Objektorientert design av kode. Refaktorering.

DEL 2

INF1010-forelesning 9. mars

Ragnhild Kobro Runde

Kjennetegn på god design (fra INF1050)

- En god utforming gjør den jobben den er ment å gjøre.
- En god utforming er enkel og elegant.
  - Eleganse innebærer å finne akkurat riktig abstraksjonsnivå.
- En god utforming er gjenbrukbar, utvidbar og enkel å forstå.
- Et godt objekt har et lite og veldefinert ansvarsområde.
- Et godt objekt skjuler implementasjonsdetaljer fra andre objekter.

- Grady Booch

Eksempel: Barnehage-programmet

Et større eksempel: Barnehage

Vi skal lage et program for å administrere en barnehage.
I barnehagen finnes det to typer avdelinger:

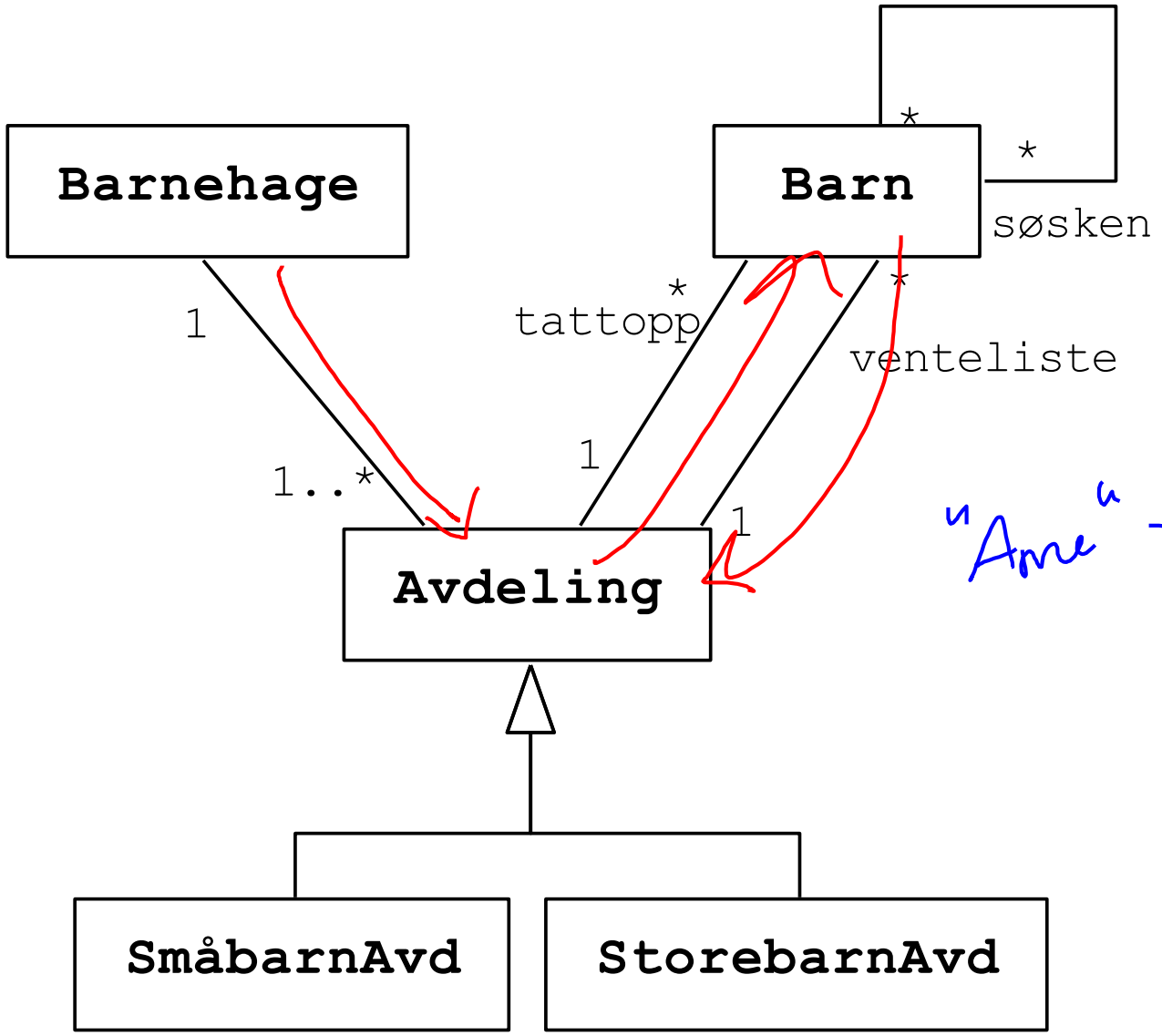
- Småbarnsavdeling
  - Barn opp til 3 år
  - Plass til 9 barn
- Avdeling for større barn
  - Barn over 3 år
  - Plass til 18 barn

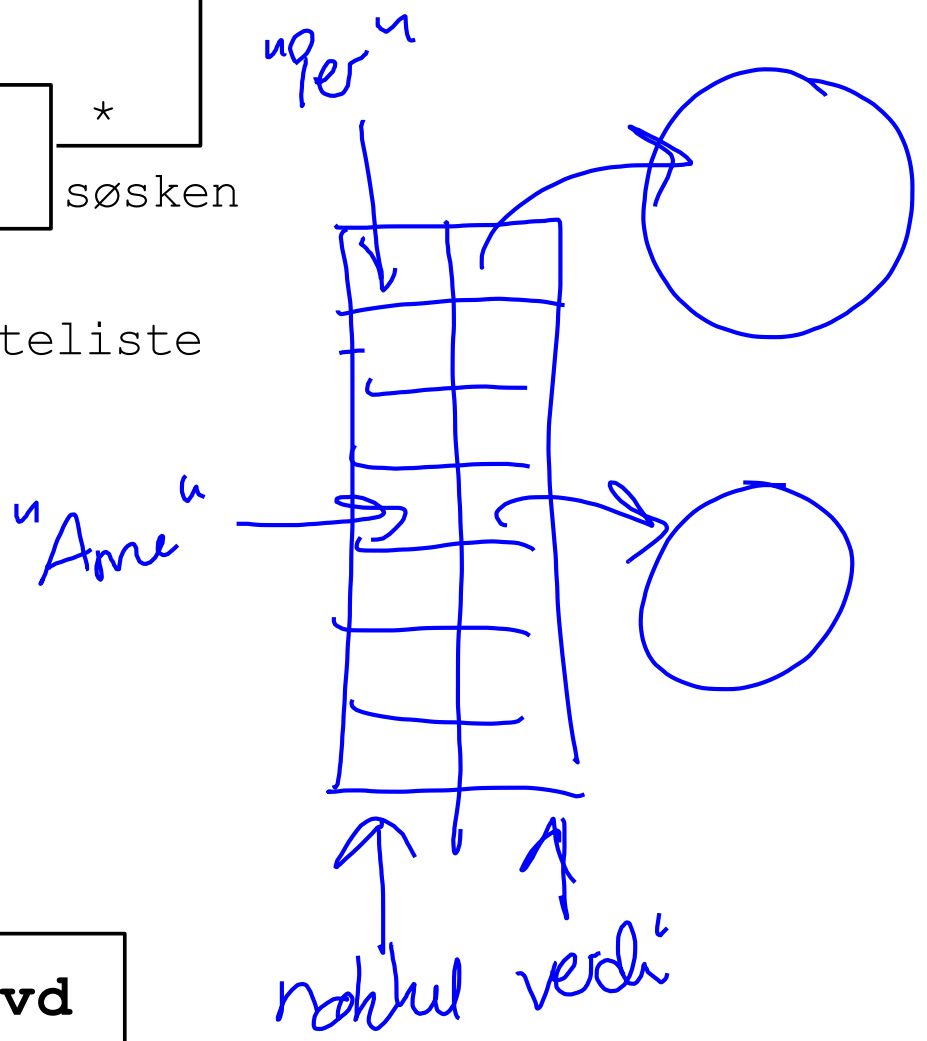
Et barn kan være tatt opp til maksimalt en avdeling. For hver avdeling vedlikeholdes en venteliste. Et barn kan bare stå på en venteliste. Derimot kan et barn både være tatt opp i en småbarnsavdeling, og stå på venteliste til en avdeling for større barn.

Opptak fra ventelisten skjer "ved loddtrekning", men med søskenprioritet ved opptak til småbarnsavdelingene. Til avdelingene for større barn har barn som allerede har plass i en småbarnsavdeling prioritet.

Hvilken datastruktur ville du foreslått her?

Barnehage: Klassediagram





Klassen Avdeling med subklasser

```
class Avdeling {
     String navn;
     int antallPlasser;
     HashMap tattopp;
     HashMap venteliste;

     Avdeling(String navn) {
          this.navn = navn;
          tattopp = new HashMap();
          venteliste = new HashMap();
     }
}
```

```
class SmåbarnAvd extends Avdeling {
     SmåbarnAvd(String navn) {
          super(navn);
          antallPlasser = 9;
     }
}
```

```
class StorebarnAvd extends Avdeling {
     StorebarnAvd(String navn) {
          super(navn);
          antallPlasser = 18;
     }
}
```

Klassene Barn og Barnehage

```
class Barn {
     String navn;
     int født;
     HashMap søsken;
     Avdeling venter, opptak;

     Barn(String navn) {
          this.navn = navn;
          søsken = new HashMap();
     }
}
```

```
class Barnehage {
     HashMap alleBarn = new HashMap();
     HashMap alleAvd = new HashMap();
}
```

Barnehage: noen metoder

- Vi skal konsentrere oss om følgende metoder:
  - **Ny avdeling**
  - **Ledige plasser:** Skriver ut antall ledige småbarnsplasser og antall ledige plasser for store barn totalt i barnehagen.
  - **Søknad:** Setter et barn på venteliste til en avdeling.
  - **Opptak:** Foretar opptak av barn til alle avdelingene i barnehagen.
- I tillegg finnes selvfølgelig metoder for å registrere barn/søsken, skrive informasjon om barn/avdeling, …
- Se kursets hjemmeside for komplett program.

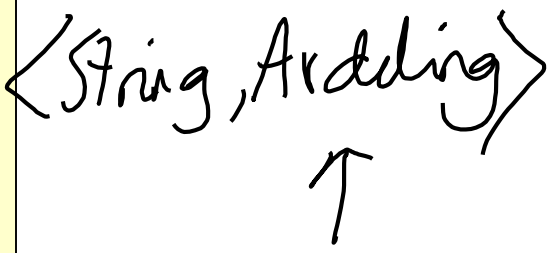
Metoden nyAvdeling()

```
// I klassen Barnehage:

void nyAvdeling() {
     String navn;
     Avdeling a;

     System.out.print("Navn på ny avdeling: ");
     navn = inn.inText();
     if (alleAvd.containsKey(navn)) {
          System.out.println("Registrert allerede");
          return;
     }

     System.out.print("Småbarnsavdeling (j/n)? ");
     if (inn.inChar() == 'j') {
          a = new SmåbarnAvd(navn);
     } else {
          a = new StorebarnAvd(navn);
     }
     alleAvd.put(navn, a);
}
```



Metoden ledigePlasser()

```
// I klassen Barnehage:

void ledigePlasser() {
     int antallSmå = 0;
     int antallStore = 0;
     Iterator it = alleAvd.values().iterator();
     while (it.hasNext()) {
          Avdeling a = (Avdeling) it.next();
          if (a instanceof SmåbarnAvd)
               antallSmå += a.antallLedige();
          else
               antallStore += a.antallLedige();
     }
     System.out.println("Småbarnsplasser: " + antallSmå);
     System.out.println("For store barn: " + antallStore);
}
```

```
// I klassen Avdeling:

int antallLedige() {
     return antallPlasser - tattopp.size();
}
```

Søknad om opptak

```
// I klassen Barnehage:

void søknad() {
     String anavn, bnavn;
     Avdeling a;
     Barn b;

     System.out.print("Avdelingens navn: ");
     anavn = inn.inText();
     System.out.print("Barnets navn: ");
     bnavn = inn.inText();

     a = (Avdeling) alleAvd.get(anavn);
     b = (Barn) alleBarn.get(bnavn);
     if (a == null || b == null) {
          System.out.println("Finner ikke barn/avdeling");
          return;
     }

     a.søknad(b);
}
```

Søknad om opptak (klassen Avdeling)

```
// I klassen Avdeling:

void søknad(Barn b) {
    venteliste.put(b.navn, b);
    b.settVenteliste(this);
}
```

```
// I klassen StorebarnAvd:

void søknad(Barn b) {
    if (b.alder() >= 3) {
        super.søknad(b);
    } else {
        System.out.println("For liten");
    }
}
```

```
// I klassen SmåbarnAvd:

void søknad(Barn b) {
    if (b.alder() < 3) {
        super.søknad(b);
    } else {
        System.out.println("For stor");
    }
}
```

Søknad om opptak (klassen Barn)

```
// I klassen Barn:

int alder() {
   int iår = Calendar.getInstance().get(Calendar.YEAR);
   return iår - født;
}

void settVenteliste(Avdeling a) {
   if (venter != null) {
      venter.fjernFraVenteliste(this);
   }
   venter = a;
}
```

```
// I klassen Avdeling:

void fjernFraVenteliste(Barn b) {
   venteliste.remove(b.navn);
}
```

Opptak

```
// I klassen Barnehage:

void opptak() {
    Iterator it = alleAvd.values().iterator();
    while (it.hasNext()) {
        Avdeling a = (Avdeling) it.next();
        a.opptak();
    }
}
```

Opptak (klassen Avdeling)

```
// I klassen Avdeling:

void opptak() {
  ArrayList søknader = new ArrayList();
  Iterator it = venteliste.values().iterator();
  while (it.hasNext()) {
    Barn b = (Barn) it.next();
    if (prioritert(b)) {
      søknader.add(0,b);      // Legg til først
    } else {
      søknader.add(b);        // Legg til sist
    }
  }

  // Søknadslisten er nå sortert slik at barn med prioritet
  // ligger først i arrayen.
  while (antallLedige() > 0 && søknader.size() > 0) {
    Barn b = (Barn) søknader.remove(0);
    taopp(b);
  }
}
```

Opptak (klassen Avdeling forts.)

```
// I klassen Avdeling:

boolean prioritert(Barn b) {
    return false;
}

void taopp(Barn b) {
    tattopp.put(b.navn, b);
    fjernFraVenteliste(b);
    b.tattopp(this);
}
```

```
// I klassen SmåbarnAvd:

boolean prioritert(Barn b) {
    return b.søskenMedPlass();
}
```

```
// I klassen StorebarnAvd:

boolean prioritert(Barn b) {
    return b.harPlass();
}
```

```
// I klassen Barn:

void tattopp(Avdeling a) {
    if (opptak != null)
        opptak.slutter(this);
    if (venter == a)
        venter = null;
    opptak = a;
}
```

```
// I klassen Avdeling:

void slutter(Barn b) {
  tattopp.remove(b.navn);
}
```

Opptak (klassen Barn)

```
// I klassen Barn:

boolean harPlass() {
     return opptak != null;
}

boolean søskenMedPlass() {
     boolean plass = false;
     Iterator it = søsken.values().iterator();
     while (it.hasNext() && !plass) {
          Barn b = (Barn) it.next();
          if (b.harPlass()) {
               plass = true;
          }
     }
     return plass;
}
```

Ønsket endring 1

- Opptak av barn skjer ikke lenger ved loddtrekning, men slik at de eldste barna inneN hver kategori (avdelingstype, prioritet) blir tildelt plass først.

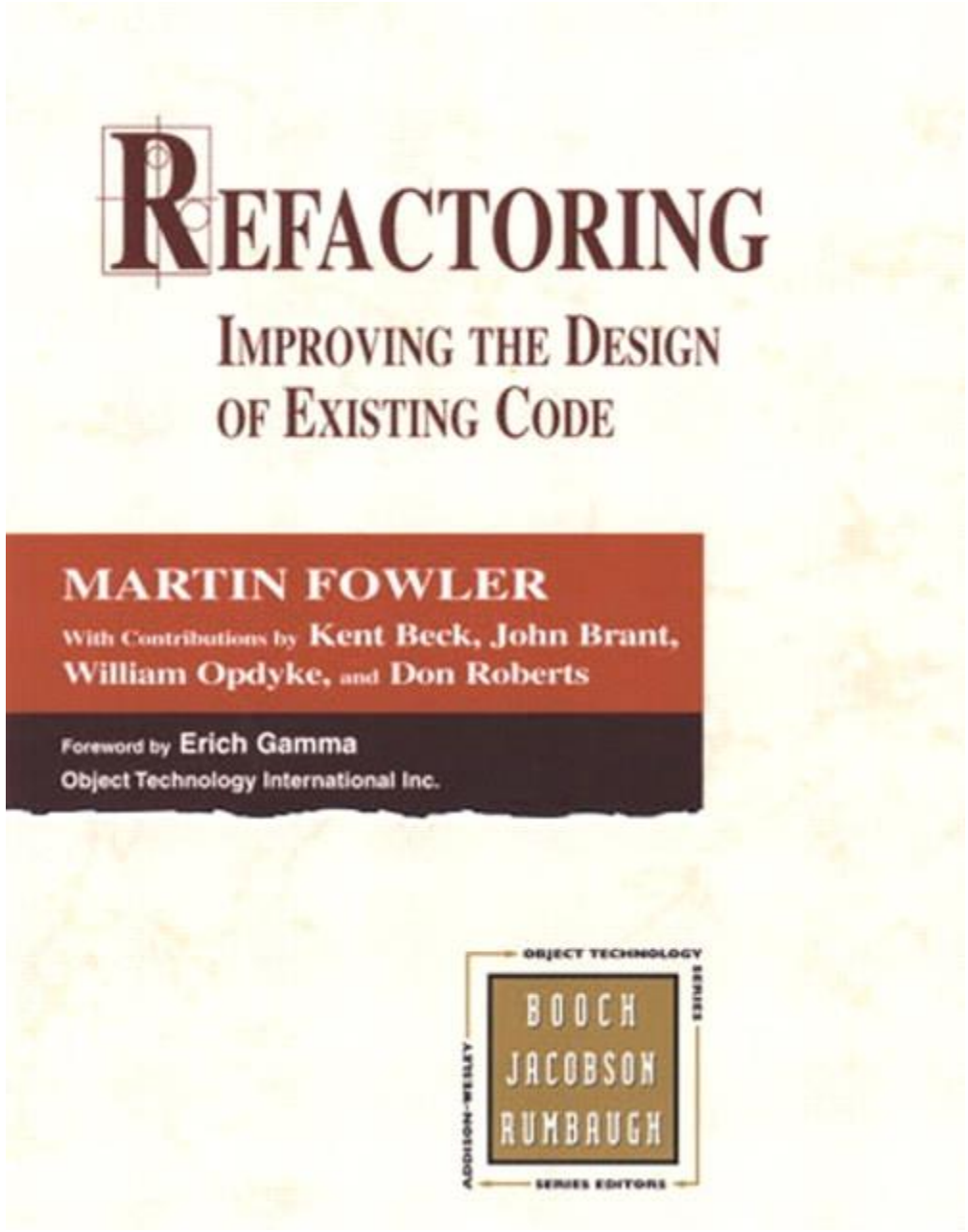
Hva/hvor mye må endres for å få til dette?

Ønsket endring 2

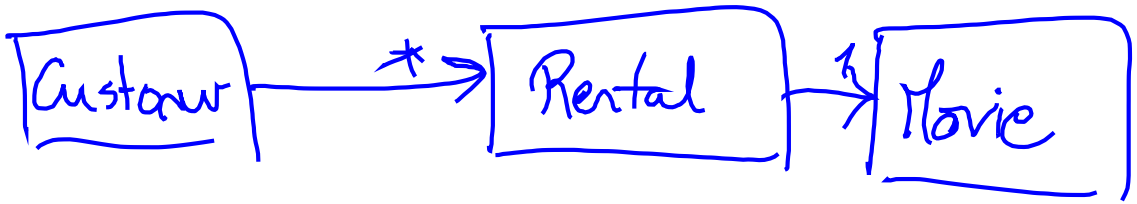
- Systemet skal nå brukes for å administrere flere barnehager, ikke bare en.
- Variant A: Opptak skal fortsatt kjøres separat for hver barnehage.
- Variant B: Opptaksprosessen skal samordnes ved at
  - Det kan søkes om plass i inntil sju barnehager i prioritert rekkefølge.
  - Hver barnehage melder inn antall ledige plasser i hver kategori (små/store barn).
  - Opptak skjer felles etter samme kriterier som før (alder og prioritet).

Hva/hvor mye må endres for å få til dette?

Refaktorering



- Prosessen med å endre et program/system slik at det ikke endrer ekstern oppførsel men likevel forbedrer den interne strukturen.
- Mål: Rydde opp i koden, uten å introdusere (nye) feil.



Generelle tips (Fowler)

- Når du skal legge til funksjonalitet og programkoden ikke er strukturert i forhold til dette – gjør først refaktorering og legg så til den ønskede funksjonaliteten.
- Før du starter med refaktorering, sørg for at du har en solid mengde selv-sjekkende tester.
- Refaktorering endrer programmet i små skritt. Hvis du gjør noe galt, er det lett å finne feilen.
- «Any fool can write code that a computer can understand. Good programmers write code that humans can understand.»