.out.print("Velg brus nummer: ");
      automat.velgVare(tastatur.nextInt());
      System.out.print("Vil du kjøpe mer brus (j/n)? ");
```

```
      svar = tastatur.next();
    } while (svar.charAt(0) == 'j' || svar.charAt(0) == 'J');
    System.out.println("Status for automaten:");
    automat.visUtvalg();
  }
}
```

Eksempel III: Brus-automater (forts.)

- En kjøring av `BrusAutomatKlient`-klassen:

```
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 10  Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 5   Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5   Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5   Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 0.0
Innkast:0.0
Oppgi beløp: 18
Mottatt: 18.0 kr
Velg brus nummer: 1
Pepsi 0.5l koster 20.0 kr
Vil du kjøpe mer brus (j/n)? j
```

```
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 10  Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 5   Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5   Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5   Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 0.0
Innkast:18.0
Oppgi beløp: 2
Mottatt: 20.0 kr
Velg brus nummer: 1
Varen Pepsi 0.5l ligger i luken
Vekslepenger 0.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
Vil du kjøpe mer brus (j/n)? n
Status for automaten:
Automaten inneholder varene:
```

```
Varenummer: 0
Varenavn: Coca cola 0.5l        Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
Varenavn: Pepsi 0.5l    Beholdning: 9   Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l     Beholdning: 5   Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l   Beholdning: 5   Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l        Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l    Beholdning: 5   Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 20.0
Innkast:0.0
```

Eksempel III: Brus-automater (forts.)

- Kjøring der test av vare nummer slår til:

```
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l  Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
Varenavn: Pepsi 0.5l  Beholdning: 10  Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l  Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l  Beholdning: 5 Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 0.0
Innkast:0.0
Oppgi beløp: 20
Mottatt: 20.0 kr
Velg brus nummer: 6
Feil vareslag angivelse: 6
Mottatt beløp 20.0 returneres.
Vil du kjøpe mer brus (j/n)? j
```

```
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l  Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
Varenavn: Pepsi 0.5l  Beholdning: 10  Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l  Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l  Beholdning: 5 Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 0.0
Innkast:0.0
Oppgi beløp: 15
Mottatt: 15.0 kr
Velg brus nummer: 2
For lite penger innkast. Solo 0.5l koster 20.0 kr
Vil du kjøpe mer brus (j/n)? j
Automaten inneholder varene:
Varenummer: 0
Varenavn: Coca cola 0.5l  Beholdning: 20  Enhetspris: 20.0
Varenummer: 1
```

```
Varenavn: Pepsi 0.5l  Beholdning: 10  Enhetspris: 20.0
Varenummer: 2
Varenavn: Solo 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 3
Varenavn: Farris 0.5l Beholdning: 5 Enhetspris: 20.0
Varenummer: 4
Varenavn: Coca cola 0.3l  Beholdning: 10  Enhetspris: 12.0
Varenummer: 5
Varenavn: Pepsi 0.3l  Beholdning: 5 Enhetspris: 12.5
Vekslepenger: 200.0
Kasse: 0.0
Innkast:15.0
Oppgi beløp: 5
Mottatt: 20.0 kr
Velg brus nummer: 2
Varen Solo 0.5l ligger i luken
Vekslepenger 0.0 ligger i pengeluken
Takk for handelen og velkommen tilbake!
Vil du kjøpe mer brus (j/n)? n
Status for automaten:
Automaten inneholder varene:
...
```

OOP (Objektorientert programmering)

OOP er en implementasjonsmetode,

der program er organisert som samarbeidende samlinger av objekter,

der hvert objekt er en instans av en klasse, og

der alle klassene er medlemmer i et hierarki av klasser forent ved arv-relasjoner.

Dersom det siste punktet ikke er oppfylt, kalles implementasjonsmetoden for *OBP* (*Objektbasert programmering*).

Metoder + data = objekter

- En av de største fordelene med OOP er *gjenbruk, dvs. at klasser kan brukes om igjen til å lage nye programmer.*
  - Gjenbruk av klasser sparer utviklingstid og reduserer sannsynligheten for feil siden klassene er testet ut tidligere.
  - Utfordringen er å lage *gjenbrukbare klasser!*