Løsning på småoppgaver etter hvert underkapittel. kap. 19-22

Kun til bruk sammen med læreboka "Programmering i Java", Else Lervik og Vegard B. Havdal. Stiftelsen TISIP og Gyldendal Akademisk.

Tilpasset 3.utgave av boka.

Kapittel 19.1

Oppgave 1

Endringer i tjenerprogrammet:

```
.....
/* Sender innledning til klienten */
skriveren.println("Hei, du har kontakt med tjenersiden!");
skriveren.println("Skriv tall, et på hver linje, avslutt med 'x'.");

/* Mottar data fra klienten */
int sum = 0;
String enLinje = leseren.readLine().trim();
while (!enLinje.equals("x")) {  // siste linje er lik 'x'
    int tall = Integer.parseInt(enLinje);   // omformer teksten til et tall, ingen
feilbehandling
  sum += tall;
  System.out.println("Har lagt til " + tall);
  enLinje = leseren.readLine();
}
System.out.println("Nå er det slutt, summen ble " + sum);
skriveren.println("Summen av tallene er " + sum); // sender summen til klienten

/* Lukker forbindelsen */
 ...

/* Utskrift på tjenersiden:
Logg for tjenersiden. Nå venter vi...
Har lagt til 3
Har lagt til 4
Har lagt til 5
Nå er det slutt, summen ble 12
*/
```

Endringer i klientprogrammet:

```
...
/* Leser innledning fra tjeneren og skriver den til konsollet */
String innledning1 = leseren.readLine();
String innledning2 = leseren.readLine();
System.out.println(innledning1 + "\n" + innledning2);

/* Leser tekst fra konsoll (brukeren) */
String tekst = leserFraKonsoll.readLine().trim();
while (!tekst.equals("x")) {  // avslutter tallrekken med x
  skriveren.println(tekst);  // sender teksten til tjeneren, tjeneren omformer til tall
```

```
  tekst = leserFraKonsoll.readLine();
}
skriveren.println(tekst);  // sender avslutningstegnet
System.out.println(leseren.readLine()); // henter summen fra tjeneren

/* Lukker forbindelsen */
....

/* Utskrift på klientsiden:
Oppgi navnet på maskinen der tjenerprogrammet kjører: localhost
Nå er forbindelsen opprettet.
Hei, du har kontakt med tjenersiden!
Skriv tall, et på hver linje, avslutt med 'x'.
3
4
5
x
Summen av tallene er 12
*/
```

Kapittel 19.2

Oppgave 1

Vi legger inn følgende setninger etter den eksisterende rebind()-setningen:

```
System.out.println("Vi lager et til");
tellemaskin = new JaNeiTellerImpl();
System.out.println("Nå er det laget!");
String objektnavn2 = "AS_Tellefirma";
Naming.rebind(objektnavn2, tellemaskin);
```

Og helt til slutt:

```
Naming.unbind(objektnavn2);
```

Oppgave 2

```
import java.rmi.Naming;
import javax.swing.*;
import static javax.swing.JOptionPane.*;
import java.awt.*;
import java.awt.event.*;

class TellemaskinVindu extends JFrame {
  private JList liste;
  private JButton jaknapp = new JButton("Ja-stemmer");
  private JButton neiknapp = new JButton("Nei-stemmmer");
  private JaNeiTeller[] tellemaskiner;
  private JLabel status1 = new JLabel("Status");
  private JLabel status2 = new JLabel("");
  private String[] navnene;  // navnene på tellemaskin-objektene

  public TellemaskinVindu () {
```

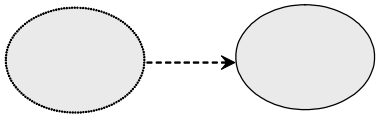
```
    setTitle("Registrerer ja- og nei-stemmer");
    try {
      navnene = Naming.list("//localhost/");
      tellemaskiner = new JaNeiTeller[navnene.length];
      for (int i = 0; i < navnene.length; i++) {
        tellemaskiner[i] = (JaNeiTeller) Naming.lookup(navnene[i]);
      }
    } catch (Exception e) {
      System.out.println("Feil oppstått: " + e);
    }

    liste = new JList(navnene);
    liste.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    liste.setSelectedIndex(0);

    setLayout(new BorderLayout(10,10));
    add(new JLabel("    Velg tellemaskin: "), BorderLayout.NORTH);
    add(liste, BorderLayout.CENTER);
    add(new Søndrepanel(), BorderLayout.SOUTH);
    add(new JLabel("    "), BorderLayout.EAST);
    add(new JLabel("    "), BorderLayout.WEST);
  }

  private class Søndrepanel extends JPanel {
    public Søndrepanel() {
      setLayout(new BorderLayout());
      add(new Knappepanel(), BorderLayout.NORTH);
      add(status1, BorderLayout.CENTER);
      add(status2, BorderLayout.SOUTH);
    }
  }

  private class Knappepanel extends JPanel {
    public Knappepanel() {
      add(jaknapp);
      add(neiknapp);
      Knappelytter lytter = new Knappelytter();
      jaknapp.addActionListener(lytter);
      neiknapp.addActionListener(lytter);
    }
  }

  private class Knappelytter implements ActionListener {
    public void actionPerformed(ActionEvent hendelse) {
      int antall = 0;
      try {
        int indeks = liste.getSelectedIndex();
        JButton kilde = (JButton) hendelse.getSource();
        if (kilde == jaknapp) {
          String svar = showInputDialog(null, "Antall ja-stemmer: ");
          antall = Integer.parseInt(svar);
```

```
            tellemaskiner[indeks].økAntallJa(antall);
          }
          else if (kilde == neiknapp) {
            String svar = showInputDialog(null, "Antall nei-stemmer: ");
            antall = Integer.parseInt(svar);
            tellemaskiner[indeks].økAntallNei(antall);
          }
          status1.setText("    Status: " + navnene[indeks]);
          status2.setText("    Antall ja: " + tellemaskiner[indeks].finnAntallJa()
              + ", antall nei: " + tellemaskiner[indeks].finnAntallNei());
        }
        catch (NumberFormatException e) {  // antall stemmer blir i dette tilfellet 0
        }
        catch (Exception e) {
          System.out.println("Feil oppstått: " + e);
        }
      }
    }
  }

  class Oppg19_2_2 {
    public static void main(String[] args) {
      TellemaskinVindu vindu = new TellemaskinVindu();
      vindu.setSize(300, 200);
      vindu.setLocation(400, 300);
      vindu.setVisible(true);
    }
  }
```

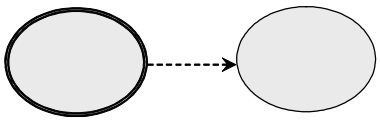
Kapittel 19.3

Oppgave 1

Vi bruker følgende notasjon



Stedfortrederobjekt med pil til det ekte objektet

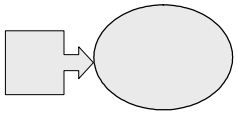


Kopi (serialisert objekt) med pil til originalen.
Etter at kopieringen er utført eksisterer det ingen forbindelse mellom de to objektene. Da tegnes de som to separate objekter uten pil mellom.

Først utføres de tre setningene i tjenerprogrammet:

```
Medlemsfabrikk fabrikkobjekt = new MedlemsfabrikkImpl();  // lager objektet
 Naming.rebind("Medlemsfabrikk", fabrikkobjekt);  // registrerer det i rmi-registeret
System.out.println("Nå venter vi bare på at noen skal lage medlemsobjekter...");
```

Da har vi ett objekt på tjenersiden:



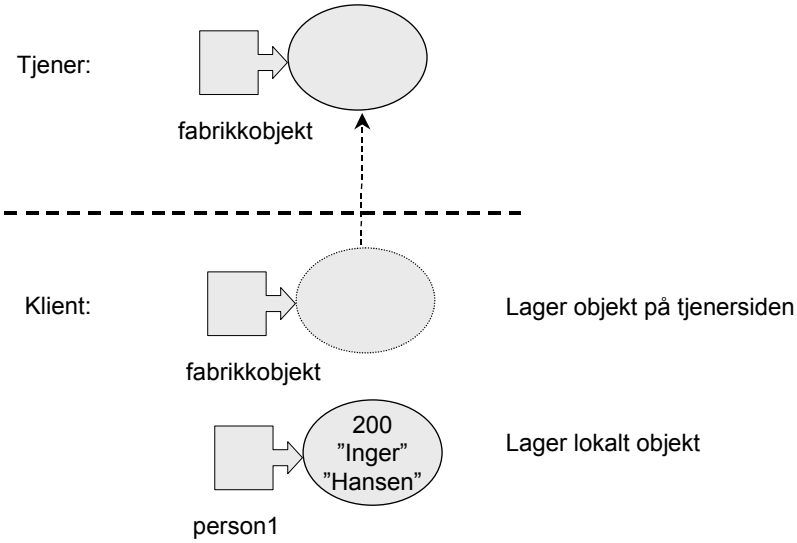
fabrikkobjekt

Vi nummererer setningene på klientsiden:

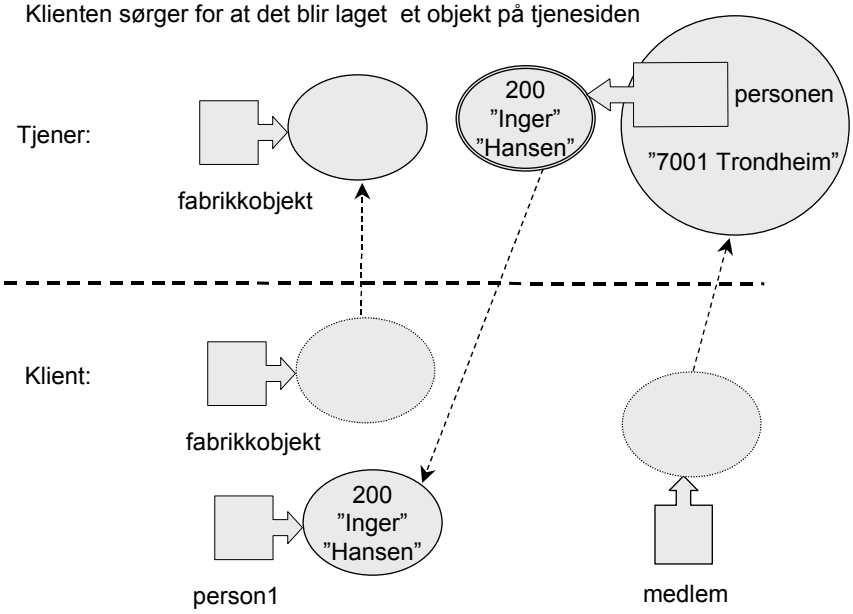
```
1   Medlemsfabrikk fabrikkobjekt =
                (Medlemsfabrikk) Naming.lookup(url);
2  Person person1 = new Person(200, "Inger", "Hansen");

   /* Lager (fjern-)objekt på tjenersiden */
3   Medlem medlem = fabrikkobjekt.lagMedlem(person1, "7001 Trondheim");

4   person1.settEtternavn("Olsen");  // endrer ikke data i fjernobjektet
5   Person person2 = medlem.finnPerson();
6   System.out.println(person2); // bruker toString()
7   person2.settEtternavn("Jensen"); // endrer ikke data i fjernobjektet
8   medlem.settPerson(person2);  // nå endrer vi data i fjernobjektet
9   Person person3 = medlem.finnPerson();
```

Vi får følgende prosess:

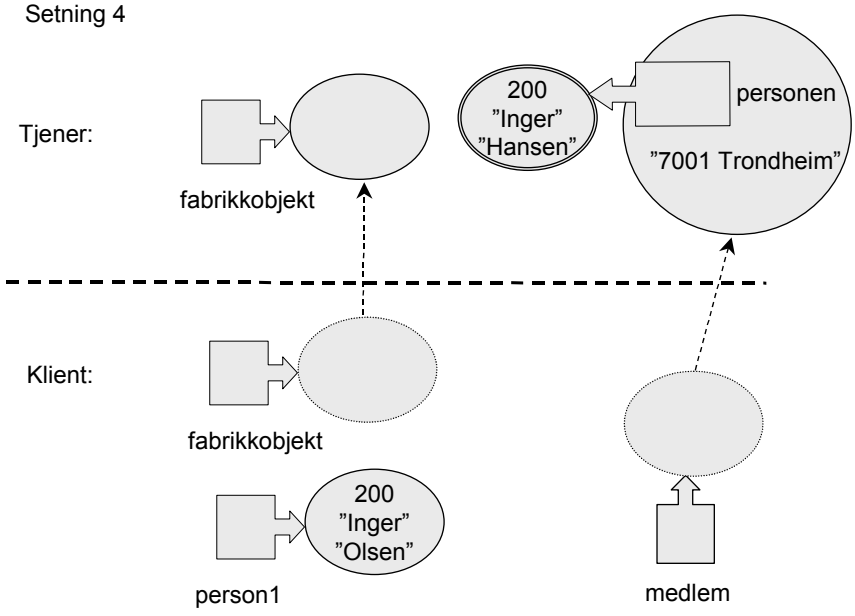
Etter setning 1 og 2:



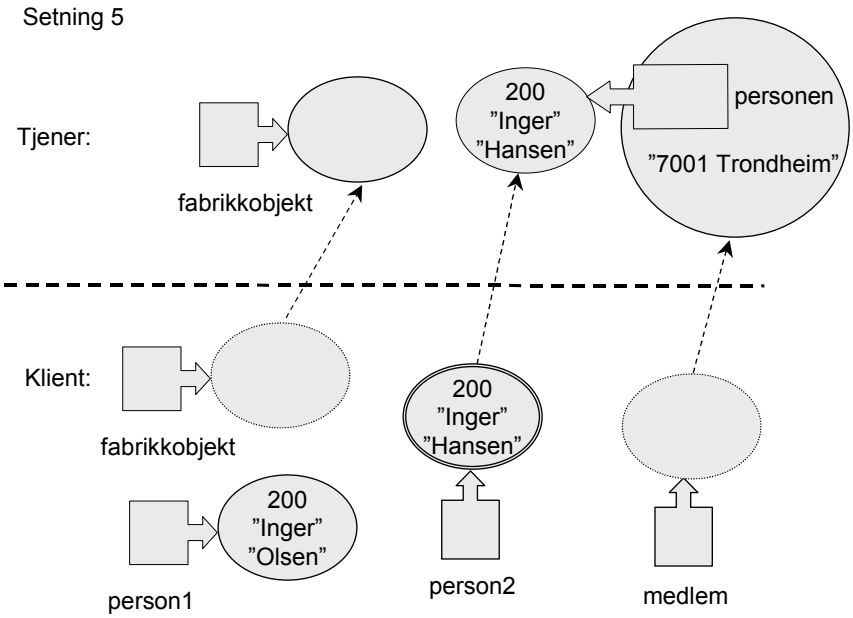
Setning 3:
Klienten sørger for at det blir laget et objekt på tjenesiden



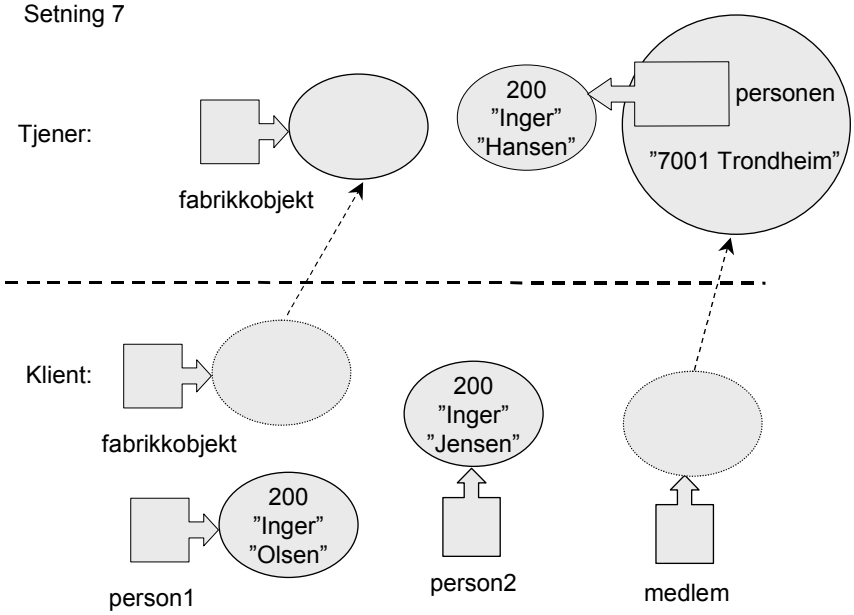
Setning 4



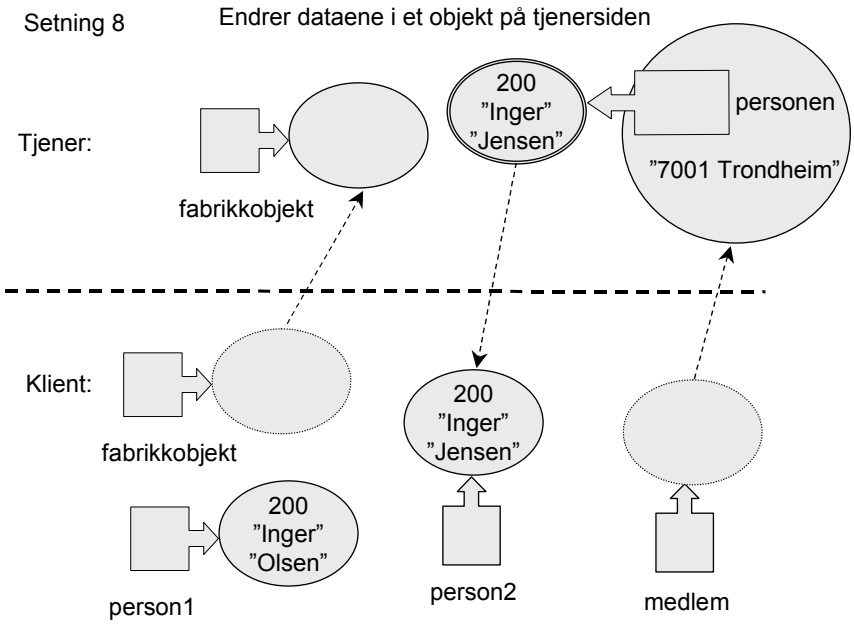
Endrer dataene i objektet på klientsiden

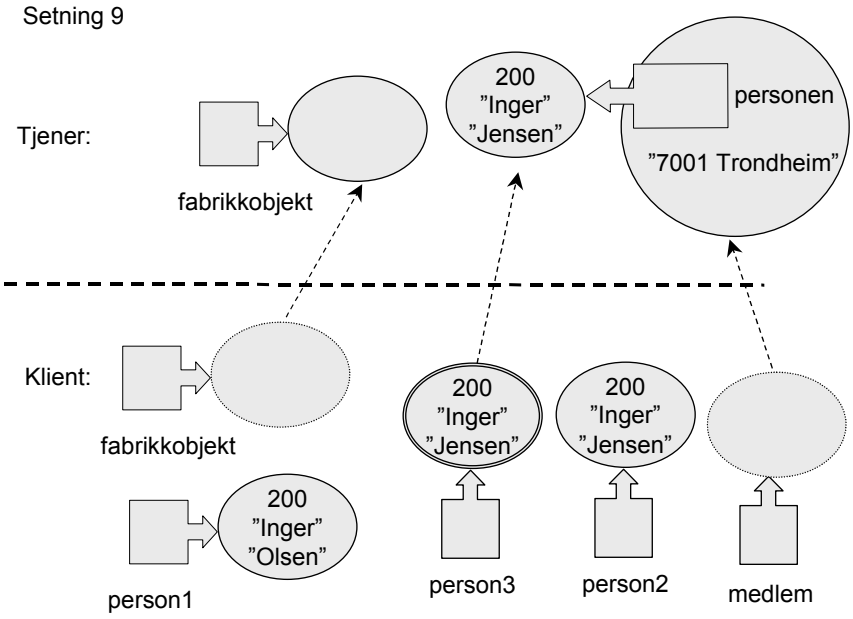


Henter en kopi av et personobjekt fra tjenersiden.



Endrer dataene i et objekt på klientsiden





Henter en kopi av et objekt fra tjenersiden.

Oppgave 2

Interfacene forsvinner. Klassen MedlemsfabrikkImpl får navnet Medlemsfabrikk. Den ser slik ut:

```
class Medlemsfabrikk {
  public Medlem lagMedlem(Person startPerson, String startAdresse) {
    return new Medlem(startPerson, startAdresse);
  }
}
```

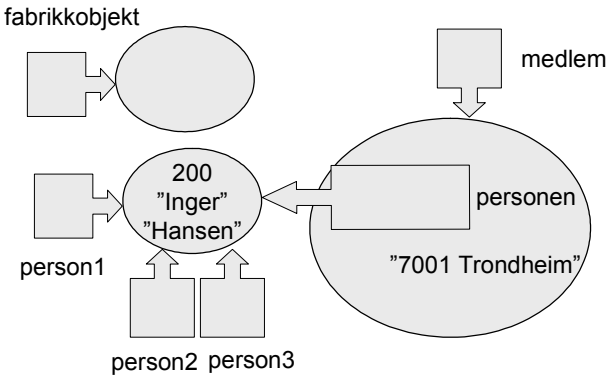
Klassen MedlemImpl får navnet Medlem. De første linjene i klassen ser slik ut:

```
class Medlem {
  private Person personen;
  private String adressen;
  public Medlem(Person startPerson, String startAdresse) {
    personen = startPerson;
    adressen = startAdresse;
  }

  public Person finnPerson()  {
    ....
```

Hovedprogrammet ser slik ut:

```
class Oppg19_3_2 {
  public static void main(String[] args)  {
    Medlemsfabrikk fabrikkobjekt = new Medlemsfabrikk();
    Person person1 = new Person(200, "Inger", "Hansen");
    Medlem medlem = fabrikkobjekt.lagMedlem(person1, "7001 Trondheim");
    person1.settEtternavn("Olsen");
    Person person2 = medlem.finnPerson();
    System.out.println(person2);
    person2.settEtternavn("Jensen");
    medlem.settPerson(person2);
    Person person3 = medlem.finnPerson();
    System.out.println(person3);
  }
}
```

Figuren viser de referansene og de objektene som vi har før vi forandrer på noe:



Vi har nå kun ett personobjekt. person1 er den første referansen som settes til å peke til dette objektet. Når vi i neste omgang lager et medlemsobjekt, får vi en referanse inne fra dette medlemsobjektet som også peker til det ene personobjektet. person2 og person3 vil dermed også peke til personobjektet.

Begge meldingene settEtternavn() endrer på innholdet i det ene personobjektet vi har. Først endres "Hansen" til "Olsen", deretter til "Jensen". Utskriften ser slik ut:

```
Inger Olsen
Inger Jensen
```

Oppgave 3

Løsningen bør ses i sammenheng med løsningene til oppgave 1 og 2.

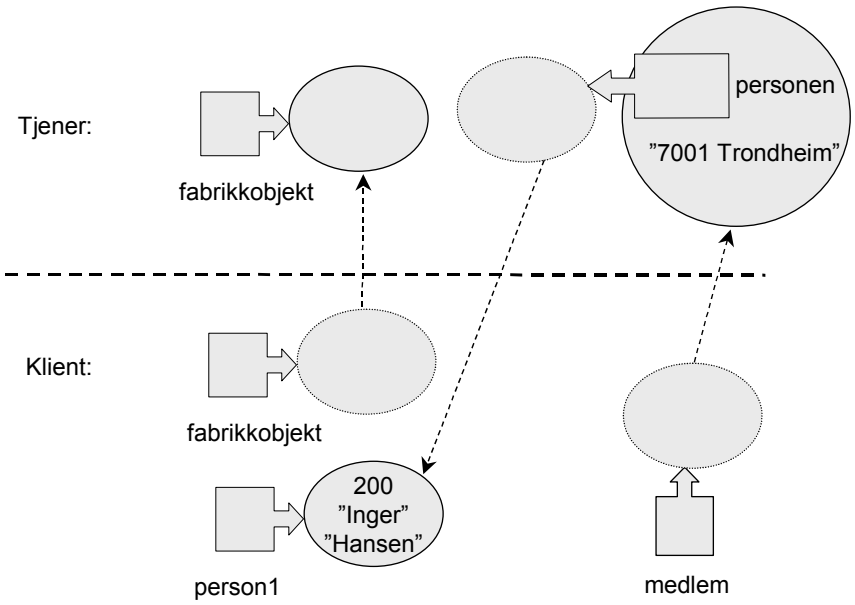
De personobjektene vi lager blir fjernobjekter. Det vil si at de "bor" i den Java-tolkeren der de ble laget. Andre Java-tolkere har tilgang til dem via stedfortrederobjekter.

I dette eksemplet lages det kun ett personobjekt. Det skjer i klientprogrammet, det vil dermed "leve" i den Java-tolkeren som kjører klientprogrammet.

Klientprogrammet ser nå slik ut:

```
Medlemsfabrikk fabrikkobjekt = (Medlemsfabrikk) Naming.lookup(url);
Person person1 = new PersonImpl(200, "Inger", "Hansen");
Medlem medlem = fabrikkobjekt.lagMedlem(person1, "7001 Trondheim");
   ... osv, som før ...
```

Etter at disse setningene er utført, ser ting slik ut:



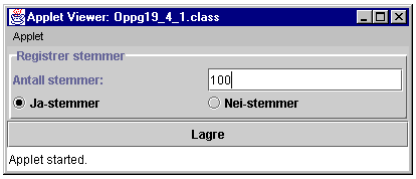
Sirklene med stiplet omkrets viser stedfortrederobjekter. Pilene peker til de virkelige objektene. Situasjonen er den samme som i oppgave 2, på den måten at vi kun har ett personobjekt. Alle endringer skjer i dette objektet. Utskriften blir derfor som i oppgave 2.

Hvorfor er det risikabelt å la personobjektene være fjernobjekter? Jo, fordi de lever på klientsiden, og en klient kan kople seg fra. Dermed forsvinner objekter som tjenersiden er avhengig av. Ja, det kan også tenkes at andre klienter har referanser til disse objektene. De kan for eksempel ha bedt om dem fra et objekt på tjenersiden.

Kapittel 19.4

Oppgave 1

Brukergrensesnittet ser slik ut:



Kildekoden er som følger:

```
public class Oppg19_4_1 extends JApplet {
  private JTextField antall = new JTextField(8);
  private JRadioButton jaKnapp = new JRadioButton("Ja-stemmer", true);
  private JRadioButton neiKnapp = new JRadioButton("Nei-stemmer", false);
  private JButton lagreknapp = new JButton("Lagre");
  private JaNeiTeller tellemaskin;

  public void init() {
    try {
      tellemaskin = (JaNeiTeller) Naming.lookup("rmi://localhost/AS_Tellebyrå");
      add(new JLabel(), BorderLayout.NORTH);
      add(new InputPanel(), BorderLayout.CENTER);
      add(lagreknapp, BorderLayout.SOUTH);

      Knappelytter knappelytter = new Knappelytter();
      lagreknapp.addActionListener(knappelytter);

      antall.requestFocus();
    } catch (Exception e) {
      System.out.println("Feil i TelleApplet: " + e);
    }
  }

  private class InputPanel extends JPanel {
    public InputPanel() {
      setLayout(new GridLayout(2, 2));
      add(new JLabel("Antall stemmer: "));
```

```
        add(antall);
        ButtonGroup gruppe = new ButtonGroup();
        gruppe.add(jaKnapp);
        gruppe.add(neiKnapp);
        add(jaKnapp);
        add(neiKnapp);
        SoftBevelBorder ramme = new SoftBevelBorder(BevelBorder.RAISED);
        Border boks = BorderFactory.createTitledBorder(ramme, "Registrer stemmer");
        setBorder(boks);
      }
    }

    private class Knappelytter implements ActionListener {
      public void actionPerformed(ActionEvent hendelse) {
        int antallStemmer = 0;
        try {
          antallStemmer = Integer.parseInt(antall.getText());
        } catch (NumberFormatException e) {
          JOptionPane.showMessageDialog(null, "Ugyldig tallformat.");
          antall.requestFocus();
        }
        try {
          if (jaKnapp.isSelected()) tellemaskin.økAntallJa(antallStemmer);
          else tellemaskin.økAntallNei(antallStemmer);
        } catch (Exception e) {
          System.out.println("Feil oppstått i lytteren til lagreknappen: " + e);
        }
        antall.setText("");
        antall.requestFocus();
      }
    }
  }
```

Du trenger en *html*-fil som kaller opp appleten. Her er den som verktøyet TextPad genererer automatisk:

```
<!DOCTYPE HTML><HTML><HEAD></HEAD><BODY>
<APPLET CODE="Oppg19_4_1.class" CODEBASE="."
  WIDTH=400 HEIGHT=300></APPLET>
</BODY></HTML>
```

For å kunne kjøre appleten i appletviewer (se vedlegg A2) må du først starte følgende to program:

- rmi-registeret
- tjenerprogrammet TellerTjener fra programliste 19.3

Når du nå har fått appleten til å kjøre i appletviewer, er en vanlig nettleser neste skritt. Da må vi kjøre webtjener. Den må kjøre på samme maskin som rmi-registeret. Årsaken til det er at en applet kun kan kommunisere med program på den maskinen den ble lastet ned fra.

I kapittel 21 lærer du hvordan du bruker webtjeneren Tomcat. Du kan imidlertid bruke en hvilken som helst webtjener, bare den kjører på samme maskin som rmiregisteret og tjenerprogrammet (TellerTjener).

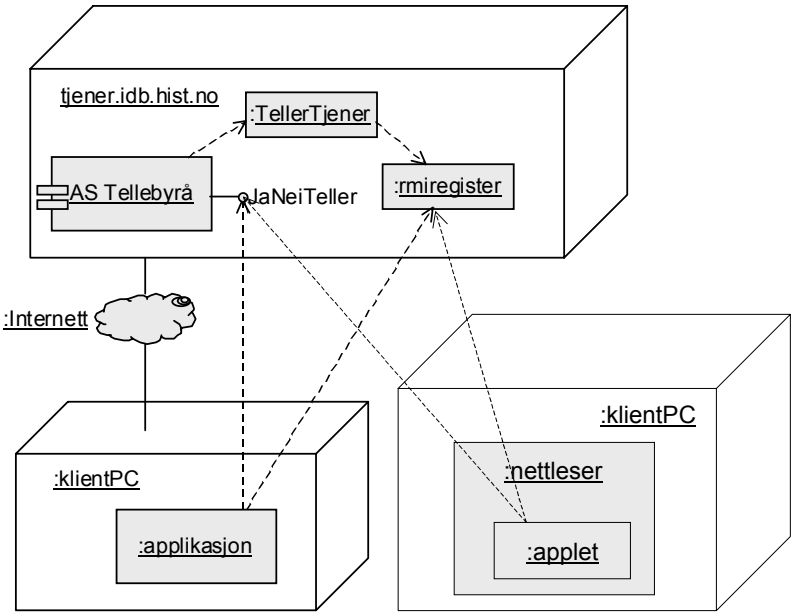
Før du kan laste ned appleten i en nettleser må altså tre programmer kjøre på den maskinen der du henter appleten fra:

- webtjener
- rmi-register
- tjenerprogrammet TellerTjener fra programliste 19.3

Med Tomcat kjørende på port 9090, vil URL'en se f.eks. slik ut:
http://156.76.50.237:9090/examples/JaNeiTeller/Oppg19_4_1.html

Kapittel 19.5

Oppgave 1



Kapittel 19.6

1.opplag av boka: Vi bruker den versjonen som trykkfeil-listen har lenke til under kapittel 19.6 som utgangspunkt.

Videre endrer vi implementasjonen av skrivStatus( ) i klasen KlientImpl slik at den er uavhengig av det grafiske vindet:

```
public synchronized void skrivStatus(String[] status) throws RemoteException {
  for (String linje : status) System.out.print(linje + ", ");
  System.out.println();
}
```

Dette gjør det mulig å lage enkle kleintprogrammer med utskrift til konsollet.

Oppgave 1

Vi utvider interfacet TellemaskinFront med følgende metode:

```
String[] navnPåAlleKlienter() throws RemoteException;
```

Implementasjonen i klassen TellemaskinFrontImpl:

```
public synchronized String[] navnPåAlleKlienter() throws RemoteException {
  String[] navnene = new String[klientene.size()];
  for (String enKlient : alleKlienter) System.out.println(enKlient);
  return navnene;
}
```

Testing: Tjenerprogrammet kan være som i programliste 19.6. På klientsiden tester vi dette med følgende kode:

```
tellemaskin.registrerMeg(denneKlienten);
String[] alleKlienter = tellemaskin.navnPåAlleKlienter();
System.out.println("Nå er " + alleKlienter.length + " klienter koplet til:");
for (int i = 0; i < alleKlienter.length; i++) System.out.println(alleKlienter[i]);
```

Oppgave 2

Vi utvider interfacet TellemaskinFront med følgende metode:

```
boolean sendMelding(Klient fra, String tilNavn, String melding)
                                        throws RemoteException;
```

Implementasjonen ser slik ut:

```
public synchronized boolean sendMelding(Klient fra, String tilNavn,
  String melding) throws RemoteException {
  boolean funnet = false;
  int tilIndeks = 0;
  Klient tilKlient = null;
  while (tilIndeks < klientene.size() && !funnet) {
    tilKlient = klientene.get(tilIndeks);
    if (tilKlient.finnNavn().equals(tilNavn)) funnet = true;
    else tilIndeks++;
  }
  if (funnet) {
    tilKlient.mottaMelding(fra.finnNavn(), melding);
    return true;
  }
  else return false;
}
```

Vi har en metode på klientsiden, skrivStatus(), som i prinsippet gjør det vi ønsker. Vi velger imidlertid å lage en ny metode, uavhengig av dialogvinduer, som tar navnet på avsenderen og meldingen som argument, og skriver til konsollet. Interfacet Klient utvides med følgende:

```
void mottaMelding(String navn, String melding) throws RemoteException;
```

Implementasjonen ser slik ut:

```
public synchronized void mottaMelding(String navn, String melding)
                                        throws RemoteException {
  System.out.println("Melding fra " + navn + ": " + melding);
```

```
}
```

Vi tester dette med en løkke i klientprogrammet:

```
boolean flere = true;
do {
 String tilNavn = showInputDialog(null, "Hei " + klientnavn + ", hvem skal du sende
melding til?");
 String melding = showInputDialog(null, "Skriv meldingen:");
 String tekst;
 if (tellemaskin.sendMelding(denneKlienten, tilNavn, melding)) {
  tekst = "Hei " + klientnavn + ", melding sendt. ";
 }
 else tekst = "Hei " + klientnavn + ", melding ikke sendt. ";
 int svar = showConfirmDialog(null, tekst +
       "Flere meldinger?", "Sender meldinger", YES_NO_OPTION);
 if (svar == NO_OPTION) flere = false;
} while (flere);
```

Oppgave 3

For å kunne sende meldinger direkte fra en klient til en annen, må klientene vedlikeholde en liste over stedfortrederobjektene til andre påloggede klienter. Denne listen administreres fra den sentrale tjeneren, det vil si at vi legger kode inn i metodene registrerMeg( ) og meldMegUt( ).

Revidert utgave av registrerMeg( ):

```
public synchronized void registrerMeg(Klient klienten) throws RemoteException {
 try {
   // Oppgave 3 - oppdaterer liste hos alle klienter
   int klientIndeks = 0;
   while (klientIndeks < klientene.size()) {
     Klient enKlient = null;
      try {
       enKlient = klientene.get(klientIndeks);
       enKlient.leggTilNyKlient(klienten);
       klientIndeks++;
      } catch (ConnectException e) { // klienten er koplet ned
       System.out.println("Nå er klienten " + enKlient.finnNavn() + " fjernet.");
       klientene.remove(klientIndeks);
      }
    }

    // Oppdaterer listen hos den nye klienten
    for (Klient enKlient : klientene) klienten.leggTilNyKlient(enKlient);

    // Oppgave 3, slutt
  klientene.add(klienten);
  System.out.println("Nå er " + klienten.finnNavn() + " registrert.");
  klienten.skrivStatus(lagMelding());
 } catch (Exception e) {
  System.out.println("Feil oppstått i registrerMeg(): " + e);
  e.printStackTrace();
 }
```

```
    }
```

Revidert utgave av meldMegUt( ):

```
    public synchronized void meldMegUt(Klient klienten) throws RemoteException {
      boolean funnet = false;
      int klientIndeks = 0;
      while (klientIndeks < klientene.size() && !funnet) {
        Klient denne = klientene.get(klientIndeks);
        if (denne.equals(klienten)) {  // sammenlikner stedfortrederobjektene
          funnet = true;
          klientene.remove(klientIndeks);

          // Oppgave 3 - oppdaterer liste hos alle klienter
          int klientIndeks2 = 0;
          while (klientIndeks2 < klientene.size()) {
            Klient enKlient = null;
            try {
              enKlient = klientene.get(klientIndeks2);
              enKlient.fjernKlient(denne);
              klientIndeks2++;
            } catch (ConnectException e) { // klienten er koplet ned
              System.out.println("Nå er klienten " + enKlient.finnNavn() + " fjernet.");
              klientene.remove(klientIndeks2);
            }
          }
          // Oppgave 3, slutt
          System.out.println("Nå er klienten " + klienten.finnNavn() + " fjernet.");
        } else klientIndeks++;
      }
    }
```

Vi ser at dette krever nye metoder i interfacet Klient:

```
    void leggTilNyKlient(Klient nyKlient) throws RemoteException;
    void fjernKlient(Klient klienten) throws RemoteException;
```

Ny objektvariabel i klassen KlientImpl:

```
    private ArrayList<Klient> andreKlienter = new ArrayList<Klient>();
```

Implementasjonen av metodene ser slik ut:

```
    public synchronized
              void leggTilNyKlient(Klient nyKlient) throws RemoteException {
      andreKlienter.add(nyKlient);
    }
    public synchronized
              void fjernKlient(Klient klienten) throws RemoteException {
      andreKlienter.remove(klienten);
    }
```

Vi trenger, som i opppgave 2, en metode for å skrive ut meldingen som kommer, denne gangen direkte fra en annen klient:

```
    void skrivMelding(String fraNavn, String melding) throws RemoteException;
```

Implementasjon:

```
public synchronized
        void skrivMelding(String fraNavn, String melding) throws RemoteException {
  System.out.println("Melding fra " + fraNavn + ": " + melding);
}
```

Til hjelp for klientprogrammet er det laget to metoder i KlientImpl som ikke er med i interfacet:

```
public synchronized void skrivKlientliste() throws RemoteException {
  if (andreKlienter.size() > 0) {
     System.out.println("Klientlisten ser nå slik ut:\n");
     for (Klient enKlient : andreKlienter) System.out.println(enKlient.finnNavn());
   } else System.out.println("Ingen andre pålogget");
}

public boolean
          sendMelding(String melding, String tilNavn) throws RemoteException {
  // finner klienten
  for (Klient enKlient : andreKlienter) {
    if (enKlient.finnNavn().equals(tilNavn)) {
      enKlient.skrivMelding(navn, melding);
      return true;
    }
  }
  return false; // klient ikke funnet
}
```

Klientprogrammet er laget som et enkelt testprogram der vi går i løkke og lar klienten sende meldinger til de andre klientene. For hver ny melding skrives liste over påloggede klienter ut:

```
TellemaskinFront tellemaskin = (TellemaskinFront) Naming.lookup(url);
String klientnavn = showInputDialog(null, "Oppgi klientnavn:");
System.out.println("Her er konsollvinduet til " + klientnavn);

/* Trenger å ha KlientImpl som type, pga at lokale metoder brukes */
KlientImpl denneKlienten = new KlientImpl(klientnavn);
 tellemaskin.registrerMeg(denneKlienten);

boolean flere = true;
do {
  denneKlienten.skrivKlientliste();
  String tilKlient = showInputDialog(null,
                  "Hei " + klientnavn + ", hvem skal du sende melding til?");
  String melding = showInputDialog(null, "Skriv meldingen:");

  String tekst;
  if (denneKlienten.sendMelding(melding, tilKlient)) {
    tekst = "Hei " + klientnavn + ", melding sendt. ";
  }
  else tekst = "Hei " + klientnavn + ", melding ikke sendt. ";
  int svar = showConfirmDialog(null, tekst +
        "Flere meldinger?", "Sender meldinger", YES_NO_OPTION);
```

```
    if (svar == NO_OPTION) flere = false;
  } while (flere);
  tellemaskin.meldMegUt(denneKlienten);
  System.exit(0);
```

Kapittel 20.2

Oppgave 1

```
    ResultSet res = setning.executeQuery(
          "select persnr, fornavn from person where etternavn = 'HANSEN'");
    System.out.println("Alle som heter Hansen til etternavn: ");
    while (res.next()) { // Henter ut persnr og fornavn, ikke etternavn
      int persNr = res.getInt("persnr");
      String fornavn = res.getString("fornavn");
      System.out.println(persNr + ": " + fornavn);
    }
```

Oppgave 2

```
  Statement setning = forbindelse.createStatement();
  String søkenavn = showInputDialog(null, "Oppgi etternavn, avslutt med Cancel: ");
  while (søkenavn != null) {
    søkenavn = søkenavn.trim().toUpperCase();
    ResultSet res = setning.executeQuery
      ("select persnr, fornavn from person where etternavn = '" + søkenavn + "'");
    System.out.println("Alle som heter " + søkenavn + " til etternavn: ");
    while (res.next()) {
      int persNr = res.getInt("persnr");
      String fornavn = res.getString("fornavn");
      System.out.println(persNr + ": " + fornavn);
    }
    res.close();
    søkenavn =  showInputDialog(null, "Oppgi etternavn, avslutt med Cancel: ");
  }
  setning.close();
```

Kapittel 20.3

Oppgave 1

Metoden ser slik ut:

```
  public ArrayList<String> finnAlleForskjelligeEtternavn() {
    ArrayList<String> alle = new ArrayList<String>();
    String sqlsetning = "select distinct etternavn from person";
    System.out.println(sqlsetning);
    Statement setning = null;
    ResultSet res = null;
    try {
      setning = forbindelse.createStatement();
      res = setning.executeQuery(sqlsetning);
      while (res.next()) {
        String etternavn = res.getString("etternavn");
        alle.add(etternavn);
      }
```

```
    } catch (SQLException e) {
      skrivMelding(e, "finnAlleForskjelligeEtternavn()");
      alle = null;
    } finally {
      lukkResSet(res);
      lukkSetning(setning);
      return alle;
    }
  }
```

Testprogrammet kan se slik ut:

```
  public static void main(String[] args) throws Exception {
    String dbDriver = "oracle.jdbc.driver.OracleDriver";
    String dbNavn = "jdbc:oracle:thin:@oracle.stud.aitel.hist.no:1521:oracle";
    String brukernavn = showInputDialog("Brukernavn: ");
    String passord = showInputDialog("Passord: ");
    int oracleFeilkode = 1; // hvis duplikate primærnøkler

    Database db = new Database(dbDriver, dbNavn, brukernavn,
                                          passord, oracleFeilkode);
    System.out.println("Finner alle forskjellige etternavn:");
    ArrayList<String> alle = db.finnAlleForskjelligeEtternavn();
    for (String etNavn : alle) System.out.println(etNavn);
    db.kobleNedForbindelse();
  }
```

Kapittel 20.4

Oppgave 1

Metoden ser slik ut:

```
  public int finnPerson(String fornavn, String etternavn) {
    String sqlSetning = "select * from person where fornavn = '" +
      fornavn.toUpperCase() + "'and etternavn = '" + etternavn.toUpperCase() + "'";
    System.out.println(sqlSetning);
    Statement setning = null;
    ResultSet res = null;
    int pnr = -1;
    try {
      setning = forbindelse.createStatement();
      res = setning.executeQuery(sqlSetning);
      if (res.next()) pnr = res.getInt("persnr");
    } catch (SQLException e) {
      skrivMelding(e, "finnPerson()");
    } finally {
      lukkResSet(res);
      lukkSetning(setning);
    }
    return pnr;
  }
```

Vi lager en applikasjon med et enkelt vindu:


```
mport javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import mittBibliotek.Person;
import static javax.swing.JOptionPane.*;

class DatabaseGUI extends JFrame {
  private Database databasekontakt;
  private JButton trykknapp = new JButton("Fins personen i databasen?");
  private JTextField fornavnFelt = new JTextField(20);
  private JTextField etternavnFelt = new JTextField(20);

  public DatabaseGUI(Database startDatabase) {
    databasekontakt = startDatabase;
    add(new InputPanel(), BorderLayout.CENTER);
    trykknapp.setBackground(Color.PINK);
    add(trykknapp, BorderLayout.SOUTH);
    trykknapp.addActionListener(new Knappelytter());
    addWindowListener(new Vinduslytter());
    pack();
  }

  private class InputPanel extends JPanel {
    public InputPanel() {
      setLayout(new GridLayout(2, 2, 5, 5));
      add(new JLabel("Fornavn: ", JLabel.RIGHT));
      add(fornavnFelt);
      add(new JLabel("Etternavn: ", JLabel.RIGHT));
      add(etternavnFelt);
    }
  }

  private class Knappelytter implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      String fornavn = fornavnFelt.getText();
      String etternavn = etternavnFelt.getText();
      int id = databasekontakt.finnPerson(fornavn, etternavn);
      if (id > 0) {
          showMessageDialog(null,  "Personen er registrert med nummeret " + id);
      } else {
        Person p = databasekontakt.registrerNyPerson(fornavn, etternavn);
        showMessageDialog(null,
          "Navnet er lagret, og personen har fått nummer " + p.finnPersonNr());
      }
    }
```

```
  }

  private class Vinduslytter extends WindowAdapter {
    public void windowClosing(WindowEvent hendelse) {
      databasekontakt.kobleNedForbindelse();
      System.exit(0);
    }
  }
}

class Oppg20_4_1 { // Exception fra Database-konstruktøren
  public static void main(String[] args) throws Exception {
    String databasedriver = "oracle.jdbc.driver.OracleDriver";
    String databasenavn = "jdbc:oracle:thin:@oracle.stud.aitel.hist.no:1521:oracle";
    String brukernavn = showInputDialog("Brukernavn: ");
    String passord = showInputDialog("Passord: ");
    Database db =
        new Database(databasedriver, databasenavn, brukernavn, passord, 1);
    DatabaseGUI vindu = new DatabaseGUI(db);
    vindu.setVisible(true);
  }
}
```

Kapittel 20.5

Oppgave 1

For å kunne kjøre testprogrammet fra en annen katalog,. må alle klassene (og interfacene) ligge i en egen katalog. Navnet på kaalogen blir navnet på pakken som klassene tilhører. Vi kaller katalogen (og pakken) Oppg20_5_1.

Interfacene er nå klassenes grensesnitt mot omverden, og de ser slik ut:

```
package Oppg20_5_1;
public interface Forbindelse {
  int finnNr();
  java.sql.Connection finnForbindelse();
  boolean erLedig();
  String toString();
}

package Oppg20_5_1;
public interface DatabasePool {
  Forbindelse reserverForbindelse();
  boolean frigiForbindelse(int nr);
  String lagUtskrift();
  void lukkAlleForbindelser();
}
```

Implementasjonen er som i boka, men skal ikke nåes "utenfra". Klassene har derfor kun pakketilgang. Første del av filene ser slik ut:

```
package Oppg20_5_1;
import java.sql.*;
class ForbindelseImpl implements Forbindelse {
```

og

```
package Oppg20_5_1;
import java.util.ArrayList;
import java.sql.*;
class DatabasePoolImpl implements DatabasePool {
```

Vi må lage en mekanisme slik at en klient kan opprette et objekt av klassen DatabasePoolImpl. Vi lager en "fabrikk":

```
package Oppg20_5_1;
public class DatabasePoolFabrikk {
  public static DatabasePool lagPool(int startKap, String dbDriver,
      String startDbNavn, String startBrukernavn, String startPassord)
                                  throws Exception {
    return new DatabasePoolImpl(startKap, dbDriver, startDbNavn,
                                                  startBrukernavn, startPassord);
  }
}
```

Testprogrammet ligger nå i katalogen over Oppg20_5_1. Vi må importere pakken:

```
import Oppg20_5_1.*;
```

Og vi oppretter databasepoolen slik:

```
DatabasePool dbPool = DatabasePoolFabrikk.lagPool(poolKapasitet,
                                  dbDriver, dbNavn, brukernavn, passord);
```

Kapittel 20.6

Oppgave 1

Kun få endringer er nødvendig.

Tillegg i import-listen i begynnelsen av filen:

```
import mittBibliotek.database.Database;
import mittBibliotek.database.DbWrapperFabrikk;
import java.rmi.Naming;
import java.rmi.RemoteException;
```

Vi må gjøre en liten endring i main( ) slik at vi kontakter databasen via databasepoolen og ikke direkte. Vi skifter ut setningen

```
Database databasekontakt =
   new Database(dbDriver, dbNavn, brukernavn, passord, feilkode);
```

med

```
DbWrapperFabrikk dbForb =
    (DbWrapperFabrikk) Naming.lookup("rmi://localhost/Persondatabase");
Database databasekontakt = dbForb.lagDbWrapper();
```

Alle operasjoner mot databasen går nå via RMI, og kan derfor medføre at RemoteException kastes. Vi må derfor legge inn unntakshåndtering i metoden fyllListeMedData( ) (på grunn av kallet på database.finnAlle( )), og i metoden actionPerformed( ) (på grunn av database.slettPerson( ), m.m.):

```
try {
  ...kode med databaseoperasjon(er) ...
```

```
} catch (RemoteException e) {
  showMessageDialog(DatabaseGUI.this,
          "Må avslutte på grunn av kommunikasjonsproblemer: " + e);
  e.printStackTrace();
  System.exit(0);
}
```

Vi skal ikke lenger koble ned databaseforbindelsen, det tar poolen seg av. Vi kan derfor sløyfe klassen Vinduslytter. I konstruktøren DatabaseGUI bytter vi ut

```
setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
```

med

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Vi kjører systemet ved først å starte rmiregisteret og deretter tjenerprogrammet som vi finner i pakken mittBibliotek.database. Endelig starter vi applikasjonsprogrammet. Vi kan gjerne kjøre opp flere utgaver av det.

Oppgave 2

Det er flere ting å passe på når vi skal lage en applet.

Vi begynner med mittBibliotek. Appleten vil bruke klasser fra denne pakken. Vanlige Java-program finner denne pakken ved å bruke CLASSPATH. For en applet blir forholdene annerledes. En applet skal i prinsippet kunne kjøres fra hvor som helst. Nettleseren laster ned klassen fra tjenermaskinen, og den bryr seg ikke om CLASSPATH, verken den som måtte finnes på tjenermaskinen eller på klientmaskinen. CODEBASE i *html*-filen bestemmer hvor nettleseren skal lete etter klasser. Dersom CODEBASE ikke er oppgitt, vil nettleseren lete på den katalogen der *html*-filen ble lastet ned fra. Først lastes applet-klassen ned. Dersom flere klasser trengs (noe som ofte er tilfelle), vil den lete på den katalogen der appleten ble lastet ned fra. Det enkleste er derfor å legge underkatalogen mittBibliotek der de øvrige klassene våre ligger. Kopier katalogen, ikke bare innholdet i katalogen.

Vi skal gjøre om klassen DatabaseApplikasjon fra programliste 20.3 til en applet. Vi tar bort main( ) og konstruktøren. Oppbygningen av brukergrensesnittet skal nå ligge i init( )-metoden. Oppkolpongen mot databasen ligger i start( )-metoden som kalles etter init( )-metoden (se figur 18.2). Vi skal videre gjøre de endringene som er nødvendig for at appleten kan kjøre i trelagsarkitekturen på figur 20.7. Det betyr at endringene fra oppgave 1 skal være med. Første del av klassen DatabaseApplet ser slik ut (merk at klassen må være public):

```
... nødvendige import-setninger ...
public class DatabaseApplet extends JApplet {
  private Database databasekontakt;

  private DefaultListModel listeinnhold = new DefaultListModel();
  private JList liste = new JList(listeinnhold);

  private JButton slettKnapp = new JButton("Slett");
  private JButton nyKnapp = new JButton("Ny");
```

```
private JButton endreKnapp = new JButton("Endre");
private JButton friskOppKnapp = new JButton("Gjenoppfrisk");
private PersonDialog persondialog = new PersonDialog(null); // null i appleter

/*
 * init()-metoden:
 * Konstruerer vinduet når appleten lastes inn.
 */
public void init() {
  persondialog.setLocation(300, 400);
  add(new OverskriftPanel("Vedlikehold av navneregister"),
                                                    BorderLayout.NORTH);
  add(new ListePanel(), BorderLayout.CENTER);
  add(new KnappePanel(), BorderLayout.SOUTH);
}

/* start()-metoden:
 * Kobler opp mot databasepoolen,
 * og fyller listeboksen med de registrerte navnene.
 */
public void start() {
   String databasedriver = "oracle.jdbc.driver.OracleDriver";
   String databasenavn = "jdbc:oracle:thin:@oracle.stud.aitel.hist.no:1521:oracle";
   String brukernavn = JOptionPane.showInputDialog("Brukernavn: ");
   String passord = JOptionPane.showInputDialog("Passord: ");

  try {   // Oppretter kontakt
    /* Henter navnet på maskinen der appleten ble lastet ned fra: */
    String tjenermaskin = getCodeBase().getHost();
    String url = "rmi://" + tjenermaskin + "/Persondatabase";
    DbWrapperFabrikk dbForb = (DbWrapperFabrikk) Naming.lookup(url);
    databasekontakt = dbForb.lagDbWrapper();
    fyllListeMedData();
  } catch (Exception e) {
    System.out.println("Feil i start(): " + e);
   JOptionPane.showMessageDialog(null, "Får ikke koplet opp mot databasen, " +
      "kan skyldes ugyldig brukernavn og passord. " +
      "Se Java Console-vindu for mer informasjon.");
    removeAll();
  }
}
```

Deretter kommer metoden fyllListeMedData(), klassene OverskriftPanel, ListePanel, KnappePanel og KnappeLytter som i programliste 20.3.

Du trenger en *html*-fil som kaller opp appleten. Her er den som verktøyet TextPad genererer automatisk:

```
<!DOCTYPE HTML><HTML><HEAD></HEAD><BODY>
<APPLET CODE="DatabaseApplet.class" CODEBASE="."
  WIDTH=400 HEIGHT=300></APPLET>
</BODY></HTML>
```

For å kunne kjøre appleten i appletviewer (se vedlegg A2) må du først starte rmi-registeret og deretter tjenerprogrammet i pakken mittBibliotek.database.

Når du nå har fått appleten til å kjøre i appletviewer, er en vanlig nettleser neste skritt. Da må vi kjøre webtjener. Den må kjøre på samme maskin som rmi-registeret. Årsaken til det er at en applet kun kan kommunisere med program på den maskinen den ble lastet ned fra.

Den midterste boksen på figur 20.7 må utvides med et objekt for webtjeneren, mens applikasjonen i den øverste boksen skiftes ut med en nettleser med en applet inne i seg.

I kapittel 21 lærer du hvordan du bruker webtjeneren Tomcat. Du kan imidlertid bruke en hvilken som helst webtjener, bare den kjører på samme maskin som rmiregisteret og tjenerprogrammet (DatabaseTjener).

Før du kan laste ned appleten i en nettleser, må tre programmer kjøre på den maskinen der du henter appleten fra:

- webtjener
- rmi-register
- tjenerprogrammet DatabaseTjener fra pakken mittBibliotek.database.

Med Tomcat kjørende på port 9090, vil URL'en se f.eks. slik ut:

http://156.76.50.237:9090/Oppg20_6_2/database.html

Kapittel 20.7

Oppgave 1

Ingen endringer registreres i databasen før setningen

```
forbindelse.commit();
```

utføres. Dersom vi avbryter programmet før dette skjer, vil databaseinnholdet være det samme som før programmet startet.

Kapittel 20.8

Oppgave 2

```
// Oppgave a)
Statement setning = forbindelse.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

// Oppgave b)
ResultSet res = setning.executeQuery("select person.* from person");

// Oppgave c)
System.out.println("Oppgave c");
while (res.next()) {
  int persNr = res.getInt("persnr");
  String fornavn = res.getString("fornavn");
  String etternavn = res.getString("etternavn");
  System.out.println(persNr + ": " + fornavn + " " + etternavn);
```

```
}

// Oppgave d)
res.absolute(3);
res.updateInt("persnr", 555);
res.updateRow();

// Oppgave e)
System.out.println("Oppgave e");
res.beforeFirst();
while (res.next()) {
  int persNr = res.getInt("persnr");
  String fornavn = res.getString("fornavn");
  String etternavn = res.getString("etternavn");
  System.out.println(persNr + ": " + fornavn + " " + etternavn);
}

showMessageDialog(null, "Trykk OK når vi skal fortsette");
```

Endringene er lagret i databasen.

Vi legger inn en ny rad (se online API-dokumentasjonen for beskrivelse av de metodene som brukes):

```
res.moveToInsertRow();
res.updateInt("persnr", 371);
res.updateString("fornavn", "Unni");
res.updateString("etternavn", "Olsen");
res.insertRow();
```

(Feil i Oracle ojdbc14.jar-driveren: Resultatsettet oppdateres ikke med den nye raden, men den lagrs i databasen.)

Vi sletter en første rad:

```
res.last();
System.out.println("Sletter person " + res.getInt("persnr"));
res.deleteRow();
```

Kapittel 20.9

Oppgave 1

Vi må legge inn alle setningsobjektene som objektvariabler i klassen Database og åpne dem i konstruktøren.

Ny objektvariabler:

```
private PreparedStatement setningFinnAlle;
private PreparedStatement setningEndreNavn;
private PreparedStatement setningRegNyPerson;
private PreparedStatement setningSlett;
private PreparedStatement setningFinnMaks;
```

Vi lager en privat hjelpemetode for åpning av disse:

```
private void lagSetningsobjekt() {
  try {
```

```
      setningFinnAlle = forbindelse.prepareStatement(
              "select persnr, initcap(fornavn) as fn, initcap(etternavn) as en " +
              "from person order by etternavn, fornavn");
      setningEndreNavn = forbindelse.prepareStatement(
        "update person set fornavn = upper(?), etternavn = upper(?) where persnr = ?");
      setningFinnMaks = forbindelse.prepareStatement(
        "select max(persnr) as maks from person");
      setningRegNyPerson = forbindelse.prepareStatement(
         "insert into person values(?, upper(?), upper(?))");
      setningSlett = forbindelse.prepareStatement(
        "delete from person where persnr = ?");
    } catch (SQLException e) {
      skrivMelding(e, "lagSetningsobjekt()");
      lukkSetningsobjekt();
    }
  }
```

Denne metoden kaller vi helt til slutt i konstruktøren.

Tilsvarende lager vi en metode som vi kaller i begynnelsen av kobleNedForbindelse():

```
  private void lukkSetningsobjekt() {
    lukkSetning(setningFinnAlle);
    lukkSetning(setningEndreNavn);
    lukkSetning(setningRegNyPerson);
    lukkSetning(setningSlett);
    lukkSetning(setningFinnMaks);
  }
```

Nå ser de øvrige metodene slik ut:

```
  public ArrayList<Person> finnAlle() {
     ArrayList<Person>  alle = new ArrayList<Person>();
     ResultSet res = null;
     try {
       res = setningFinnAlle.executeQuery();
       while (res.next()) {
         int nr = res.getInt("persnr");
         String fornavn = res.getString("fn");
         String etternavn = res.getString("en");
         Person navnet = new Person(nr, fornavn, etternavn);
         alle.add(navnet);
       }
     } catch (SQLException e) {
       skrivMelding(e, "finnAlle()");
       alle = null;
     } finally {  // setningene nedenfor utføres alltid
       lukkResSet(res);
       return alle;
     }
   }

   public boolean endreNavn(Person personen) {
     int nr = personen.finnPersonNr();
```

```
  String nyttFornavn = personen.finnFornavn();
  String nyttEtternavn = personen.finnEtternavn();
  try {
    setningEndreNavn.setString(1, nyttFornavn);
    setningEndreNavn.setString(2, nyttEtternavn);
    setningEndreNavn.setInt(3, nr);
    if (setningEndreNavn.executeUpdate() == 0) return false;
    else return true;
  } catch (SQLException e) {
    skrivMelding(e, "endreNavn()");
  }
  return false;
}

public Person registrerNyPerson(String fornavn, String etternavn) {
  int nyttNr = 1; // dersom det ikke er data i databasen
  boolean ok = true;
  do {
    ResultSet res = null;
    try {
      res = setningFinnMaks.executeQuery();
      if (res.next()) nyttNr = res.getInt("maks") + 1;
      setningRegNyPerson.setInt(1, nyttNr);
      setningRegNyPerson.setString(2, fornavn);
      setningRegNyPerson.setString(3, etternavn);
      setningRegNyPerson.executeUpdate();
    } catch (SQLException e) {
      /* Dersom feilen skyldes at person med samme nummer eksisterer fra før,
         må det bety at en annen klient har vært inne i databasen mellom
         våre to sql-setninger. Vi kjører derfor gjennom en gang til.
         (Dette er noe som vil inntreffe svært sjelden.) */
      if (e.getErrorCode() == kode) ok = false;
      else skrivMelding(e, "registrerNyPerson()");
    } finally {
      lukkResSet(res);
    }
  } while (!ok);
  return new Person(nyttNr, fornavn, etternavn);
}

public boolean slettPerson(int nr) {
  try {
    setningSlett.setInt(1, nr);
    if (setningSlett.executeUpdate() == 0) return false;
    else return true;
  } catch (SQLException e) {
    skrivMelding(e, "slettPerson()");
  }
  return false;
}
```

Kapittel 21.3

Oppgave 2

body-elementet ser slik ut:

```
out.println("<body>");
out.println("Tidspunkt for nedlasting: " + new java.util.Date());
out.println("<p>Tilfeldige tall i intervallet [0, 1000]: ");
java.util.Random tilftall = new java.util.Random();
for (int i = 0; i < 5; i++) out.print(tilftall.nextInt(1001) + " ");
out.println("</body>");
```

Oppgave 3

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  String maskinNavn = request.getRemoteHost();
  System.out.println("Forespørsel fra " + maskinNavn);
  .....
```

Kapittel 21.5

Oppgave 1

```
<%@ page import="java.util.*,java.text.*"%>
<html><head><title>Oppgave 21_5_1</title></head>
<body>
Dagens dato:
<%
GregorianCalendar kalender = new GregorianCalendar();
int dag = kalender.get(Calendar.DAY_OF_WEEK);
/* Formaterer i henhold til det miljøet programmet kjører i (norsk) */
DateFormat datoformat = DateFormat.getDateInstance();
String tekst = datoformat.format(new java.util.Date());
if (dag == Calendar.SATURDAY || dag == Calendar.SUNDAY) {
  tekst += " - helg";
  if (dag == Calendar.SUNDAY) {
    tekst = "<font color = 'red'>" + tekst + "</font>";
  }
}
%>
<%= tekst %>
</body>
</html>
```

Oppgave 2

```
<%@ page import="java.text.*"%>
<html><head><title>Oppgave 21_5_2</title></head>
<body>
<table border=1>
<tr><th>Tall</th><th>Kvadratrot</th></tr>
<%
DecimalFormat formatet = new DecimalFormat("###0.000");
for (int nr = 1; nr <= 20; nr++) {
```

```
    out.println("<tr><td>" + nr + "</td><td>" + formatet.format(Math.sqrt(nr)) +
"</td></tr>");
}
%>
</table>
</body></html>
```

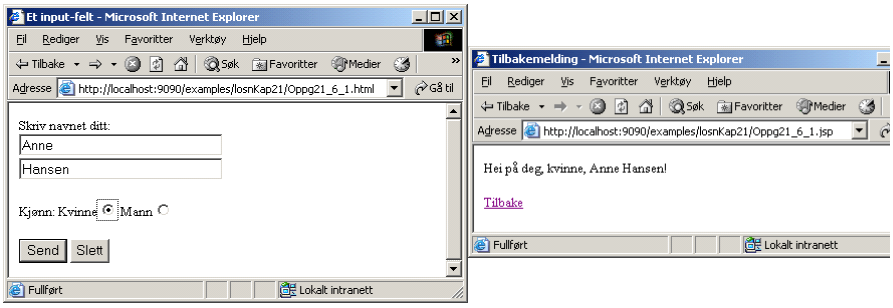
Oppgave 3

```
<%@ page import="java.util.*"%>
<html><head><title>Oppgave 21_5_3</title></head>
<body>
<%! private int teller = 0; %>
<%! private int sum = 0; %>
<%! private Vector<Integer> alleTallene = new Vector<Integer>(); %>
<%
teller++;
int random = (int) (100 * Math.random() + 1);
alleTallene.add(random);  // autoboxing til Integer-objekt
sum += random;
%>
<P>Siden er lastet ned <%= teller %> ganger.
<P>Summen av de tilfeldige tallene er <%= sum %>.
<P>De tilfeldige tallene er:
<% for (int tall : alleTallene) {
    out.print(" " + tall);
  }
 %>
</body></html>
```

Kapittel 21.6

Oppgave 1

Nettleservinduene ser slik ut:



Her er *html*-filen:

```
<html><head><title>Et input-felt</title></head>
<form action="Oppg21_6_1.jsp" method="post">
<p>Skriv navnet ditt:<br>
<input name="fornavn" type="text" size="30"><br>
<input name="etternavn" type="text" size="30">
<p>
```

```
Kjønn:
Kvinne<input name="kjonn" type="radio" value="kvinne" checked>
Mann<input name="kjonn" type="radio" value="mann">
<p>
<input type="submit" name="Send" value="Send">
<input type="reset" name="Slett" value="Slett">
</form></body></html>
```

Og her er *jsp*-filen:

```
<html><head><title>Tilbakemelding</title></head><body>
<%
String fornavnet = request.getParameter("fornavn");
String etternavnet = request.getParameter("etternavn");
if (fornavnet != null) {
  fornavnet = fornavnet.trim();
  etternavnet = etternavnet.trim();
  if (etternavnet.equals("") || fornavnet.equals("")) {
    out.println("Du må skrive både fornavn og etternavn!");
  }
  else out.println("Hei på deg, " + request.getParameter("kjonn") +
                   ", " + fornavnet + " " + etternavnet + "!");
}
%>
<p>
<a href = "Oppg21_6_1.html">Tilbake</a>
</body></html>
```

Oppgave 2

Her er den nye filen, som aktiviseres fra *RestaurantEvaluation.html*:

```
<%--
   En hjelpemetode som setter sammen en tabell av strenger til én streng.
   Denne metoden hadde vi tidligere i filen RestaurantEvaluation.jsp
--%>
<%!
String lagEnTekst(String[] verdier) {
  String tekst = "";
  if (verdier != null) {
    for (int i = 0; i < verdier.length - 1; i++) tekst += verdier[i] + ", ";
    tekst += verdier[verdier.length - 1];
  }
  return tekst;
}
%>

<%
String navnRestaurant = request.getParameter("navnSpisested");
String fornøyd = lagEnTekst(request.getParameterValues("fornoyd"));
String ikkeFornøyd = lagEnTekst(request.getParameterValues("ikkeFornoyd"));
String meny = request.getParameter("matrett");
String kommentarer = request.getParameter("kommentarer");
String kjønn = request.getParameter("kjonn");
String alder = request.getParameter("alder");
```

```
%>

<p>Du har skrevet inn følgende data:

<%
out.println("<br>Restaurant: " + navnRestaurant);
out.println("<br>Fornøyd med: " + fornøyd);
out.println("<br>Ikke fornøyd med: " + ikkeFornøyd);
out.println("<br>Gjestene spiste: " + meny);
out.println("<br>Kommentarer: " + kommentarer);
out.println("<br>Kjønn: " + kjønn);
out.println("<br>Alder: " + alder);
%>
<form action="RestaurantEvaluation.jsp" method="post">
<!-- bruker skjulte felt for å overføre data til
      den reviderte RestaurantEvaluation.jsp-filen -->
<input name="navnSpisested" type="hidden" value="<%= navnRestaurant %>">
<input name="fornoyd" type="hidden" value="<%= fornøyd %>">
<input name="ikkeFornoyd" type="hidden" value="<%= ikkeFornøyd %>">
<input name="matrett" type="hidden" value="<%= meny%>">
<input name="kommentarer" type="hidden" value="<%= kommentarer%>">
<input name="kjonn" type="hidden" value="<%= kjønn %>">
<input name="alder" type="hidden" value="<%= alder %>">

<input type="submit" name="Lagre" value="OK å lagre dataene">
</form>
<font color = "red">Hvis du vil endre dataene, trykk på Tilbake-knappen i
nettleseren.
</font>
</body></html>
```

Filen *RestaurantEvaluation.jsp* er revidert:

```
<%@ page import="java.io.*" %>
<%! String relFilnavn = "EvalRest.txt"; %>

<%
  if (request.getParameter("Lagre") != null) {
    String filnavn = application.getRealPath(relFilnavn);

    String utskrift = // henter nå data fra skjulte felt
      "Restaurant: " + request.getParameter("navnSpisested") + "\n" +
      "Fornøyd med: " +  request.getParameter("fornoyd") + "\n" +
      "Ikke fornøyd med: " + request.getParameter("ikkeFornoyd") + "\n" +
      "Gjesten spiste: " +  request.getParameter("matrett") + "\n" +
      "Kommentarer: " + request.getParameter("kommentarer") + "\n" +
      "Kjønn: " + request.getParameter("kjonn") + "\n" +
      "Alder: " + request.getParameter("alder") + "\n" +
      "************************************************************" + "\n" + "\n";

    String fileName = application.getRealPath(relFilnavn);

    FileWriter skriveforbTilFil = new FileWriter(filnavn, true);
```

```
    PrintWriter skriver = new PrintWriter(new BufferedWriter(skriveforbTilFil));
    skriver.print(utskrift);
    skriver.close();

    out.println("<P>Din evaluering er lagret");
   } else {
    out.println("<p><strong>Ingenting lagret</strong><p>");
   }
  %>
```

Kapittel 21.8

Oppgave 1

```
<table bgcolor="skyblue" border="1">
  <tr><th>Personnr</th><th>Fornavn</th><th>Etternavn</th></tr>
  <%
  int teller = 0;
  try{
   .......
   while (resSet.next()) {
     out.print("<tr><td>" + resSet.getString("persnr") + "</td>");
     out.print("<td>" + resSet.getString("fornavn") + "</td>");
     out.println("<td>" + resSet.getString("etternavn") + "</td></tr>");
     teller++;
   }
   ......
  %>
</table>
<p>Antall personer er <%= teller%>
</body></html>
```

Oppgave 2

Vi legger inn et <form>-element mellom page-direktivet og <table>-elementet:

```
<%
String søkeVerdi = request.getParameter("etternavn");
if (søkeVerdi == null) søkeVerdi = "";
%>
<p>
<form action="Oppg21_8_2.jsp" method="post">
  Skriv etternavn for å begrense søket:
  <br><input name="etternavn" type="text" size="30" value="<%=søkeVerdi%>">
  <p><input type="submit" name="send" value="Search">
</form>
```

Vi legger inn følgende foran resSet = setning.executeQuery();

```
if (!søkeVerdi.equals("")) {
  sql += " where etternavn = '" + søkeVerdi.toUpperCase() + "'";
}
```

Oppgave 3

De viktigste endringene er i filen *Navneregister.jsp*.

For å unngå separate sider som varsler om oppdateringer, håndterer vi oppdateringene i filen *Navneregister.jsp*. Avhengig av typen oppdatering inkluderer vi riktig fil. Kontakt med databasen og håndtering av unntakssituasjoner i forbindelse med dette må tas ut av includefilene og legges i *Navneregister.jsp*. Dermed blir inkludefilene for oppdatering svært små, og vi velger derfor å sløyfe dem, og heller legge koden inn i Navneregister.jsp, slik:

```
<html><head><title>Navneregister</title></head>
<body bgcolor="wheat" text="darkgreen"
 link="brown" vlink="steelblue" alink="darkblue">
<h3>Navneregister</h3>
<%@include file = "dbPool.jsp"%>
<%
try {
  Database databasen = dbForb.lagDbWrapper();

  /* Sletter en person fra registeret */
 if (request.getParameter("slett") != null) {
   String valgtPerson = request.getParameter("personer");
   java.util.StringTokenizer text = new java.util.StringTokenizer(valgtPerson, ":, ");
   int personNr = Integer.parseInt(text.nextToken());
   if (databasen.slettPerson(personNr)) {
     out.println("Sletting av " + personNr + " foretatt\n");
    }
    else out.println("Fant ikke personen, kan være slettet av andre\n");

 /* Lagrer nye data */
 } else if (request.getParameter("ny") != null) {
   String fornavn = request.getParameter("fornavn");
   String etternavn = request.getParameter("etternavn");
   Person personen = databasen.registrerNyPerson(fornavn, etternavn);
   out.println("<br>Personen har fått nummer: " + personen.finnPersonNr() + "\n");

 /* Endrer data */
 } else if (request.getParameter("endre") != null) {
  int pNr = Integer.parseInt(request.getParameter("personnr"));
  String fornavn = request.getParameter("fornavn");
  String etternavn = request.getParameter("etternavn");
  Person personen = new Person(pNr, fornavn, etternavn);
  if (databasen.endreNavn(personen)) out.println("<br>Dataene er endret.");
  else out.println("Fant ingen med dette nummeret. Kan være slettet av andre.");
 }
%>

<form action = "BehandleValg.jsp" method = "post">
... osv. som tidligere ...
```

Filen *BehandleValg.jsp* endres noe:

```
<html><head><title>Håndterer brukerens valg</title></head>
<body bgcolor="wheat" text="darkgreen"
```

```
 link="brown" vlink="steelblue" alink="darkblue">

<%@ page import="java.util.StringTokenizer" %>
<% /* Sletting av data */
  if (request.getParameter("slett") != null) {
    %>
    <%-- Sender forespørselen videre til siden Navneregister.jsp --%>
    <jsp:forward page="Navneregister.jsp"/>
  <%
  }

  /* Registrere ny person */
  if (request.getParameter("ny") != null) {
    String persnr = "ikke definert";
    String fornavn = "";
    String etternavn = "";
    String submitknapp = "ny";
    %>
    <%@ include file = "PersonForm.jsp"%>
  <%
  }

  /* Endre navn på registrert person */
  if (request.getParameter("endre") != null) {
    String valgtPerson = request.getParameter("personer");
    StringTokenizer tekst = new StringTokenizer(valgtPerson, ":, ");
    String persnr = tekst.nextToken();
    String etternavn = tekst.nextToken();
    String fornavn = tekst.nextToken();
    int antOrdIgjen = tekst.countTokens();
    for (int i = 0; i < antOrdIgjen; i++) fornavn += (" " + tekst.nextToken());
    String submitknapp = "endre";
    %>
    <%@ include file = "PersonForm.jsp"%>
  <%
  }
%>
</body>
</html>
```

PersonForm.jsp skal nå sende klienten videre til *Navneregister.jsp*. Dette gjelder både ved endring og nyregistrering av data. Imidlertid trenger *Navneregister.jsp* å vite om det var en oppdatering eller en nyregistrering. Det forteller vi med verdien til variabelen submitknapp. Bare et par små endringer i *PersonForm.jsp*:

```
<br><strong>Registrere/endre data</strong>
<form action="Navneregister.jsp" method = "post">
  <table>
   ... som før ...
  </table>
  <input name = "<%=submitknapp%>" type = "submit" value = "Lagre">
  <input name = "blank" type = "reset" value = "Blank">
```

```
</form>
```

Kapittel 21.9

Oppgave 1

Utvider tabellen i *CookiesTest.jsp*:

```
<table>
  <tr>
    <td align = left>Fornavn:</td>
    <td align = left><input type = "text" name = "fornavn" size = "20"></td>
  </tr><tr>
    <td align = left>Etternavn:</td>
    <td align = left><input type = "text" name = "etternavn" size = "20"></td>
  </tr><tr>
    <td align = left>Adresse:</td>
    <td align = left><input type = "text" name = "adresse" size = "20"></td>
  </tr><tr>
    <td align = left>Postnummer:</td>
    <td align = left><input type = "text" name = "postnr" size = "20"></td>
  </tr><tr>
    <td align = left>Poststed:</td>
    <td align = left><input type = "text" name = "poststed" size = "20"></td>
</td>
</table>
```

Her er den reviderte *LagreNavnSomCookie.jsp*:

```
<html><head><title>Lagrer navn som cookie</title></head><body>
<%
  String fornavn = request.getParameter("fornavn");
  String etternavn = request.getParameter("etternavn");
  String adr = request.getParameter("adresse");
  String postnr = request.getParameter("postnr");
  String sted = request.getParameter("poststed");
  out.println("<br>Har funnet: " + fornavn + " " + etternavn + "<br>" +
              adr + "<br>" + postnr + " " + sted);
  response.addCookie(new Cookie("fornavn", fornavn));
  response.addCookie(new Cookie("etternavn", etternavn));
  response.addCookie(new Cookie("adresse", adr));
  response.addCookie(new Cookie("postnr", postnr));
  response.addCookie(new Cookie("poststed", sted));
  out.print("<br>Nå er cookie'ne lagret");
%>
</body></html>
```

Oppgave 2

Sett inn

```
cookie1.setMaxAge(24 * 3600);
```

i *LagreNavnSomCookie.jsp*, etter at cookie-objektet er laget.

Oppgave 3

Vi må gjøre noen få endringer i *RestaurantEvaluation.jsp*:

```
<%
  if (request.getParameter("Send") != null) {
   if (session.getAttribute("lagret") == null) {
      ... som før ...
      skriver.print(utskrift);
      skriver.close();

      session.setAttribute("lagret", "ok");
      out.println("<P>Din evaluering er lagret");
    } else out.println("<P>Evalueringen er allerede lagret.");
  }
%>
```

Kapittel 22.2

Oppgave 1

```
/*
* Merknad: Tester på umulige flater, med "negativt areal" er kommentert
 * ut fordi klassen Flate godtar dette.
 */

import junit.framework.*;

public class JUnitOppussingTest extends TestCase {

  private Belegg etBelegg;
  private Maling enMaling;
  private Tapet etTapet;
  private Flate enLettFlate;
  private Flate enVerreFlate;
  private Flate enTomFlate;
  private Flate enUmuligFlate;

  public JUnitOppussingTest(String name) {
    super(name);
  }

  protected void setUp() {
    etBelegg = new Belegg("Testbelegg", 100, 2);
    enMaling = new Maling("Testmaling", 50, 2, 4);
    etTapet = new Tapet("Testtapet", 150, 3, 2);
    enLettFlate = new Flate("Lettflate", 4, 2);
    enVerreFlate = new Flate("Verreflate", 5.5, 2);
    enTomFlate = new Flate("Tomflate", 0, 2);
    enUmuligFlate = new Flate("Umuligflate", -5, 2);
  }

  protected void tearDown() {

  }
```

```
  public void testBelegg() {
    assertTrue("Feil med finnBredde().", etBelegg.finnBredde()==2);
    double behov;
    behov = etBelegg.finnMaterialbehov(enLettFlate);
    assertTrue("Feil med finnMarerialbehov(), lett flate", behov==4);
    behov = etBelegg.finnMaterialbehov(enVerreFlate);
    assertTrue("Feil med finnMarerialbehov(), verre flate", behov==6);
    behov = etBelegg.finnMaterialbehov(enTomFlate);
    assertTrue("Feil med finnMarerialbehov(), tom flate", behov==0);
    behov = etBelegg.finnMaterialbehov(enUmuligFlate);
    //assertTrue("Feil med finnMarerialbehov(), umulig flate", behov>=0);
  }

  public void testMaling() {
    assertTrue("Feil med finnAntStrøk()", enMaling.finnAntStrøk()==2);
    assertTrue("Feil med finnAntKvmPrLiter()", enMaling.finnAntKvmPrLiter()==4);
    double behov;
    behov = enMaling.finnMaterialbehov(enLettFlate);
    assertTrue("Feil med finnMarerialbehov(), lett flate", behov==4);
    behov = enMaling.finnMaterialbehov(enVerreFlate);
    assertTrue("Feil med finnMarerialbehov(), verre flate", behov==5.5);
    behov = enMaling.finnMaterialbehov(enTomFlate);
    assertTrue("Feil med finnMarerialbehov(), tom flate", behov==0);
    behov = enMaling.finnMaterialbehov(enUmuligFlate);
    //assertTrue("Feil med finnMarerialbehov(), umulig flate", behov>=0);
  }

  public void testTapet() {
    assertTrue("Feil med finnLengde()", etTapet.finnLengde()==3);
    assertTrue("Feil med finnBredde()", etTapet.finnBredde()==2);
    double behov;
    behov = etTapet.finnMaterialbehov(enLettFlate);
    assertTrue("Feil med finnMarerialbehov(), lett flate", behov==2);
    behov = etTapet.finnMaterialbehov(enVerreFlate);
    assertTrue("Feil med finnMarerialbehov(), verre flate", behov==3);
    behov = etTapet.finnMaterialbehov(enTomFlate);
    assertTrue("Feil med finnMarerialbehov(), tom flate", behov==0);
    behov = etTapet.finnMaterialbehov(enUmuligFlate);
    //assertTrue("Feil med finnMarerialbehov(), umulig flate", behov>=0);
  }

  public static Test suite() {
    return new TestSuite(JUnitOppussingTest.class);
  }

  public static void main(String[] args) {
    junit.textui.TestRunner.run(suite());
  }
}
```

Oppgave 2

```
/*
 * Klassenavn leses inn som første parameter til programmet.
 *
 */

import java.lang.reflect.*;
import java.util.*;

class KlasseLister {

  private String klassenavn;
  private Class klasse;

  private void settKlassenavn(String navn) {
    klassenavn = navn;
    try {
      klasse = Class.forName(klassenavn);
    }
    catch (ClassNotFoundException e) {
      // Klassen ikke funnet. Vil nå sjekke om klassen finnes i java.lang
      System.out.println("Klassen "+klassenavn+" finnes ikke.");
      int sistePunktum = klassenavn.lastIndexOf(".");
      if (sistePunktum==-1) {
        // Ingen punktum i strengen..
        klassenavn = "java.lang."+klassenavn;
      } else {
        // Den opprinnelige klassen, som ikke fantes, var et fullt navn,
        // med pakkenavn.
        klassenavn = "java.lang."+klassenavn.substring(sistePunktum+1);
      }
      try {
        klasse = Class.forName(klassenavn);
      }
      catch (ClassNotFoundException e2) {
        System.out.println("Klassen "+klassenavn+" finnes ikke.");
        System.exit(0);
      }
    }
  }

  public void listMetoder() {
    Method[] metoder = klasse.getMethods();
    TreeSet sorterteNavn = new TreeSet();
    System.out.println("Metoder i klassen " + klassenavn + ":");
    for (int i = 0 ; i < metoder.length ; i++) {
      sorterteNavn.add(metoder[i].getName());
      //System.out.println(metoder[i].getName());
    }
    System.out.println(sorterteNavn);
```

```
  }

  public void listKonstruktører() {
    Constructor[] konstruktører = klasse.getConstructors();
    TreeSet sorterteKons = new TreeSet();
    System.out.println("Konstruktører i klassen " + klassenavn + ":");
    for (int i = 0 ; i < konstruktører.length ; i++) {
      sorterteKons.add(konstruktører[i].getName());
    }
    System.out.println(sorterteKons);
  }

  public void listVariabler() {
    Field[] medlemmer = klasse.getFields();
    TreeSet sorterteMedlemmer = new TreeSet();
    System.out.println("Variabler i klassen " + klassenavn + ":");
    for (int i = 0 ; i < medlemmer.length ; i++) {
      sorterteMedlemmer.add(medlemmer[i].getName());
    }
    System.out.println(sorterteMedlemmer);
  }

  public static void main(String[] args) {
    if (args.length<1) {
      System.out.println("Gi inn klassenavn som parameter.");
      System.exit(1);
    }
    String navn = args[0];
    KlasseLister lister = new KlasseLister();
    lister.settKlassenavn(navn);
    lister.listMetoder();
    lister.listKonstruktører();
    lister.listVariabler();
  }
}
```

Kapittel 22.3

Oppgave 1

Dette vil neppe være tilstrekkelig. I utgangspunktet vil ikke enhetstester fokusere på samspill mellom systemets deler/enheter på samme måte som systemtester. Systemtester må utføre operasjoner på systemet som er relevante og mer eller mindre sannsynlige i normal bruk. Det er vanskelig å tenke seg at enhetstester på en eller annen måte skal danne et mønster som medfører dette.