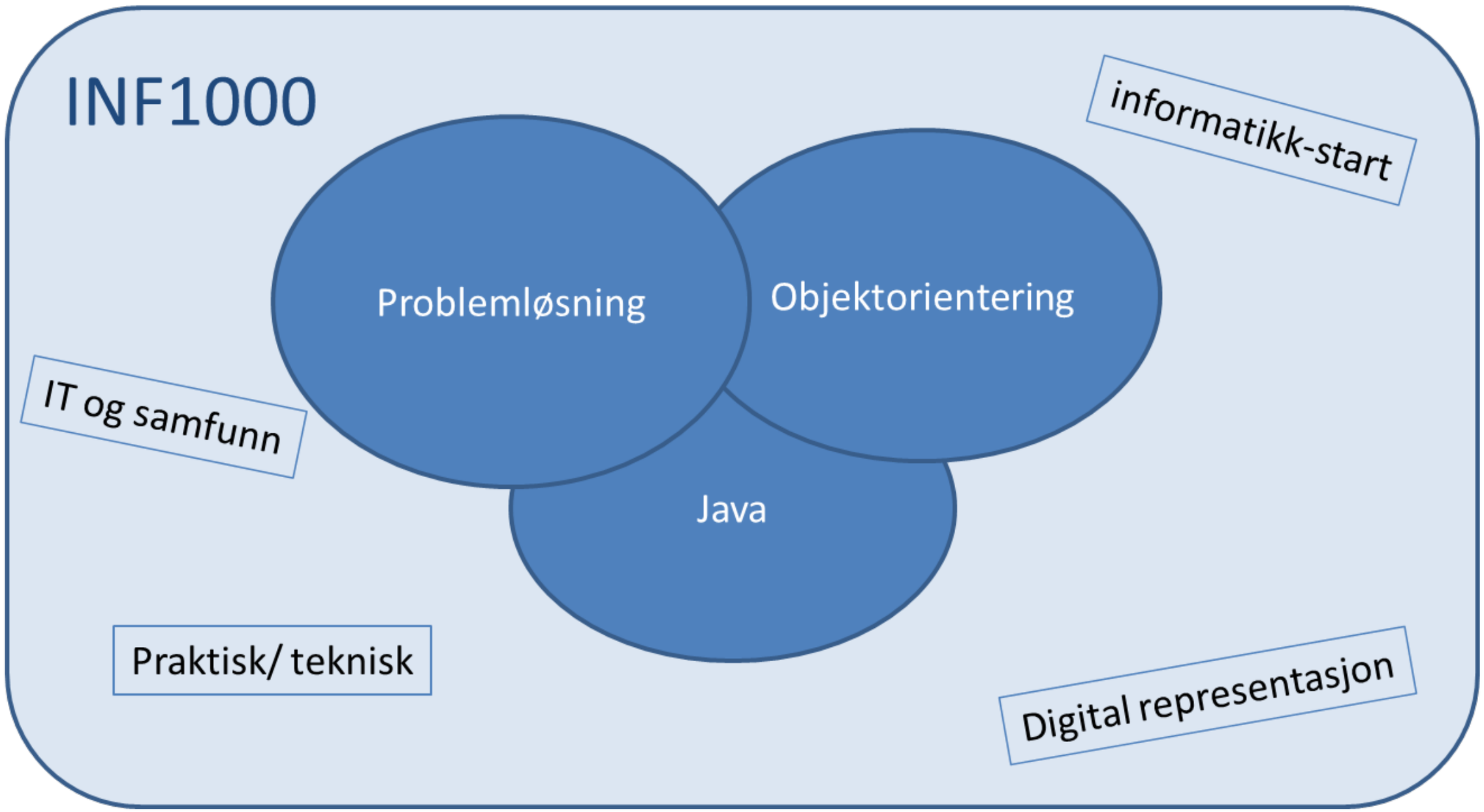
Introduksjon til objektorientert programmering

Samt litt mer om strenger og variable

INF1000, uke6

Ragnhild Kobro Runde

Grunnkurs i objektorientert programmering



Strategi: 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 i objekter

- Objektene tilbyr resten av programmet et sett metoder

grensesnitt

- Hvordan objektet representerer og manipulerer data er skjult

innkapsling

Vi har tidligere brukt objekter som

- ..representerer tekststrenger

```
String navn = "Eric";
if (navn.equals("INF1000")) { … }
```

- ..representerer innlesning

```
Scanner inn = new Scanner(System.in);
String tekst = inn.nextLine();
```

Objekter vi definerer selv brukes ofte

- ..for å representere ting/enheter i problemområdet
  - personer, biler, studenter, bankkontoer, …

- Disse objektene har ulike egenskaper og oppførsel, og de samhandler med hverandre.
- Objekter av samme type (samme egenskaper og oppførsel) sier vi tilhører samme klasse.

```
String navn = "Eric";
String kurs = "INF1000";
```

To objekter av samme klasse (String)

Et nærmere syn på String-metoden int length()

- Dette er en instans-metode:
  - Navnet på metoden er "length".
  - Den tar ikke inn parametre.
  - Den returnerer en verdi av type int (antallet tegn i en String).
- Denne kalles på et String-objekt, dvs en instans av klassen String:

```
"hallo verden".length();          // Gir lengden 12
String tekst = "hallo verden";
tekst.length();                   // Gir lengden 12
```

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!

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 *instansvariabelen* **`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++;
  }
}
```

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

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 () {}
```

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 instansvariablene (tilstanden) i et objekt endres

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

Hvor mange klasser (og objekter) er det?

- Hva vi velger å betrakte som et objekt, og hvilke klasser vi bruker for å beskrive en problemstilling, er ikke bestemt på forhånd.
- Varierer etter hvor mye detaljer vi trenger og hvilke spørsmål vi ønsker å kunne gi svar på:
  - Beskrive problemet med veitrafikk og køer på veiene:
    - Telle antall biler og kanskje skille mellom lastebiler, busser og personbiler
    - Men neppe mer…
  - Beskrive problemet til en bilfabrikk
    - Trenger en detaljert og komplisert beskrivelse av hver bil (bestående av motor, hjul, karosseri, lys,..., hver beskrevet med sin klasse)
    - Mange ulike typer av biler
    - Har da en rekke klasser som hver beskriver sine objekter.

Objektorientert Programmering

- Når vi betrakter et problem vi skal lage et datasystem for, gjør vi to avgrensninger:
  - Vi ser bare på en del av verden (vårt problemområde)
  - Innenfor problemområdet betrakter og beskriver vi bare det som er der med en viss detaljeringsgrad – bare så mange detaljer vi trenger for å svare på de spørsmål datasystemet skal kunne gi svar på.

- Eks: Hvordan beskrive en student ? Skal vi lage:
  - Studentregister: bare registrere navn, personnummer, adresse, tidligere utdannelse og kurs (avlagte og kurs vedkommende tar nå)
  - Legesystem for studenter: ta med svært mange opplysninger om hver student (medisiner, sykdommer, resultat fra blodprøver, vekt...) som vi ikke ville drømme om å ha i et vanlig studentregister

Oppgave: Brusautomat

- For å kjøpe en brusflaske må kunden putte på minst 20kr. Når det er gjort, utleveres en flaske i åpningen nederst på automaten. Det gis ikke vekslepenger. Automates kan fylles med flere flasker. Noen ganger tømmes maskinen for penger. Målet er å til enhver tid vite hvor mange flasker og hvor mye penger som er i maskinen til enhver tid.

1. Design grensesnittet for klassen Brusautomat.
2. Design en passende datarepresentasjon.

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 med etternavn først
}
```

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

- Vi trenger en konstruktør

Navn: Konstruktør

- Konstruktør: public <klassenavn> (<parametere>)

```
public class Navn {
    private String fornavn, mellomnavn, etternavn;
    public Navn (String for, String mellom, String etter) {
        fornavn = for;
        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);
  }
}
```

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 fil el. mappe)

String – en nesten vanlig klasse

- To særegenheter ved String

- String-konstanter har egen notasjon:

```
String navn = "Eric";
```

- String-konkatenering har egen operator:

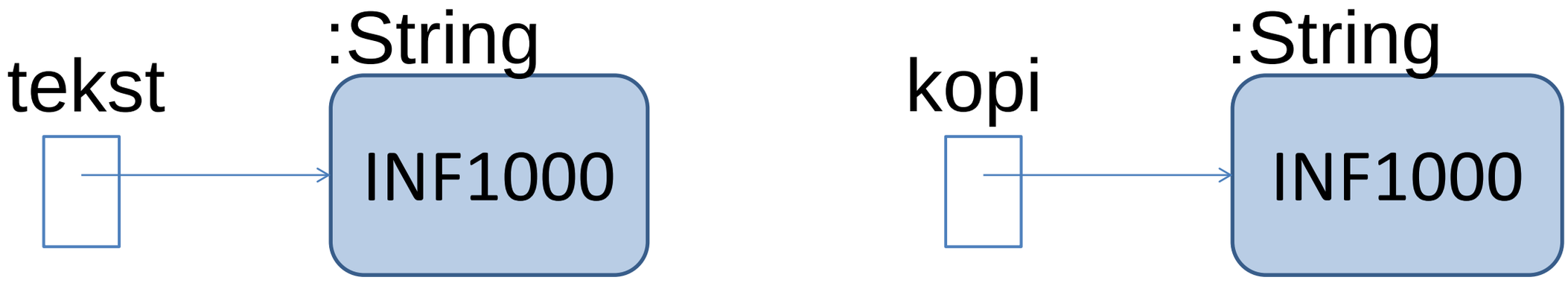
```
System.out.println("Navn: " + navn);
```

Sammenligne tekster

- Forskjell: Samme objekt, eller likt innhold?

```
String tekst = "INF1000";

String kopi = new String (tekst);
```



Å sammenligne tekster

```
String tekst = "INF1000";

String kopi = new String (tekst);

// 1. Denne testen gir false:

if (kopi == tekst) {...}

// 2. Denne testen gir true:

if (kopi.equals(tekst)) {...}
```

- Test 1 (**==**) sammenligner pekerne til **to ulike objekter**

- Test 2 (**equals**) 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**

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.

- 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