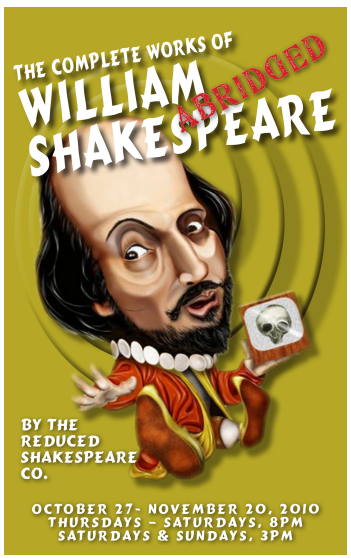
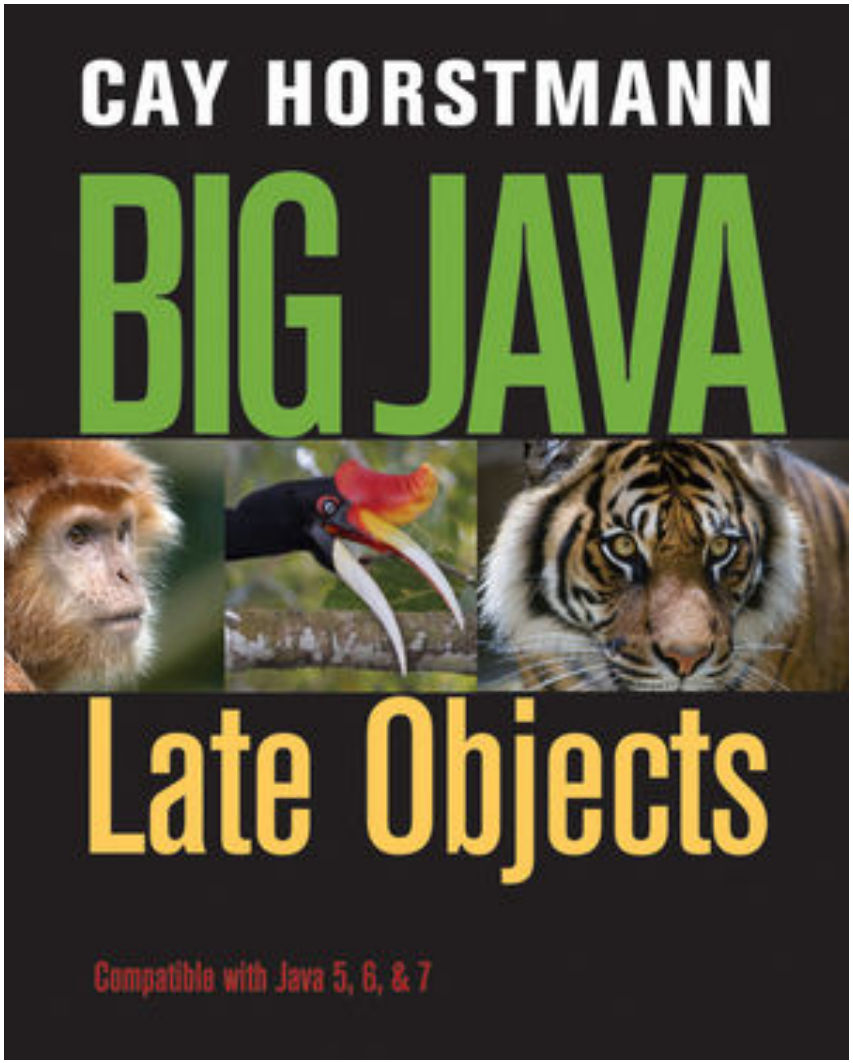
Læreboken på 45 minutter

- Hva er viktig?
- Hva er ikke fullt så viktig?





Hvorfor har vi en lærebok?

- Læreboken forteller stort sett mer detaljert enn forelesningene.
- Det er en fordel å få ting fortalt på to ulike måter.
- Læreboken er bedre å slå opp i.
- *Random fact* og *Special topic* er artig å lese og utdyper faget (selv om de ikke er pensum).
- *Common error*, *Programming tip* og *Self check* er nyttige (men de er heller ikke pensum).

Kapittel 1: Introduction

Innholder først og fremst bakgrunnsinformasjon for det som kommer senere.

«Programmering er problemløsing» [BJ: 1.7]

Nyttig lærdom: Det første viktige steget i programmering er å omforme problemet til en **algoritme**, dvs gi det en form som datamaskinen kan løse. Så kan algoritmen skrives i Java.

Kapittel 2: Fundamental data types

Dette kapittelet omhandler tre ting:

1. numeriske typer og uttrykk; **int** og **double** er viktigst.
2. I/O (lesing og skriving) med
   - Scanner/hasNextLine/nextLine og eventuelt Integer.parseInt
   - System.out.println
3. String

Dette er stoff vi regner med at dere kan bruke ved programmering; unntaket er formatert utskrift [BJ: 2.3.2] som ikke er pensum.

Kapittel 3: Decisions

Dette kapittelet tar for seg det som har med valg å gjøre:

1. **if**-setninger
2. ulike typer sammenligninger
3. datatypen **bool**

Dette må dere kunne bruke i programmering; unntaket er flytskjemaer [BJ: 3.5] som ikke er pensum.

Kapittel 4: Loops

Nå er tiden kommet til løkker:

1. **while**-løkker går så lenge en betingelse gjelder
2. **for**-løkker går et gitt antall ganger

Dette er uunnværlige redskaper i en programmerers verktøykasse. Omtrent alle programmer inneholder en løkke.

Dette er ikke pensum:

- **do**-løkker [BJ: 4.4]
- simulering og tilfeldige tall [BJ: 4.9] (selv om det har vært forelest som eksempel)

Kapittel 5: Methods

Nå begynner det å bli mer avansert. Alt i kapittelet er sentralt pensum (unntatt rekursive metoder [BJ: 5.9]):

[BJ: 5.1] om metoder som «svarte bokser» for lettere å holde oversikten over programmet.

[BJ: 5.2 og 5.6] Hva bruker vi metoder til?

- unngå å gjenta kode (**static**-metoder)
- definere grensesnittet til en klasse (se kapittel 8)

[BJ: 5.3] Parametre! Noe av det aller viktigste i hele kurset!

[BJ: 5.4–5] Returverdier (for de metoder som ikke er **void**-metoder)

[BJ: 5.7] Problemløsing med stegvis forfining beskriver en god teknikk til å la et program bli til litt etter litt.

[BJ: 5.8] En variabels *skop* er delen av et program en variabel er tilgjengelig. Enkelt sagt: En lokal metodevariabel finnes bare når metoden utføres.

Kapittel 6: Arrays and array lists

Array-er er en veldig mye brukt datastruktur.

[BJ: 6.1] introdusere array-er.

[BJ: 6.2] spesialisert **for**-løkke brukes for ArrayList og HashMap.

[BJ: 6.3] gir eksempler på bruk av array-er.

[BJ: 6.4] beskriver hvordan man bruker array-er i metoder.

[BJ: 6.5] er et nyttig avsnitt om hvordan man kan løse problemer ved å bruker array-er.

[BJ: 6.6] todimensjonale arrayer er ikke pensum.

[BJ: 6.7] tar for seg klassen ArrayList som man kan bruke når man ikke vet arrayens størrelsen på forhånd.

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

class More {
    public static void main(String[] arg) throws Exception {
        ArrayList<String> buffer = new ArrayList<>();
        Scanner s = new Scanner(new File(arg[0]));

        while (s.hasNextLine()) {
            buffer.add(s.nextLine());
        }
        for (int i = 0;  i < buffer.size();  ++i) {
            System.out.println((i+1) + ": " + buffer.get(i));
        }
    }
}
```

Kapittel 7: Input/output and exception handling

Dette kapittelet tar opp mye mer om lesing og skriving enn vi trenger i INF1000, så det lille heftet om I/O du finner på nettsiden, er vårt pensum isteden.

Unntakshåndtering er ikke pensum.

Kapittel 8: Objects and classes

Dette er det viktigste kapittelet i INF1000; kurset heter jo *Grunnkurs i objektorientert programmering*. Hele kapittelet er pensum.

Jeg vil anbefale å legge like mye vekt på forelesningspresentasjonene som på læreboken for å få fullt innblikk i objektorientert programmering.

Fremgangsmåte

1. Finn ut hvilke klasser vi trenger og hvordan de skal henge sammen; her er UML-diagram nyttig.

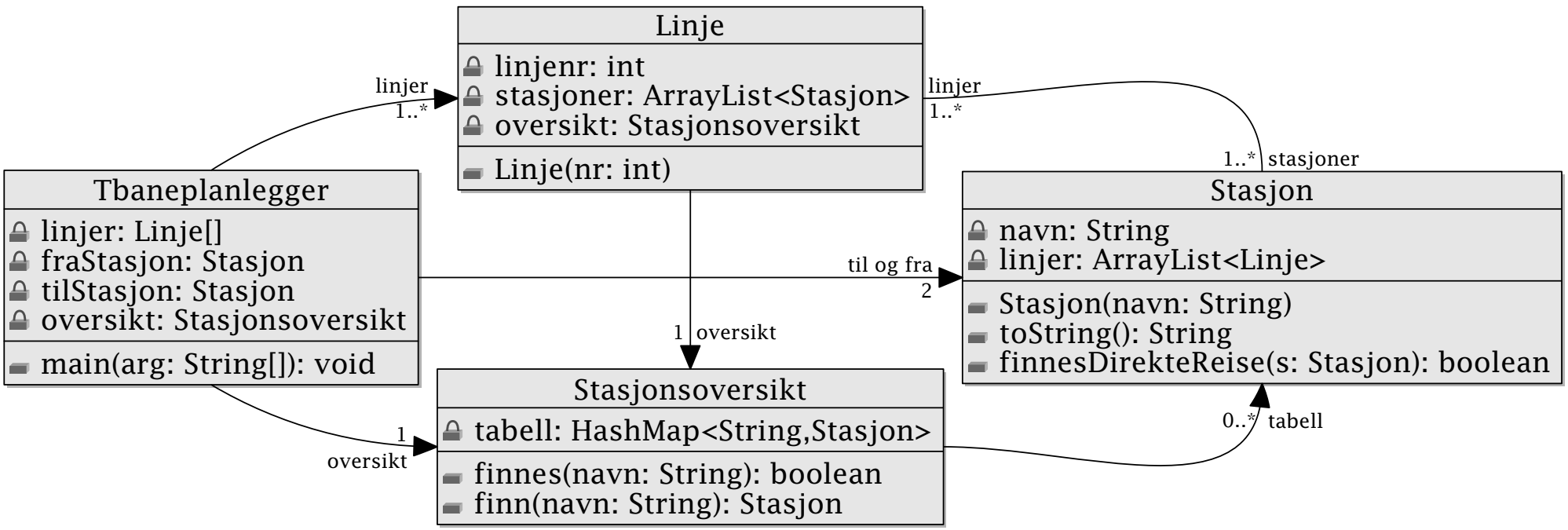
   Husk: En klasse skal representere noe spesifikt, enten et konsept eller en fysisk gjenstand.
2. Definer grensesnittet (dvs metodene).
3. Finn representasjonen (dvs objektvariablene).
4. Programmer grensesnittmetodene.

Annet forelest pensumstoff

UML klassediagrammer [BJ: app K]

UML klassediagrammer er et nyttig verktøy. De brukes under planleggingen av programmeringen og viser klassene (med representasjon og grensesnitt) og hvordan de forholder seg til andre klasser.

Relasjonsangivelsen (navnet og antallet) gir et grunnlag for å definere mye av representasjonen.


Linje
linjenr: int
stasjoner: ArrayList<Stasjon>
oversikt: Stasjonsoversikt
Linje(nr: int)
linjer
1..*
linjer
1..*
1..*
stasjoner
Tbaneplanlegger
linjer: Linje[]
fraStasjon: Stasjon
tilStasjon: Stasjon
oversikt: Stasjonsoversikt
main(arg: String[]): void
til og fra
2
Stasjon
navn: String
linjer: ArrayList<Linje>
Stasjon(navn: String)
toString(): String
finnesDirekteReise(s: Stasjon): boolean
1
oversikt
1
oversikt
Stasjonsoversikt
tabell: HashMap<String,Stasjon>
finnes(navn: String): boolean
finn(navn: String): Stasjon
0..*
tabell

Oppslagstabeller (HashMap) [BJ: 15.4]

En **HashMap** er som en ArrayList der indeksen er en String i stedet for et heltall. Viktig.

```
import java.util.HashMap;

class Hoh {
    public static void main(String[] arg) {
        HashMap<String,Integer> hoh = new HashMap<>();
        hoh.put("Oslo", 10);
        hoh.put("Bergen", 5);
        hoh.put("Hamar", 127);

        for (int i = 0;  i < arg.length;  i++) {
            if (hoh.containsKey(arg[i])) {
                System.out.println(arg[i] + " ligger " +
                    hoh.get(arg[i]) + " moh.");
            } else {
                System.out.println(arg[i] + " er ukjent.");
            }
        }
    }
}
```

Tegnsett [BJ: RF 2.2, app A]

Det finnes mange tegnsett; de mest brukte er **ISO Latin-1** (= **ISO 8859-1**) og **Unicode**; den siste kodes gjerne som **UTF-8**.

Dette er ingen viktig del av pensum.

Digital representasjon av tall [BJ: app I]

Dere bør kunne konvertere små tall ≤ 64 fra desimalt til binært og heksadesimalt og omvendt; ellers ikke viktig i dette kurset.

Digital koding av lyd

Ikke pensum.

Digital koding av bilder

Ikke pensum.

IT og samfunn, personvern

Dette er viktig i pensum. For mer informasjon, se Siris presentasjon i uke 6.

Til sist

Husk at:

- Det er lurt å lese teori, men praksis er det man virkelig lærer programmering av.
- Oversikt er viktigere enn detaljer under eksamen.
- Formålet med en eksamen er å overbevise sensorene om at du kan det viktige i dette stoffet.