En introduksjon til objektorientert programmering

- Hvordan oppsto oop?
- Hva er oo?
- To eksempler [Horstmann 8.1–8.3]
- Klassen String [Horstmann 2.5]
- Java-biblioteket [Horstmann appendix D]
- En oppsummering

Hvordan oppnår man slik heder?

Ifi-professorene *Ole-Johan Dahl* og *Kristen Nygaard*:

2000: kommandør av St Olav

2001: ACM Turing award («Informatikkens Nobel-pris»)

2002: IEEE John von Neumann medal

Hva er oo?

Bakgrunnen i 1967 var at man i snart 20 år hadde programmert med variabler, arrayer og metoder. Det lar seg gjøre, men det er lett å miste oversikten når man skal skrive store programmer.

Løsningen

Hva om vi i stedet kan programmere akkurat det vi skal benytte i programmet?

Sagt med andre ord:

Vi kan programmere hvordan et sykkelspeedometer fungerer med variabler og metoder, men hadde det ikke vært bedre å programmere *selve sykkelspeedometeret*?

Eksempel 1: Et sykkelspeedometer

Et sykkelspeedometer er tilkoplet en føler på hjulet og kan

- vise farten
- vise kjørt distanse
- nullstilles
- sette omkretsen av hjulet

Klasser

Vi programmerer ting (både gjenstander og ideer) ved å definere **klasser**. Klassen blir da en **modell** av tingen.

```
class Sykkelspeedometer {
   :
}
```

Grensesnittet

Vi må aller først finne ut hvordan vi skal bruke klassen, med andre ord hvilke operasjoner vi ønsker. Summen av operasjonene kalles klassens **grensesnitt** («interface»).

Grensesnitt for Sykkelspeedometer

- lese farten
- lese distansen
- nullstille distansen
- lese omkretsen av hjulet
- øke omkretsen 1 cm
- senke omkretsen 1 cm

I tillegg finnes

- signal fra sensoren på hjulet
- intern klokke som gir puls hvert millisekund

Grensesnittet beskrives som metoder:

```
class Sykkelspeedometer {
    String visFart() { ... }
    String visDistanse() { ... }
    void nullstill() { ... }

    String visOmkrets() { ... }
    void oekOmkrets() { ... }
    void senkOmkrets() { ... }

    void tellImpuls() { ... }
    void tellMillisek() { ...}
       :
}
```

Det finnes ulike metoder

Hittil har vi brukt metoder for å slippe å gjenta så mye kode. Slike metoder er markert som **static**.

Grensesnittmetodene skal manipulere tilstanden i det enkelte objektet. De er *ikke* markert som static.

Er dette hele sannheten?

Nei; den fulle sannheten åpenbares om to uker.

Representasjon

Når grensesnittet er bestemt, må vi finne ut hvilke data vi må lagre i hvert objekt for at klassen skal fungere slik vi ønsker.

Her finnes det flere mulige alternativer; jeg har valgt:

```
private double omkrets = 0.0;   // Hjulets omkrets
private double distanse = 0.0;  // Distanse hittil
private double tid = 0.0;       // Loepende tid (i ms)
private double tid1 = 0.0;      // Tidspunkt siste impuls
private double tid2 = 0.0;      // Tidspunkt forrige impuls
```

(Legg merke til at alle er **private**.)

Innkapsling

Når vi programmerer, er det lurt å ha så få globale navn (på variabler, metoder etc) som mulig. Med mange navn er det lettere å miste oversikten og dermed gjøre feil.

En fordel med klasser er at både grensesnittet og implementasjonen defineres *inni* klassene; de er **innkapslet** der. Dermed reduserer vi antall globale navn.

Beskyttelse

Det er mulig å bruke variablene i representasjonen som vanlige variabler, men det anbefales ikke. Det er tryggere om de brukes bare av grensesnittmetodene.

Ved å angi at representasjonsvariablene er **private**, vil Java-kompilatoren hjelpe oss med dette.

Til sist kan vi skrive kode i grensesnittmetodene:

Sykkelspeedometer.java

```
class Sykkelspeedometer {
    private double omkrets = 0.0;   // Hjulets omkrets
    private double distanse = 0.0;  // Distanse hittil
    private double tid = 0.0;       // Loepende tid (i ms)
    private double tid1 = 0.0;      // Tidspunkt siste impuls
    private double tid2 = 0.0;      // Tidspunkt forrige impuls

    void nullstill() {
        distanse = 0.0;
    }

    void tellMillisek() {
        tid++;  // Samme som   tid = tid+1;
    }
```

Da er klassen ferdig!

```
void tellImpuls() {
    distanse += omkrets;
        // Samme som   distanse = distanse+omkrets;
    tid2 = tid1;
    tid1 = tid;
}

void oekOmkrets() {
    omkrets += 0.01;
}

void senkOmkrets() {
    omkrets -= 0.01;
}
```

Da er klassen ferdig!

```
    String visDistanse() {
        return "" + distanse;
    }

    String visFart() {
        double tid = (tid2-tid1) / 1000.0;
        return "" + distanse/tid/3.6;
    }

    String visOmkrets() {
        return "" + omkrets;
    }
}
```

Til slutt kan vi lage objekter av klassene våre

NB!

- En **klasse** er bare en mal (dvs en arbeidstegning eller produksjonsbeskrivelse).
- Med **new** lager vi **objekter** (også kalt **instanser**) av klassene. Vi kan lage vilkårlig mange.
- Vi benytter variabler til å holde orden på objektene og utføre metodene.

```
Sykkelspeedometer s = new Sykkelspeedometer();

s.nullstill();
s.tellImpuls();
```

TestSykkelspeedometer.java

```
class TestSykkelspeedometer {
    public static void main(String[] arg) {
        Sykkelspeedometer s = new Sykkelspeedometer();

        // ...
        System.out.println("Jeg har syklet " +
                           s.visDistanse() + " km.");
        System.out.println("Farten er " +
                           s.visFart() + " km/t.");
    }
}
```

Eksempel 2: Et telleverk [Horstmann 8.2]

Et telleverk gjør det enkelt å telle ting, for eksempel personer.

Hvordan lager man en klasse?

1. Hvilke ting/begreper skal vi modellere?
   (Eksamenshint: Hvilke substantiver er brukt i oppgaven?)
2. Hva er grensesnittet?
   Hvilke operasjoner trenger vi overfor objektene?
   (Eksamenshint: Hvilke verb brukes i forbindelse med klassene nevnt i forrige punkt?)
3. Hvordan skal klassen implementeres?
   Hvilke objektvariabler trenger vi for å lagre nok om tilstanden?
4. Lag objekter av klassen for testing.

Oppskrift for å lage en klasse

1. Hva skal vi modellere?
   Svaret er enkelt: et telleverk.

```
class Counter {
  :
}
```

2. Hva er grensesnittet?
   - Vi vil klikke for å telle.
   - Vi ønsker å lese av verdien.

```
class Counter {
  void count() { ... }
  int getValue() { ... }
    :
}
```

3. Hva er representasjonen?
   En enkelt teller (en int) er nok.

```
class Counter {
  private int value;

  void count() { ... }
  int getValue() { ... }
    :
}
```

Den ferdige klassen

Counter.java

```
class Counter {
    private int value;

    void count() {
        value++;
    }

    int getValue() {
        return value;
    }
}
```

Testing

TestCounter.java

```
class TestCounter {
    public static void main (String[] arg) {
        Counter concertCounter = new Counter();
        Counter boardingCounter = new Counter();

        boardingCounter.count();
        boardingCounter.count();
        boardingCounter.count();
        System.out.println("Concert: " +
                           concertCounter.getValue());
        System.out.println("Boarding: " +
                           boardingCounter.getValue());
    }
}
```

```
$ java TestCounter
Concert: 0
Boarding: 3
```

Datatyper

De viktigste er:

int	et heltall	int n = 5;
long	et stort heltall	long v = 0L;
double	et flyt-tall	double pi = 3.14;
char	et tegn	char c = '?';
boolean	sann/usann	boolean t = true;

Datatypen char

- En **String** kan lagre tekster med vilkårlig antall tegn.
- En **char** kan lagre nøyaktig ett tegn.

Datatypen boolean[1]

Vi har brukt tester som er sanne eller usanne i løkker og valgsetninger:

```
if (alder >= 18) ...
while (n > 0 && m > 0) ...
```

Slike tester må gi et resultat **false** eller **true**.

Vi kan lagre slike verdier i **boolean**-variabler:

```
boolean myndig = alder >= 18;

if (myndig) ...
```

[1] George Boole var en engelsk/irsk matematiker.

Klassen String

String er en *nesten* vanlig klasse; den er en del av Javas klassebibliotek så vi kan bruke den til å jobbe med tekster.

Grensesnitt (det viktigste)

- **int compareTo(String other)**
  gjør en sammenligning
- **boolean equals(String other)**
  tester likhet
- **int length()**
  finner tekstens lengde
- **String substring(int begin, int end)**
  plukker ut en del av teksten

Å lage String-objekter

- **new String()**
  gir et nytt String-objekt med tomt innhold.
- **new String(s)**
  gir et nytt String-objekt der innholdet er en kopi av en annen string s.

NB!

Spesielt for akkurat String:

- **"Abc..."** er et String-objekt.
- **s1 + s2** gir et nytt String-objekt som er de to tekstene skjøtt sammen.

Hvor lang er en tekst?

- **int length()**
  forteller oss hvor mange tegn det er i teksten.

Et eksempel

TestStringLength.java

```
class TestStringLength {
    public static void main (String[] arg) {
        String g = new String("Grunnkurs");
        String o = "objektorientert";

        System.out.println("L1: " + g.length());
        System.out.println("L2: " + "i".length());
        System.out.println("L3: " + o.length());
    }
}
```

```
$ javac TestStringLength.java
$ java TestStringLength
L1: 9
L2: 1
L3: 15
```

Sammenligning av tekster

- **boolean equals(String other)**
  sammenligner to tekster og gir **sann** om de er helt like.
- **int compareTo(String other)**
  forteller om den andre teksten kommer før eller etter i en alfabetisk sortering:

$$\text{a.compareTo(b) gir} \begin{cases} < 0 & \text{om a kommer } \textit{før} \text{ b} \\ = 0 & \text{om a og b er } \textit{like} \\ > 0 & \text{om a kommer } \textit{etter} \text{ b} \end{cases}$$

(Dette fungerer egentlig bare for engelsk; for norsk med æøå må man benytte andre teknikker.)

Et eksempel

TestStringSammenligning.java

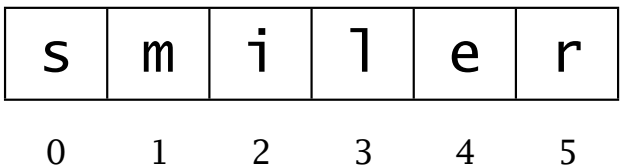
```
class TestStringSammenligning {
  public static void main(String[] arg) {
    System.out.println("Per vs Per: " + "Per".equals("Per"));
    System.out.println("Per vs per: " + "Per".equals("per"));
    System.out.println("Per vs Ida: " + "Per".compareTo("Ida"));
  }
}
```

```
$ javac TestStringSammenligning.java
$ java TestStringSammenligning
Per vs Per: true
Per vs per: false
Per vs Ida: 7
```

Å hente ut deler av en tekst

- **String substring(a, b)**
  lager et nytt String-objekt med innhold fra deler av teksten *fra og med* a *til (men ikke med)* b.

"smiler".substring(1,4)



⇒ "mil"

Parameteren til main er en array med String-er som inneholder parametre oppgitt når vi kjører programmet.

TestStringInitialer.java

```
class TestStringInitialer {
    public static void main(String[] arg) {
        String hun = arg[0];
        String han = arg[1];

        System.out.println(hun.substring(0,1) + " + " +
                           han.substring(0,1) + " = SANT");
    }
}
```

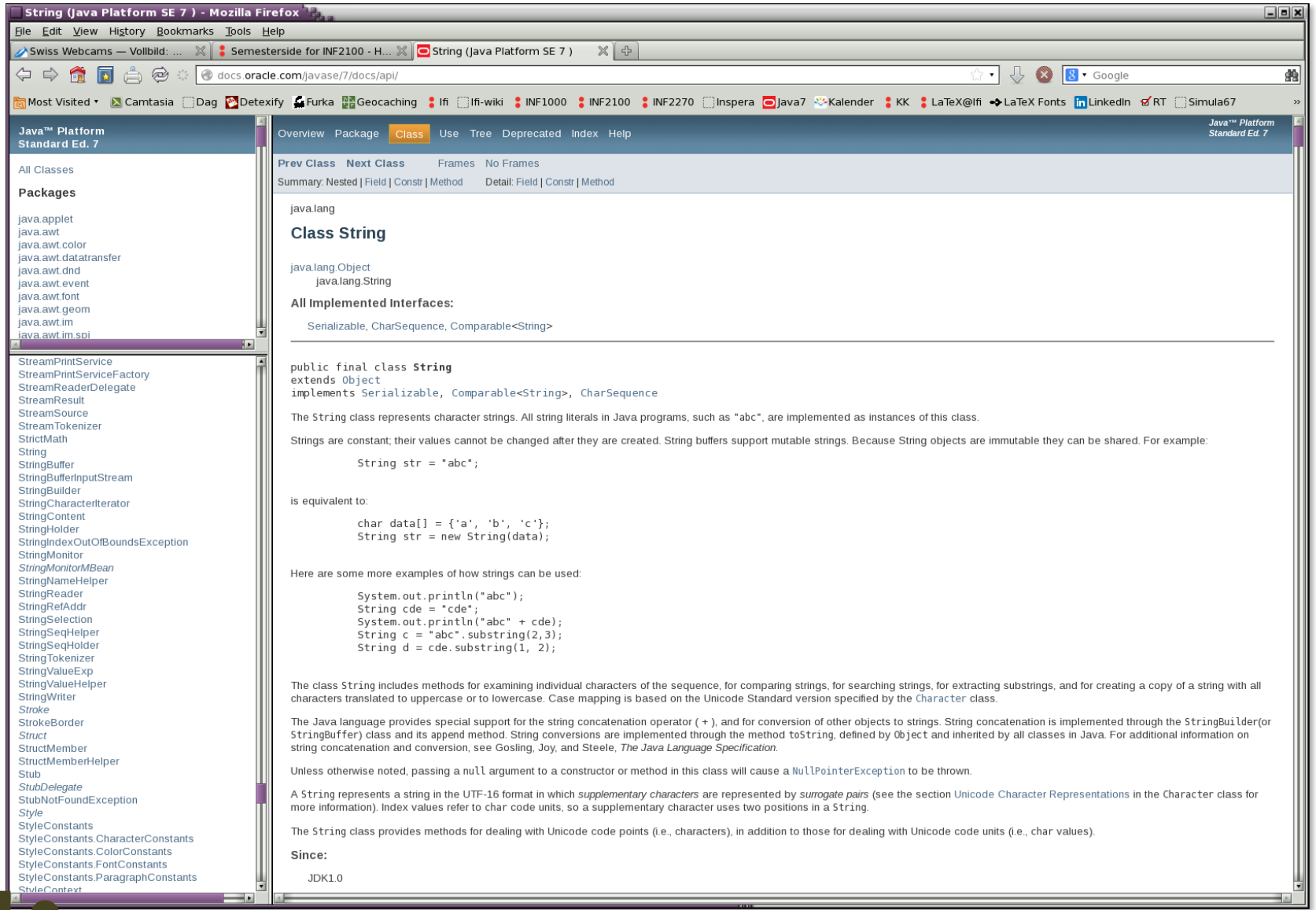
```
$ javac TestStringInitialer.java
$ java TestStringInitialer Kari Ola
K + O = SANT
```

Java-biblioteket

Java har et bibliotek med hundrevis av klasser. Heldigvis er det godt beskrevet:

- Appendix D i læreboken
- http://docs.oracle.com/javase/7/docs/api/

Hva er grensesnittet til String?



Hint:

Ikke gå dere vill i biblioteket!

Java-biblioteket er fascinerende lesing, men

- Slå opp når dere trenger svar på konkrete spørsmål (som «Hva angir 2. parameter til String.substring»?).
- Unngå leting à la «Jeg vet ikke helt hva jeg trenger, men kanskje jeg finner noe jeg kan bruke?»
- Let mer en gang dere har bedre tid.

Hovedpunkter

- Med **klasser** kan man **modellere** både virkelige ting og nyttige begreper.
- En klasse er en mal/konstruksjonstegning av det som skal modelleres.
- **Grensesnittet** til klassen angis med metoder.
- Vi må finne ut hvilken **representasjon** (dvs hvilke objektvariabler) som er nødvendig.
- Med **new** kan vi lage **objekter** basert på klassene.
- En variabel kan peke på et objekt; ved hjelp av den kan vi utføre metodene i grensesnittet.