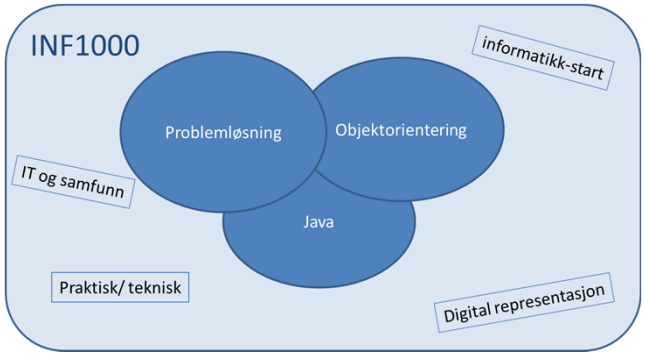
Introduksjon til objektorientert programmering

INF1000 Høst 2015

Grunnkurs i objektorientert programmering



Lokale (og globale) helter

- Kristen Nygaard
- Ole-Johan Dahl

Professorer ved Ifi

Hedersbevisninger:

2000: Kommandør av St Olav

2001: ACM Turing award (Informatikk-Nobelpris)

2002: IEEE John von Neuman medal

=> Hvorfor??

Hva skjedde ~1967?

- Store og komplekse systemer etter ~20 år programmering med de mekanismene dere har lært hittil
- Kompleksitet ga høye kostnader og feil. Arbeidsdeling og gjenbruk ble vanskelig.

Tankesprang: Konseptet objektorientering

Praktisk innfallsvinkel: Splitt og hersk

- *Metoder* kan brukes for å løse deloppgaver
- Hva om deloppgavene handler om å bearbeide felles, kanskje komplekse, data?

Forslag:

➢ Samle relaterte data og kode for å manipulere dem

➢ Nøyaktig hvordan dataene representeres og manipuleres skjules for resten av programmet

Klasser og objekter

- *Klasser* er modeller (beskriver de for oss viktigste egenskapene) av sentrale begreper i det som skal programmeres
- En klasse er et mønster for objekter med samme
  - *grensesnitt* (operasjoner objektet tilbyr omverdenen)
  - *implementasjon* (hvordan operasjonene utføres i Java).
- Under kjøring opprettes *objekter* som instanser av klassene. Hvert objekt har sine egne data som operasjonene i klassen utføres på

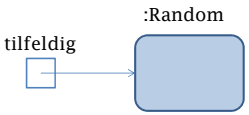
Lite objektorientering hittil i INF1000

- Introduksjon av byggestener for programmering; variable, forgreninger, løkker, metoder mm
- Programmene har bestått av én klasse med **`main`** og andre **`static`** metoder – vi har (nesten) ikke opprettet noen objekter under kjøring
- Heretter skal vi selv skrive klasser, opprette objekter av klassen og utføre metoder på objektene
- Ny måte å tenke på – men vi bruker alle de byggeklossene dere har lært så langt i Java!

Bruk av klasser/ objekter – eksempler

- Vi har brukt flere ferdige klasser allerede – ved å opprette objekter og kalle på metoder i disse
- Eks på klasser vi har brukt:
  - Scanner
  - Random

```
Scanner sc = new Scanner (System.in);
Random tilfeldig = new Random ();
```



Eksempel: Telleverk *

Hva er et telleverk?

- Lett å bruke, en hånd
- Registrerer antall trykk - det vil si hvor langt vi teller
- Lett å lese av tallet

➢ Vi skal nå programmere et telleverk i Java

* Hentet fra Big Java, 8.2

Klassen Teller

- Vi deklarerer en klasse Teller som modell for telleverk
- Hvilke operasjoner skal et Teller-objekt utføre?
  - Registrere et klikk og telle «1 til»
  - Fortelle hvor mange ganger vi har klikket
- Dette beskriver *grensesnittet* for Teller med ord

Grensesnittet for klassen Teller: Java

- Vi deklarerer en metode for hver operasjon:

```
public void leggTil () {} // Øker telleren med 1
public int lesTeller () {} // Leser av telleren
```

- Dette er klassens grensesnitt uttrykt i Java
- **public** angir at disse metodene skal kunne kalles fra utsiden av klassen

Datarepresentasjon i klassen Teller

- Hvilke data trenger hvert objekt å representere?
  - Et heltall som husker antallet mens vi teller
- Vi deklarer *objektvariabelen* **antall** , og angir denne som **private** – kun til bruk inne i klassen.

```
public class Teller {
  private int antall = 0; // Husker hvor langt vi har telt

  public void leggTil () {} // Øk telleren med 1
  public int lesTeller () {} // Leser av telleren
}
```

Implementasjon av klassen Teller

- Til slutt skriver vi innmaten i metodene:

```
public class Teller {
  private int antall = 0; // Husker hvor langt vi har telt

  public int lesTeller () { // Leser av telleren
    return antall;
  }

  public void leggTil () { // Øk telleren med 1
    antall = antall + 1;   // alternativ; antall++;
  }
}
```


Innkapsling

```
private int antall = 0;
```

- Metodene inne i klassen kan endre og lese av variabelen **antall** – mens kode utenfor klassen må kalle en metode på et objekt for å lese den av eller endre den i objektet.

- Innkapling er et sentralt oo prinsipp
- Gir programmereren av klassen full kontroll på hvordan objektvariablene (tilstanden) i et objekt endres

Testing av klassen Teller

Kommentar: Ikke samme fil hvis class Teller er public -> Enkel regel: Én fil per klasse

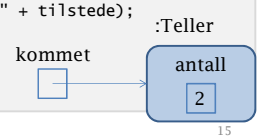
- Test-klasse med main-metode for å teste Teller. Klassene kan ligge i samme eller hver sin fil.

```
public class TestTeller {
  public static void main (String[] args) {

    Teller kommet = new Teller();

    kommet.leggTil();  // En student kommet inn
    kommet.leggTil();  // .. og en til

    int tilstede = kommet.lesTeller();
    System.out.print ("Antall tilstede: " + tilstede);
  }
}
```



Testing av klassen Teller (forts)

- Vi kan bruke klassen vår til å telle forelesere for seg – trenger da et eget objekt for dette

```
....
  Teller kommet = new Teller();
  kommet.leggTil();                          // én student

  Teller forelesere = new Teller();
  forelesere.leggTil();                      // én foreleser
  forelesere.leggTil();                      // en foreleser til

  int tilstede = kommet.lesTeller() + forelesere.lesTeller();
  System.out.print ("Antall tilstede totalt: " + tilstede);
....
```

Flere objekter av klassen Teller

```
....
  Teller kommet = new Teller();
  kommet.leggTil();                          // én student

  Teller forelesere = new Teller();
  forelesere.leggTil();                      // én foreleser
  forelesere.leggTil();                      // en foreleser til

  int tilstede = kommet.lesTeller() + forelesere.lesTeller();
  System.out.print ("Antall tilstede totalt: " + tilstede);
....
```

Grensesnittet til en klasse i Java

- En oversikt over de operasjonene en bruker av klassen kan utføre på objektene
- For hver operasjon deklareres en public metode:
  - En beskrivelse av hva den gjør (kommentar)
  - Modifikator (**`public`**), type, navn og parametre
- Eks: Skriv grensesnittet for klassen Teller

```
// Øker telleren med 1
public void leggTil () {}

// Leser av telleren
public int lesTeller () {}
```


Fremgangsmåte: OOP

1. Identifiser aktuelle klasser (hva er sentrale «ting»/ begreper vi ønsker å modellere?)
2. Design klassens grensesnitt (hvilke operasjoner trenger jeg for objekter av klassen?)
3. Design klassens datarepresentasjon (hvordan skal objektene representere dataene sine?)
4. Implementer (fyll ut) metodene i grensesnittet
5. Lag et testprogram med en main-metode som oppretter et eller flere objekter og kaller på metodene i deres grensesnitt

Eksempel: Termos

- Vi skal skrive en klasse for objekter som representerer termoser med uendelig plass
  - public class Termos {}
- Hvilke operasjoner trenger vi?
  - lage kaffe og fylle på – må kunne angi mengde
  - sjekke hvor mye som er på termosen
  - skjenke fra termosen – angi mengde. «Serverer» kaffen på brukerens terminal.

Eksempel Termos: Grensesnitt

Operasjoner
- lage en gitt mengde kaffe og fylle på
- sjekke hvor mye som er på termosen
- skjenke en gitt mengde fra termos. «Serverer» via terminalen

➢ Skriv grensesnittet i Java m/ metodebeskrivelser

```
// Legger nye desilitere til nivaaet i termosen
public void lagKaffe (int mengde) {}

// Returnerer nivaaet på termosen
public int sjekkNivaa () {}

// Skriver ut antall desilitere til skjerm, reduserer nivaa
// Juster mengden om for lite igjen på termosen
public void skjenkKaffe (int mengde) {}
```


Eksempel Termos: Hele klassen

```
public class Termos {
  private int innhold = 0;  // Antall desiliter innhold

  public void lagKaffe (int mengde) {  // Fyll på mengde
    innhold += mengde;
  }

  public int sjekkNivaa () {           // Returner nivaa
    return innhold;
  }

  public void skjenkKaffe (int mengde) { // Skjenk opp
    if (innhold >= mengde) {
      System.out.println  ("Vaersaagod: " + mengde + " dl");
      innhold = innhold - mengde;
    } else {
      System.out.println ("Slanten: " + (innhold) + " dl");
      innhold = 0;
    }
  }
}
```


Eksempel: Navn

- En klasse for å lagre for-, mellom- og etternavn til en person og presentere det på ulike måter
- Operasjoner som skal tilbys:
  - Presentere navn på «pen» form (For Mellom Etter)
  - Presentere navn på form egnet for sortering (Etter, For Mellom)

➢ Skriv grensesnitt og datarepresentasjon for klassen

Navn: Grensesnitt og datarepresentasjon

```
public class Navn {
  private String fornavn, mellomnavn, etternavn;

  public String penForm () {  // Lager navn i pent format
  }

  public String sortertForm () {  // Lager navn m etternavn først
  }
}
```

- Trenger String-variable for hvert navn
- Når og hvordan gir vi disse verdier?
  - Ved opprettelse av objektet!

➢ Vi trenger en konstruktør

Navn: Grensesnitt versjon 2

- Konstruktør: **public** <klassenavn> <parametere>

```
public class Navn {
  private String fornavn, mellomnavn, etternavn;

  public Navn (String fornvn, String mellom, String etter) {
    fornavn = fornvn;
    mellomnavn = mellom;
    etternavn = etter;
  }

...
}
```

- Utføres ved **new** på klassen Navn – og bare da!

Navn: Metodene

```
public class Navn {
...

  public String penForm () {
    return (fornavn + ' ' + mellomnavn + ' ' + etternavn);
  }

  public String sortertForm () {
    return (etternavn + ", " + fornavn + ' ' + mellomnavn);
  }

}
```


Testprogram for klassen Navn

```
class TestNavn {
  public static void main (String[] args) {

    Navn nn = new Navn ("Trine","Skei","Grande");
    System.out.println (nn.penForm());
    System.out.println (nn.sortertForm());
  }
}
```

```
M:\Ifi\Programmering\2015\Uke 5-8>javac TestNavn.java

M:\Ifi\Programmering\2015\Uke 5-8>java TestNavn
Trine Skei Grande
Grande, Trine Skei

M:\Ifi\Programmering\2015\Uke 5-8>
```


Hvilke klasser kan vi bruke?

- Java har noen innebygde klasser
  - Eks: String
  - Importeres automatisk/ alltid
- Java-biblioteket (Java API)
  - Eks: Scanner, Random
  - Må importeres eksplisitt (import ***)
- Klasser skrevet av oss eller andre programmere
  - Eks: Navn
  - Må være mulig å finne (nå: Samme mappe)

Ikke gjennomgått på forelesning

String – en nesten vanlig klasse

To særegenheter ved String

- String-*konstanter* har egen notasjon:

```
String navn = "Paal"; // Nytt objekt lages implisitt
```

- String-*konkatenering* har egen operator:

```
System.out.println ("Navn er: " + navn);
// Her opprettes det også et nytt objekt bak kulissene
```

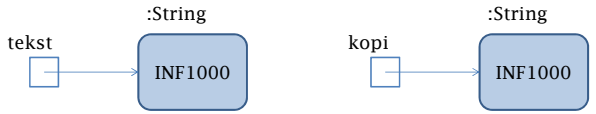

Ikke gjennomgått på forelesning

Sammenligne tekster

- Forskjell: Samme objekt, eller likt innhold?

```
String tekst = "INF1000";

// Nytt String-objekt med samme innhold som tekst
String kopi = new String (tekst);
```



Ikke gjennomgått på forelesning

Å sammenligne tekster

```
String tekst = "Hei";              // Nytt objekt
String kopi = new String (tekst);  // Enda et nytt objekt

// 1. Denne testen gir false:
if (kopi == tekst) {}     // peker ikke på samme objekt

// 2. Denne testen gir true:
if (kopi.equals(tekst)) {}  // to objekter, lik tekst
```

- Test 1 sammenligner pekerne til to ulike objekter
- Test 2 kaller på en metode inne i objektet kopi peker på. Denne metoden tar en peker til en annen String som parameter og sammenligner de to tekstene tegn for tegn

Ikke gjennomgått på forelesning

Sammenligning av Navn-objekter

- Vi kan tilby egne metoder for sammenligning i klassene våre, tilsvarenede **`equals`** i **`String`**-klassen

```
public class Navn {
  private String fornavn, mellomnavn, etternavn;

// ...

  public boolean equals (Navn navn2) {

// Gjør om begge navnene til tekster på samme form
    String tekst1 = sortertForm();
    String tekst2 = navn2.sortertForm();

// Bruker equals-metoden i String for å sjekke om de er like
    return (tekst1.equals(tekst2));
  }
```

Viktigste fra i dag

- Klasser modellerer fysiske eller abstrakte begreper. En klasse angir grensesnitt og implementasjon for objekter av klassen
- Under programkjøring opprettes objekter ved **`new`**. Disse kan vi referere til ved hjelp av pekere
- Via en peker kan vi kalle på metoder i et objekts grensesnitt. Disse metodene kan endre og lese av objektets variable.

Viktigste fra i dag (forts)

Hvordan programmere objektorientert

1. Identifiser aktuell(e) klasse(r)
2. Design klassens grensesnitt
3. Design klassens datarepresentasjon
4. Implementer metodene i grensesnittet
5. Lag et testprogram

Fremover

- Minst 16 poeng på 1-5 for å gå opp til eksamen - men kan følge undervisningen og få tilbakemelding på obliger
- Usikker på poeng? Kontakt gruppelærer!
- Oblig 6 og 7 må være godkjent for å gå opp til eksamen
- Neste uke: IT og samfunn
  - Hvilket ansvar har vi som informatikere?
  - Personopplysningsloven: Oversikt og anvendelse
  - Oblig-fri: Repeter det vi har gjenomgått hittil
- Deretter: Større og mer komplekse datastrukturer og programmer