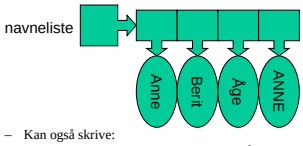
En tabell av referanser, eksempel strenger

- Hvert enkelt element i tabellen består av en referanse til String:
  - String[] navneliste = new String[4];
- Dette er en tabell av referanser. Hver enkelt av disse referansene må vi sette til å peke til objekter av klassen String:
  - navneliste[0] = new String("Anne");
  - navneliste[1] = "Berit"; // kortform går bra
- Objektet kan også være retur fra en metode som lager et String-objekt:
  - navneliste[2] = = JOptionPane.showInputDialog("Skriv et navn: ");
  - navneliste[3] = navneliste[0].toUpperCase();



- Kan også skrive:
  - String[] navneliste = {"Anne", "Berit", "Åge", "ANNE"};

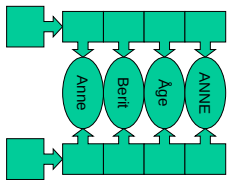
Gjennomgå programliste 7.8 side 276-277.

Programliste 7.8

```
/* TabellAvNavn.java  E.L. 2004  (2 2)
* Et lite program som oppretter en tabell av ti String -referanser.Programmet går i løkke og leser inn navn som legges i
* denne tabellen. Løkka stopper når tabellen er full eller brukeren har skrevet inn et blankt (tomt) navn.
*/

import static javax.swing.JOptionPane.*;
class TabellAvNavn {
  public static void main(String[] args) {
    String[] navnene = new String[10];
    int antNavn = 0;

    String navn = showInputDialog("Oppgi navn (avslutt med blank): ");
    navn = navn.trim();
    while (antNavn < navnene.length && !navn.equals("")) {
      navnene[antNavn] = navn;
      antNavn++;
      navn = showInputDialog("Oppgi navn (avslutt med blank): ");
      navn = navn.trim();
    }
    if (antNavn == navnene.length && !navn.equals("")) {
      showMessageDialog(null, "Ikke plass til flere navn.");
    }
    for (int i = 0; i < antNavn; i++) System.out.println(navnene[i]);
  }
}
```


Å kopiere en tabell av referanser fører ikke til at objektene kopieres

```
String[] navnelisteKopi = new String[4];
for (int i = 0; i < navneliste.length; i++) {
  navnelisteKopi[i] = navneliste[i];
}
```



Forskjeller mellom tabeller av primitive datatyper og tabeller av referansetyper

- En tabell av en primitiv datatype:
  - Elementene i tabellen inneholder dataverdiene.
  - Dataverdiene kopieres dersom tabellen kopieres element for element.
  - Elementene kan sammenlignes ved å bruke sammenligningsoperatorene.
  - Elementene initieres til 0 (ev. false) dersom ikke andre verdier gis i deklareringen av tabellen.
- En tabell av en referansetype:
  - Elementene i tabellen inneholder ikke objektene, men referanser til objektene.
  - Dersom tabellen kopieres element for element, blir bare referansene kopiert, ikke objektene. Element med samme indeks i begge tabellene peker til det samme objektet.
  - Med unntak av lik (==) og ikke lik (!=) kan vi ikke bruke sammenligningsoperatorer på referanser. Operatorene lik og ikke lik sammenligner innholdet i referansene, ikke i objektene som referansene peker til.
  - Elementene initieres til null, dersom ikke andre verdier gis i deklareringen av tabellen. Dersom vi prøver å bruke et tabellelement som ikke refererer til noe objekt, kastes NullPointerException.

Gjør oppgavene 1-3 side 278-279 –varetest.java.

Kapittel 7.12

Oppgave 1 – 3 s 278

Oppgave 1
Kodebiten vil ved kjøring kaste to typer unntak:
• NullPointerException: Tabellen varene er en tabell av referanser. Disse referansene må settes til å peke til objekter, før vi kan sende meldinger til objektene. Med andre ord: Vi må skrive for eksempel:
varene[0] = new Vare("TV-dress", 100, 575.50);
før vi kan sende melding til dette objektet:
varene[1].settPris(320.50);
• ArrayIndexOutOfBoundException: Dette unntaket kastes når vi i siste setning refererer til tabellelement med indeks 3. Elementene i en tabell med størrelse 3 nummereres 0, 1 og 2.

Oppgave 2a navneliste[1] = new String("Anders"); eller navneliste[1] = "Anders";

Oppgave 2b navneliste[3] = etNavn;

Oppgave 2c
Med vanlig for-løkke:
```
int sumLengde = 0;
for (int i = 0; i < navneliste.length; i++) {
sumLengde += navneliste[i].length();
}
```
Løkken skrevet om som utvidet for-løkke:
```
int sumLengde = 0;
for (String navn : navneliste) {
sumLengde += navn.length();
}
System.out.println(sumLengde);
```


Oppgave 1 – 3 s 278

Oppgave 2d
Med vanlig for-løkke:
```
final char tegn = 'r';
int antTegn = 0;
for (int i = 0; i < navneliste.length; i++) {
  int indeks = navneliste[i].indexOf(tegn);
  while (indeks >= 0) {
    antTegn++;
    indeks = navneliste[i].indexOf(tegn, indeks + 1);
  }
}
```
Løkken skrevet om som utvidet for-løkke:
```
for (String navn : navneliste) {
  int indeks = navn.indexOf(tegn);
  while (indeks >= 0) {
    antTegn++;
    indeks = navn.indexOf(tegn, indeks + 1);
  }
}
```

Oppgave 3
```
public static void main(String[] args) {
for (int i = 0; i < args.length; i++) {
System.out.println(args[i]);
}
}
```


Tabell av referansetype som medlem i klasse

```
/*
 * TabellAvFag.java  E.L. 2004-02-22
 *
 * Filen inneholder tre klasser:
 * Fag: Beskriver et fag med fagkode, navn og antall studiepoeng, tilbyr finn-metoder.
 * Fagkatalog: Klassen inneholder en tabell av fagobjekter. Størrelsen på tabellen  settes i konstruktøren. Objektene opprettes etter hvert som nye fag legges
 * inn. Klienten kan sortere fagene og hente ut en oversikt ved å bruke toString().  Klienten kan også hente ut en referanse til hvert enkelt fagobjekt.
 * TabellAvFag: Testklient som leser inn data om fagene fra brukeren.
 */

import static javax.swing.JOptionPane.*;

class Fag {
  private String fagkode;  // entydig
  private String fagnavn;
  private int antSp;  // antall studiepoeng

  public Fag(String startFagkode, String startFagnavn, int startAntStudiepoeng) {
    fagkode = startFagkode;
    fagnavn = startFagnavn;
    antSp = startAntStudiepoeng;
  }

  public String finnFagkode() {
    return fagkode;
  }

  public String finnFagnavn() {
    return fagnavn;
  }
  public int finnAntStudiepoeng() {
    return antSp;
  }

  public String toString() {
    return "Kode: " + fagkode + ", Navn: " + fagnavn + ", " + antSp + " studiepoeng.";
  }
}
```


Tabell av referansetype som medlem i klasse

```
/*
 * Klassen Fagkatalog tilbyr få metoder.
 * Flere metoder, se oppgave 2 etter dette delkapitlet.
 */
class Fagkatalog {
  private Fag[] fagene;
  private int antFag;

  public Fagkatalog(int startMaksAntallFag) {
    fagene = new Fag[startMaksAntallFag];
  }

  /*
   * Registrerer nytt fag. Dersom ikke plass, returneres false.
   * Ingen kontroll av at fag med denne kode er registrert fra før.
   */
  public boolean registrerNyttFag(String startFagkode,
                                  String startFagnavn, int startAntStudiepoeng) {
    if (antFag < fagene.length) {
      fagene[antFag] = new Fag(startFagkode, startFagnavn, startAntStudiepoeng);
      antFag++;
      return true;
    }
    else return false;  // ikke plass i tabellen
  }

  public int finnAntallFag() {
    return antFag;
  }

  public Fag finnFag(int indeks) {
    if (indeks >= 0 && indeks < antFag) return fagene[indeks];
    else return null;
  }
```

```
  /*
   * Sorterer fag etter fagkode. Algoritme: Sortering ved utvelging.
   * Sorterer bare den delen av tabellen der vi har fornuftige data,
   * det vil si de første antFag elementene.
   */
  public void sorterFag() {
    for (int start = 0; start < antFag; start++) {
      int hittilMinst = start;
      for (int i = start + 1; i < antFag; i++) {
        String denneFagkode = fagene[i].finnFagkode();
        String hittilMinstFagkode = fagene[hittilMinst].finnFagkode();
        if (denneFagkode.compareToIgnoreCase(hittilMinstFagkode) < 0) hittilMinst = i;
      }
      Fag hjelp = fagene[hittilMinst];
      fagene[hittilMinst] = fagene[start];
      fagene[start] = hjelp;
    }
  }

  /*
   * Bygger opp en resultatstreng ved å sende toString()-meldingen til hvert
   * enkelt fagobjekt. Legger inn linjeskift mellom hvert fag.
   */
  public String toString() {
    String resultat = "";
    for (int i = 0; i < antFag; i++) {
      resultat += fagene[i].toString() + "\n";
    }
    return resultat;
  }
}
```

Tabell av referansetype som medlem i klasse

```
class TabellAvFag {

  public static void main(String[] args) {
    Fagkatalog katalogen = new Fagkatalog(10);  // maks 10 fag

    boolean ok = true;
    /* Ingen kontroll av at showInputDialog() returnerer fornuftige data */
    String kode = showInputDialog("Fagkode (avslutt med blank): ");
    while (ok && !kode.trim().equals("")) {
      String navn = showInputDialog("Fagnavn: ");
      String spLest = showInputDialog("Antall studiepoeng: ");
      int sp = Integer.parseInt(spLest.trim());

      ok = katalogen.registrerNyttFag(kode.trim(), navn.trim(), sp);
      if (ok) {
        showMessageDialog(null, "Registrering ok");
        kode = showInputDialog("Fagkode (avslutt med blank): ");
      } else showMessageDialog(null, "Ikke plass til flere fag, avslutter.");
    }

    /* Usortert liste, prøver finnAntFag() og finnFag() */
    String utskrift = "";
    for (int i = 0; i < katalogen.finnAntallFag(); i++) {
      utskrift += katalogen.finnFag(i).toString() + "\n";
    }
    showMessageDialog(null, "Usortert:\n" + utskrift);
    katalogen.sorterFag();

    /* Sortert liste, prøver toString() */
    showMessageDialog(null, "Sortert:\n" + katalogen.toString());
  }
}
```

```
/* Eksempeldata, ferdig sortert:
Kode: LV172D, Navn: Programmering i Java, 6 studiepoeng.
Kode: LV191D, Navn: Videregående programmering, 6 studiepoeng.
Kode: LV193D, Navn: Web-programmering med JSP, 6 studiepoeng.
Kode: LV195D, Navn: Objektorientert programmering i C++, 12 studiepoeng.
*/
```


Sortering av objekter

```
class Fagkatalog {
  private Fag[] fagene;
  private int antFag;
```

Sorterer fag-objektene etter fagkode.

```
class Fag {
  private String fagkode;  // entydig
  private String fagnavn;
  private int antSp;
```

```
public void sorterFag() {
  for (int start = 0; start < antFag; start++) {
    int hittilMinst = start;
    for (int i = start + 1; i < antFag; i++) {
      String denneFagkode = fagene[i].finnFagkode();
      String hittilMinstFagkode = fagene[hittilMinst].finnFagkode();
      if (denneFagkode.compareToIgnoreCase(hittilMinstFagkode) < 0) hittilMinst = i;
    }
    Fag hjelp = fagene[hittilMinst];
    fagene[hittilMinst] = fagene[start];
    fagene[start] = hjelp;
  }
 }
```

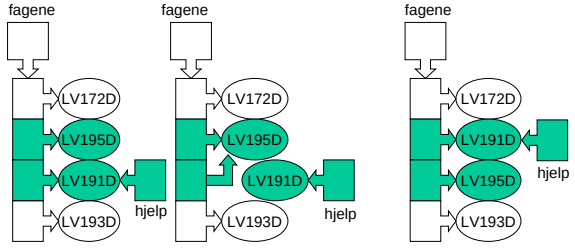
bytter om innholdet i referansene, objektene ligger i ro

Bytter om innholdet i to referanser

Trinn 1: *hjelp = fagene[hittilMinst]*

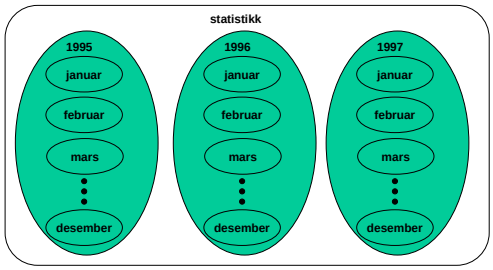
Trinn 2: *fagene[hittilMinst] = fagene[start]*

Trinn 3: *fagene[start] = hjelp*



Vi bytter om rekkefølgen på referansene. Objektene ligger i ro. At sirklene er tegnet under hverandre betyr ikke at de ligger etter hverandre i primærminnet.

Flerdimensjonale tabeller er lite brukt i objektorientert programmering

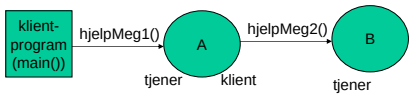


- Vi trenger å bruke flerdimensjonale tabeller dersom vi skal *behandle* dataene i flere dimensjoner.
- Ellers lager vi heller tabeller av objekter, der hvert objekt inneholder en tabell med tilleggsinformasjon knyttet direkte til tabellen.

Programmering i Java
http://www.tisip.no/boker/java/

Samarbeid mellom objekter

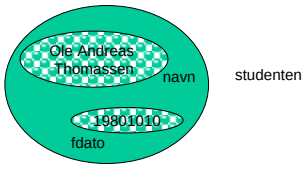
Objekter samarbeider ved å sende meldinger



Et og samme objekt kan spille både rollen klient og rollen tjener.

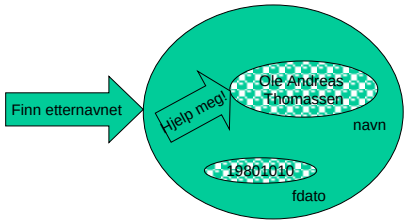
Et objekt er bygget opp av mindre deler

```
class Student {
  private String navn;
  private int fdato;  // ååååmmdd
```




```
class TestStudent1 {
  public static void main(String[] args) {
    Student studenten = new Student("Ole Andreas Thomassen", 19801010);
```


Objektet løser oppgaver ved å samarbeide med de delene det består av




```
class TestStudent1 {
  public static void main(String[] args) {
    Student studenten = …….
    String etternavn = studenten.finnEtternavn();
```

Samarbeid

- Et objekt samarbeider med et annet objekt ved å sende meldinger til det.
- For at det skal være mulig må det andre objektet være tilgjengelig.
- Enten
  - som objektvariabel (eller klassevariabel)
  - som parameter
  - som en lokal variabel

Vi deler navnet i to; fornavn og etternavn

```
class Student {
  private String fornavn;
  private String etternavn;
  private int fdato;
```

} lager en egen klasse for Navn

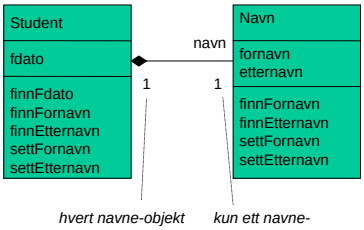
Vis klassen Navn, programliste 8.2, side 296-297.

Programliste 8.2

```
/*
 * Navn.java  E.L. 2002-07-22
 * Klassen er mutabel. Både fornavn og etternavn kan endres.
 */

class Navn {
  private String fornavn;
  private String etternavn;

  public Navn(String startFornavn, String startEtternavn) {
    fornavn = startFornavn;
    etternavn = startEtternavn;
  }

  public void settFornavn(String nyttFornavn) {
    fornavn = nyttFornavn;
  }

  public void settEtternavn(String nyttEtternavn) {
    etternavn = nyttEtternavn;
  }

  public String finnFornavn() {
    return fornavn;
  }

  public String finnEtternavn() {
    return etternavn;
  }

  public String toString() {
    return fornavn + " " + etternavn;
  }
}
```


Komposisjon, en-del-av-sammenheng

Et studentobjekt består av en fødselsdato og et navne-objekt



hvert navne-objekt er med i kun ett student-objekt

kun ett navne-objekt pr student-objekt

Vis program-liste 8.3, side 298-299.

Kompositten har det fulle og hele ansvaret for de delene den består av. Delene eksisterer ikke uten at kompositten eksisterer.

Programliste 8.3

```
/*
 * TestStudent2.java    E.L. 2004-03-20
 *
 * Filen inneholder to klasser: Student med attributt navn
og fdato. Klassen Navn brukes kun internt i denne klassen.
Klienter forholder seg til strenger og primitive datatyper.
 *
 * TestStudent2: Enkel testklient.
 */

import static javax.swing.JOptionPane.*;
class Student {
  private Navn navn;
  private int fdato;

  public Student(String startFornavn, String startEtternavn,
int startFdato) {
    navn = new Navn(startFornavn, startEtternavn);
    fdato = startFdato;
  }

  public int finnFdato() {
    return fdato;
  }
}

  public String finnFornavn() {
    return navn.finnFornavn();
  }

  public String finnEtternavn() {
    return navn.finnEtternavn();
  }

  public void settFornavn(String nyttFornavn) {
    navn.settFornavn(nyttFornavn);
  }

  public void settEtternavn(String nyttEtternavn) {
    navn.settEtternavn(nyttEtternavn);
  }

  public String toString() {
    return navn + ", født: " + fdato;
  }
}
```


Programliste 8.3

```
class TestStudent2 {
  public static void main(String[] args) {
    Student studenten = new Student("Ole Andreas", "Thomassen", 19801010);
    showMessageDialog(null, "Studenten heter " + studenten.finnFornavn() +
                                  " " + studenten.finnEtternavn());
    String nyttFornavn = showInputDialog("Oppgi nytt fornavn: ");
    String nyttEtternavn = showInputDialog("Oppgi nytt etternavn: ");
    studenten.settFornavn(nyttFornavn);
    studenten.settEtternavn(nyttEtternavn);
    showMessageDialog(null, "Studenten heter nå " + studenten.toString());
  }
}
```


Sekvensdiagram – studenten.finnEtternavn()