INF1000 Høst 2015

Uke 8:
Mer objektorientert programmering
Siri Moe Jensen

Innhold

- Unified Modelling Language – UML
- En ny type for-løkke
- Organisering av mengder av objekter
  - HashMap
- Valg av representasjon for mengder av objekter

Unified Modeling Language (UML)

Å tegne datastrukturer kan være nyttig

- under planlegging (design) av et nytt program, før koding
- for å diskutere (alternative) løsninger med andre
- som dokumentasjon

For eget bruk: Mindre betydning *hvordan* man tegner. Ved skriftlig kommunikasjon kan det være nyttig å bruke en mer formell notasjon

En modell er alltid en forenkling – må vurdere hva som skal formidles i hvert tilfelle

- UML er den meste kjente standarden innen systemutvikling

Unified Modeling Language (UML)

UML
+ En formell standard og mest kjent: Kan leses av "alle"
+ Det finnes verktøy for enkelt å lage diagrammer

- Standarden er *stor* og omfatter ca 20 typer diagrammer
- Vi bruker en praktisk tilpasning av *klassediagrammer*

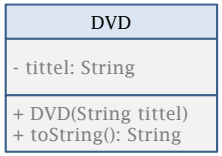
- Klassediagrammer er de vanligste til analyse og design, og kan "oversettes" direkte til objektorientere språk

UML klassediagrammer m/ assosiasjoner

- Nyttig når vi jobber med datastrukturer med flere klasser og relasjoner mellom objekter
  - Design: hvilke klasser, hvilke relasjoner
  - Implementering: oversikt over pekere og objekter
  - (Dokumentasjon: Større krav til formalisme)
- To typer informasjon:
  - om hver enkelt klasse
  - "assosiasjoner" mellom klasser, uformelt: "peker til" (tar ikke stilling til nyanser som også kan uttrykkes i UML)
- Vurder i hvert tilfelle hvilken informasjon som trengs
  (eksamen: Les oppgaveteksten ☺)

Enkeltklasse

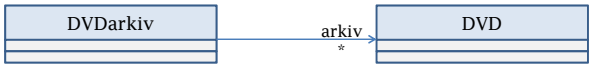
- Alltid med: Klassens navn
- Etter behov/ plass: Datarepresentasjonen
- Etter behov/ plass: Grensesnittet



'-' (eller lås-symbol) angir **`private`**
'+' (eller åpen lås) angir **`public`**

Forhold mellom klasser I

- Uformelt: En *pil* fra A til B illustrerer behov for å navigere fra objekter av klassen A til objekter av klassen B
- Navnet på pilen sier noe om hva referansen representerer
- Tallet angir antall objekter som kan refereres fra ett objekt
  - * betyr 0 eller flere
  - 1..* betyr en eller flere
  - 0..2 betyr 0, 1 eller 2
  - 5 betyr alltid 5



Forhold mellom klasser II

- Hvis det er referanser begge veier, bruker vi en *strek*
- Leses begge veier, tenker oss alltid at vi starter i ett objekt og ser hvor mange objekter dette kan referere til:
  - Én person eier 0 til mange biler
  - Én bil er eid av minst 1 person

- (vi ønsker å kunne finne eieren til et bilobjekt – og vi ønsker å finne ut hvilke(n) bil(er) en person eier)

Fra eksamen 2014

Du har påtatt deg å lage ukeplaner for hele familiens faste aktiviteter. For å gjøre det enkelt antar vi at alle aktiviteter starter på en hel time (kl 00.00, 01.00, 02.00 etc) og at de alle varer nøyaktig 1 time.

Du skal bruke klassene vist i dette UML klassediagrammet:

NB: Klassediagrammet er statisk

⇒ viser ikke øyeblikksbilder av objekter under kjøring – bare *potensialet* i datarepresentasjonen!

⇒ Vi vil fortsatt bruke mer uformelle tegninger for å vise eksempler på hvilke objekter og pekere vi har under kjøring

En ny type for-løkke

enhanced for loop
spesialisert (forenklet) for-løkke)
for-each
for hver

" Spesialisert for-løkke" – "for each"

- En mer kompakt og avansert versjon av for-løkken
- Kan brukes for å gjenta operasjoner på hvert element i blant annet array og ArrayList

- Eksempel, array:

```
//oppretter et char-array og fyller det med verdier av typen char
char[] tegnrekke = {'I', 'N', 'F', '1', '0', '0', '0'};


for (char c: tegnrekke) {
  System.out.println (c);   //skriver ut hvert tegn på en linje
}

for (int i=0; i<tegnrekke.length; i++) {  //gjør det samme
  System.out.println (tegnrekke[i]);
}
```

Spesialisert for-løkke; Begrensninger

1. Kan ikke brukes dersom vi trenger indeksen – variabelen settes lik *innholdet* i hvert element, dermed vet vi ikke inne i løkken hvilken indeks vi er kommet til

2. Variabelen er en *kopi* av innholdet i hver array-posisjon, dvs kan ikke endre innholdet i arrayen

```
char[] tegnrekke = {'I', 'N', 'F', '1', '0', '0', '0'};

// vil fylle arrayen med stjerner i stedet:

for (char c: tegnrekke) {
  c = '*';                  // endrer c, men ikke arrayen tegnrekke!
}
```


Spesialisert for-løkke med ArrayList

- Brukes på samme måte som for array
- En variabel av samme type som innholdet i ArrayList vil være en kopi av ett og ett element i ArrayListen
- Trenger ikke selv holde rede på hvor langt vi er kommet

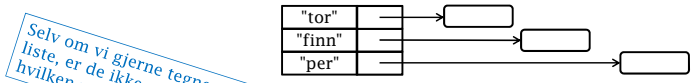
```
ArrayList<String> titler = new ArrayList<String>();
        // fyller titler med innhold – ikke vist her

for (String t: titler) {
  System.out.println (t);   //skriver ut hver tittel på en linje
}
```


HashMap

et nytt verktøy for organisering av objekter

HashMap: En "sekk" objekter I

- Klassen HashMap lar oss lagre (pekere til) objekter uten noen bestemt indre rekkefølge eller nummerering. Den er effektiv og lett å bruke til oppslag.
- Hvert objekt i en HashMap må ha en unik *nøkkel* (en String) som oppgis når vi legger det inn, og brukes for oppslag når noe skal hentes ut



Selv om vi gjerne tegner elementene i en liste, er de ikke nummerert og vi vet ikke hvilken rekkefølge de ligger i!

Bruk av HashMap I

- Importer klassen

```
import java.util.HashMap;
```

- Deklarer og opprett en HashMap

```
HashMap<String, DVD> dvdArkiv = new HashMap<String, DVD> ();
```

- Legg et objekt i en HashMap

```
DVD ny = new DVD ("Hobbiten");
dvdArkiv.put (ny.toString(), ny);
```


Bruk av HashMap II

- Hent (peker til) et objekt med en gitt nøkkel – NB sjekk resultatet før du prøver å bruke objektet!

```
DVD denne = dvdArkiv.get(tittel);
if (denne != null) {..}       // fant et objekt med rett tittel
```

- Sjekk størrelsen (antall elementer)

```
int antall = dvdArkiv.size();
```

- Fjern et objekt med en gitt nøkkel fra en HashMap

```
dvdArkiv.remove ("Hobbiten");
```


Bruk av HashMap III

- Skrive ut alle objektene i en HashMap

```
for (String s: dvdArkiv.keyset()) {
  DNV denne = dvdArkiv.get (s);
  System.out.println (denne.toString());
}
```

.. i en uforutsigbar rekkefølge!

Hvordan organisere en samling objekter av samme klasse?

array
ArrayList
HashMap

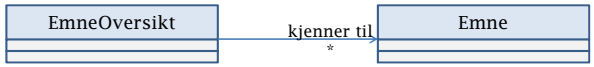
Noen ledetråder

- Skal du lagre et (kjent) antall verdier av en primitiv type (int, boolean, char,..)
  - array
- Er elementene naturlig, løpende nummerert?
  - array eller ArrayList
- Skal du lagre et ukjent/ varierende antall objekter?
  - ArrayList eller Hashmap
- Skal du lagre String- eller andre objekter som det er naturlig å identifisere med en tekst og ikke et nummer?
  - HashMap

Eksempel: Oversikt over emner I

- Oppgave: Et program som kan brukes for å holde oversikt over informatikk-emner
- Programmet skal kunne lese inn data om et nytt emne (emnekode og undervisningssemester), skrive ut alle emner, sjekke om et emne finnes og oppgi hvilket semester et emne med en gitt emnekode undervises
- Hva er aktuelle klasser?
  - Emne
  - EmneOversikt
  - (klassen med main)

Eksempel, UML i designfasen



Eksempel: Oversikt over emner II

```
class Emne {
  private String kode;
  private char semester;

  public Emne (String kode, char sem) {
    this.kode = kode;
    semester = sem;
  }

  public char naar () {
    return semester;
  }

  public String minKode () {
    return kode;
  }

  public String toString () {
    return (minKode() + " (" + semester + ")");
  }
}
```

Klassen EmneOversikt

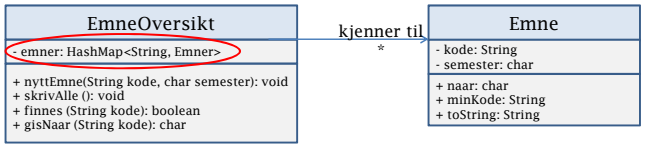
- Grensesnitt for klassen EmneOversikt

```
class EmneOversikt {
  public void nyttEmne (String kode, char semester) {
  }

  public void skrivAlle () {
  }

  public boolean finnes (String kode) {
  }

  public char gisNaar (String kode) {
  }
}
```

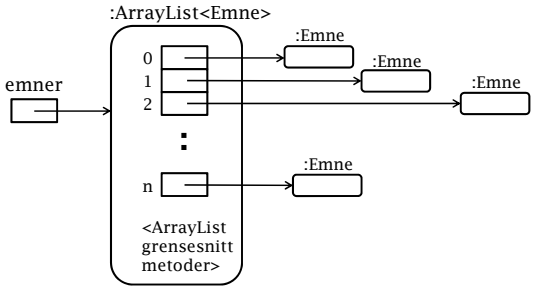

Hvordan organiserer vi Emner i EmneOversikt?

- (Pekere til) objekter, dvs ikke primitiv type
- Ukjent/ varierende antall
- Ingen entydig rekkefølge/ løpende nummerering
- Entydig identifikasjon vha en tekststreng
- Nyttig å kunne slå opp basert på tekststreng

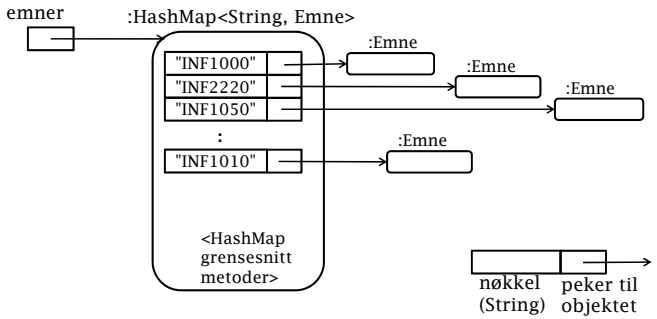
Mer detaljert UML klassediagram



Illustrasjon av ArrayList med emner



Illustrasjon av HashMap med emner



EmneOversikt: Implementasjon med ArrayList I

```
import java.util.ArrayList;

class EmneOverAL {
  private ArrayList<Emne> emner = new ArrayList<Emne>();

  public void nyttEmne (String kode, char semester) {
    Emne nytt = new Emne (kode, semester);
    emner.add(nytt);
  }

  public void skrivAlle () {
    for (Emne e: emner) {
      System.out.println (e.toString());
    }
  }

// fortsetter neste slide
```


EmneOversikt: Implementasjon med ArrayList II

```
// fortsetter fra forrige slide


 public boolean finnes (String kode) {
    for (Emne e: emner) {
      if (kode.equals (e.minKode())) {
        return true;
      }
    }
    return false;
  }
}
```

Må lete gjennom alle for å finne

EmneOversikt: Implementasjon med HashMap

```
import java.util.HashMap;
class EmneOversikt {
  private HashMap<String,Emne> emner = new HashMap<String,Emne>();

  public void nyttEmne (String kode, char semester) {
    Emne nytt = new Emne (kode, semester);
    emner.put(kode, nytt);
  }

  public void skrivAlle () {
    for (String nokkel: emner.keySet()) {
      Emne e = emner.get(nokkel);
      System.out.println (e.toString());
    }
  }

  public boolean finnes (String kode) {
    Emne e = emner.get(kode);
    return (e != null);
  }
}
```

Program med EmneOversikt

```
class EmneProgram {
  public static void main (String[] args) {
    EmneOversikt oversikt = new EmneOversikt ();

    oversikt.nyttEmne ("INF1000", 'H');
    oversikt.nyttEmne ("INF1010", 'V');
    oversikt.nyttEmne ("INF1050", 'V');
    oversikt.nyttEmne ("INF2220", 'H');
    oversikt.skrivAlle();

    System.out.println ("Finnes INF2220? "
        + oversikt.finnes("INF2220"));

    System.out.println ("Finnes INF9999? "
        + oversikt.finnes("INF9999"));

    System.out.println ("Naar gaar INF1050? "
        + oversikt.gisNaar("INF1050"));
  }
}
```

Om vi endrer EmneOversikt-klassen til å bruke ArrayList
isteden (som i EmneOverAL) kan likevel dette programmet
brukes uendret - MEN rekkefølgen i skrivAlle vil være som de ble lagt inn

```
M:\Ifi\Programmering\2015\Uke8>java EmneProgram
INF1010 (V)
INF2220 (H)
INF1050 (V)
INF1000 (H)
Finnes INF2220? true
Finnes INF9999? false
Naar gaar INF1050? V

M:\Ifi\Programmering\2015\Uke8>javac EmneAL.java

M:\Ifi\Programmering\2015\Uke8>java EmneAL
INF1000 (H)
INF1010 (V)
INF1050 (V)
INF2220 (H)
Finnes INF2220? true
Finnes INF9999? false
Naar gaar INF1050? V

M:\Ifi\Programmering\2015\Uke8>
```


INF1000 fremover

INF1000	2015	Forelesning	Obl frister
Uke 7	41	Mekanismer og teknikker for utvikling av mer komplekse, objektorienterte programmer.	
Uke 8	42		
Uke 9	43	Et større program – eksempel.	6
Uke 10	44	Digital representasjon	
Uke 11	45	Repetisjon, gjennomgang av pensum, eksamenstips	7
Uke 12	46	Prøveeksamen	
Uke 13	47	Repetisjonskurs med gruppelærere	
Uke 14	48		
Uke 15	49	Eksamen	

Fagutvalget for Informatikk (FUI) arrangerer INF1000-seminar!!

Lørdag 24. oktober! 11:00-17:00

Pizza!!!

Tips/hjelp til Oblig 7

GRATIS!!!

Foredrag

Bli bedre forberedt til eksamen

Mange gruppelærere hjelper alle som vil!!!

Påmelding kreves – se INF1000-siden/facebook