Web- og serverprogrammering

Arkitekturer i webprogrammer - dag 6

Model-View-Controller-arkitukturen
Flerlags-arkitekturer

Læsning: WJSP 10

Model-View-Controller

1.Modellen (data og de bagvedliggende beregninger - forretningslogik)

Domæne-modellen – universel anvendelse (oftest lavet som servletter – rent Java)

En bankkonto har navn på ejer, kontonummer, kort-ID, saldo, bevægelser, etc.
Saldo kan ikke ændres direkte, men påvirkes med handlingerne (metoderne) overførsel, udbetaling og indbetaling.

2.Præsentationen af data over for brugeren

Producerer HTML-kode (oftest lavet som JSP-sider)

Bankkonti præsenteres meget forskelligt.
I en pengeautomat vises ingen personlige oplysninger overhovedet.

I et netbank-system kan saldo og bevægelser ses (det kunne være en webløsning i HTML).

3.Kontrollør

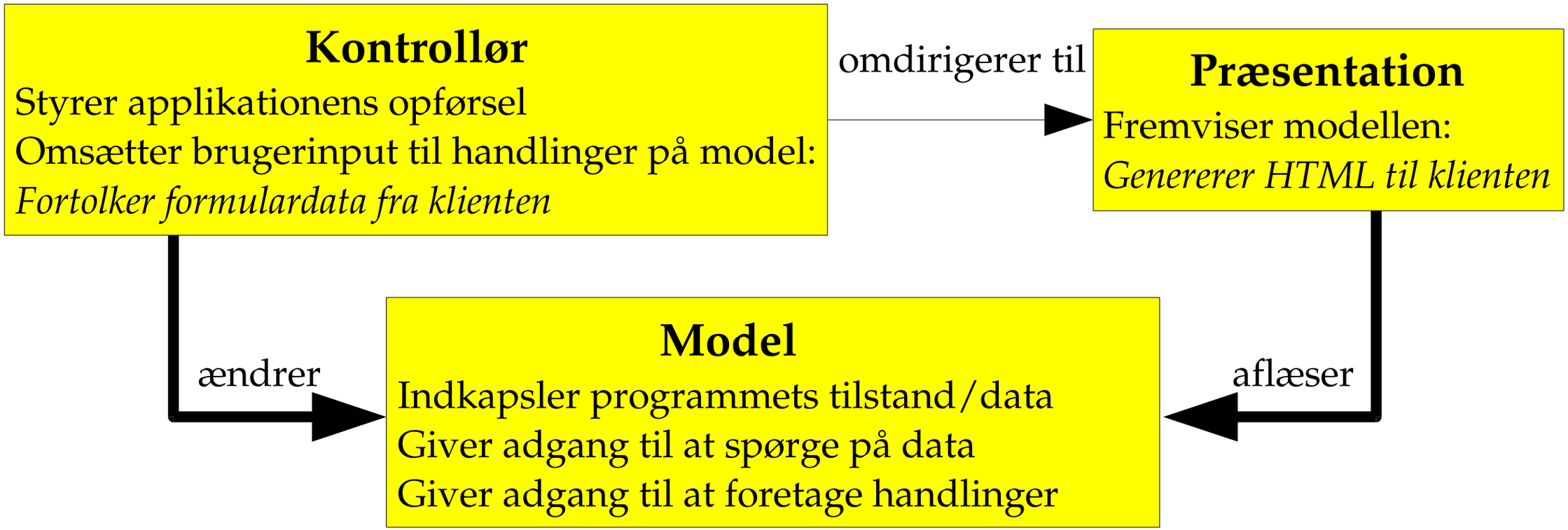
Omsætter bruger-input til handlinger:

- ændringer på modellen
- beslutning om næste præsentation

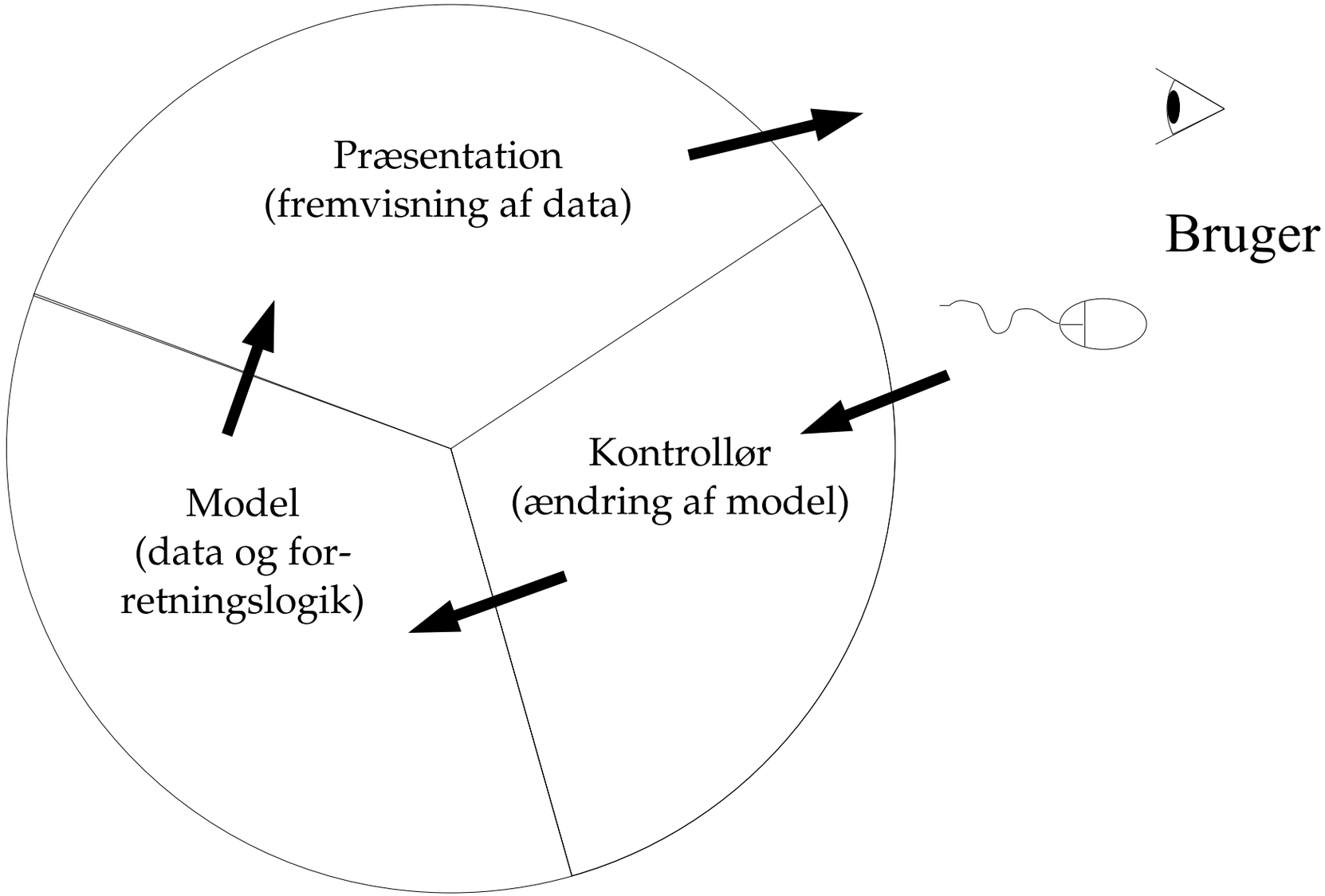
I en pengeautomat kan man kun hæve penge.

I et netbank-system kan brugeren måske lave visse former for overførsel fra sin egen konto.

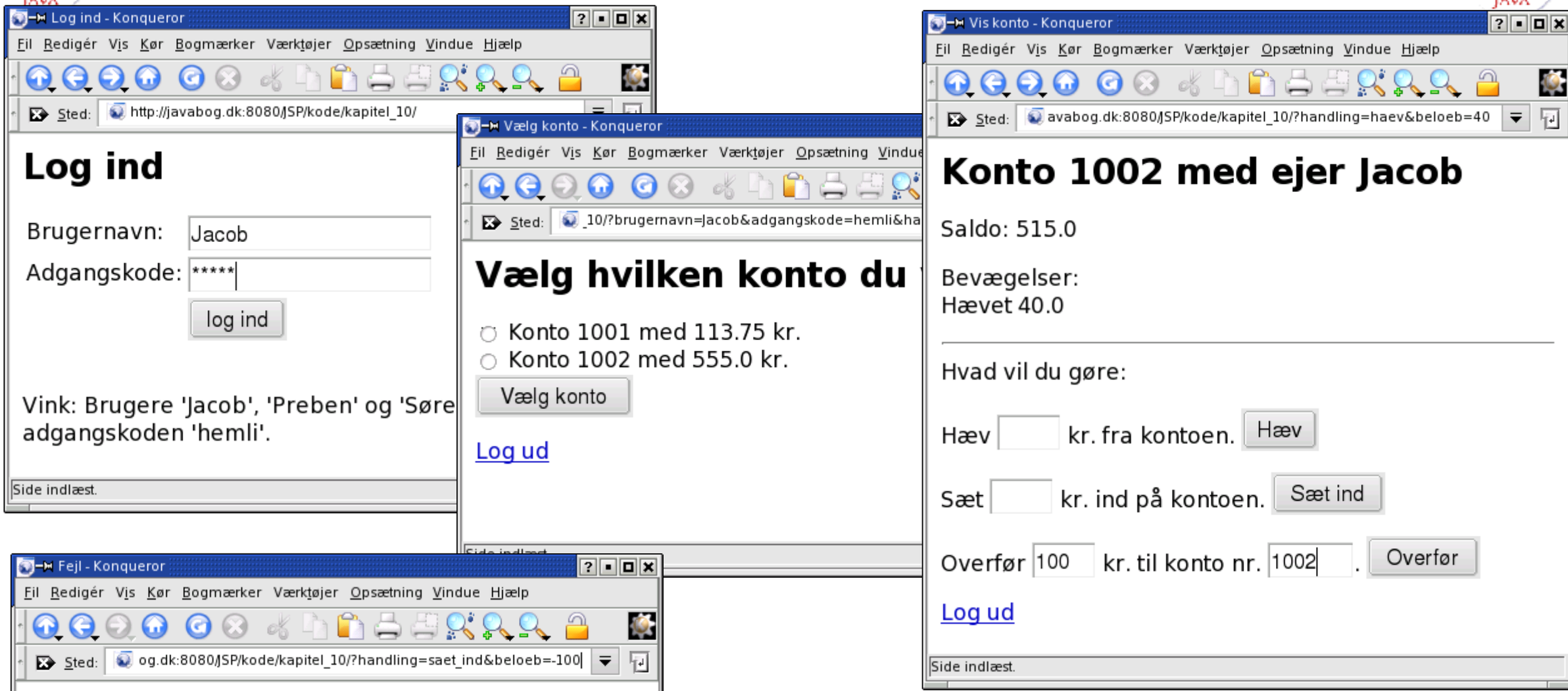
Ved skranken kan medarbejderen derudover foretage ind- og udbetalinger.

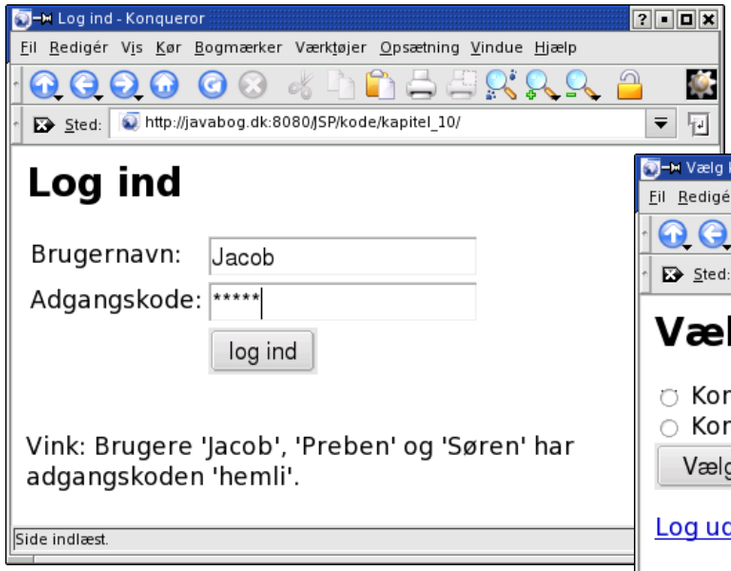

Kontrollør
Styrer applikationens opførsel
Omsætter brugerinput til handlinger på model:
Fortolker formulardata fra klienten
omdirigerer til
Præsentation
Fremviser modellen:
Genererer HTML til klienten
ændrer
Model
Indkapsler programmets tilstand/data
Giver adgang til at spørge på data
Giver adgang til at foretage handlinger
aflæser

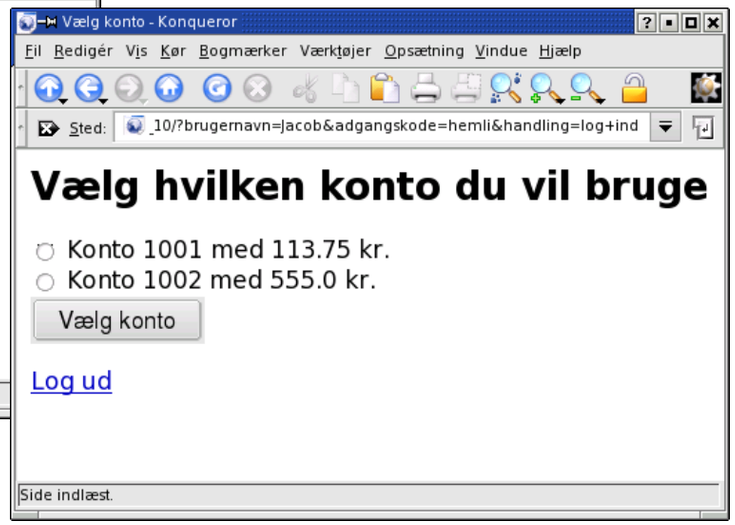
Informationsstrøm



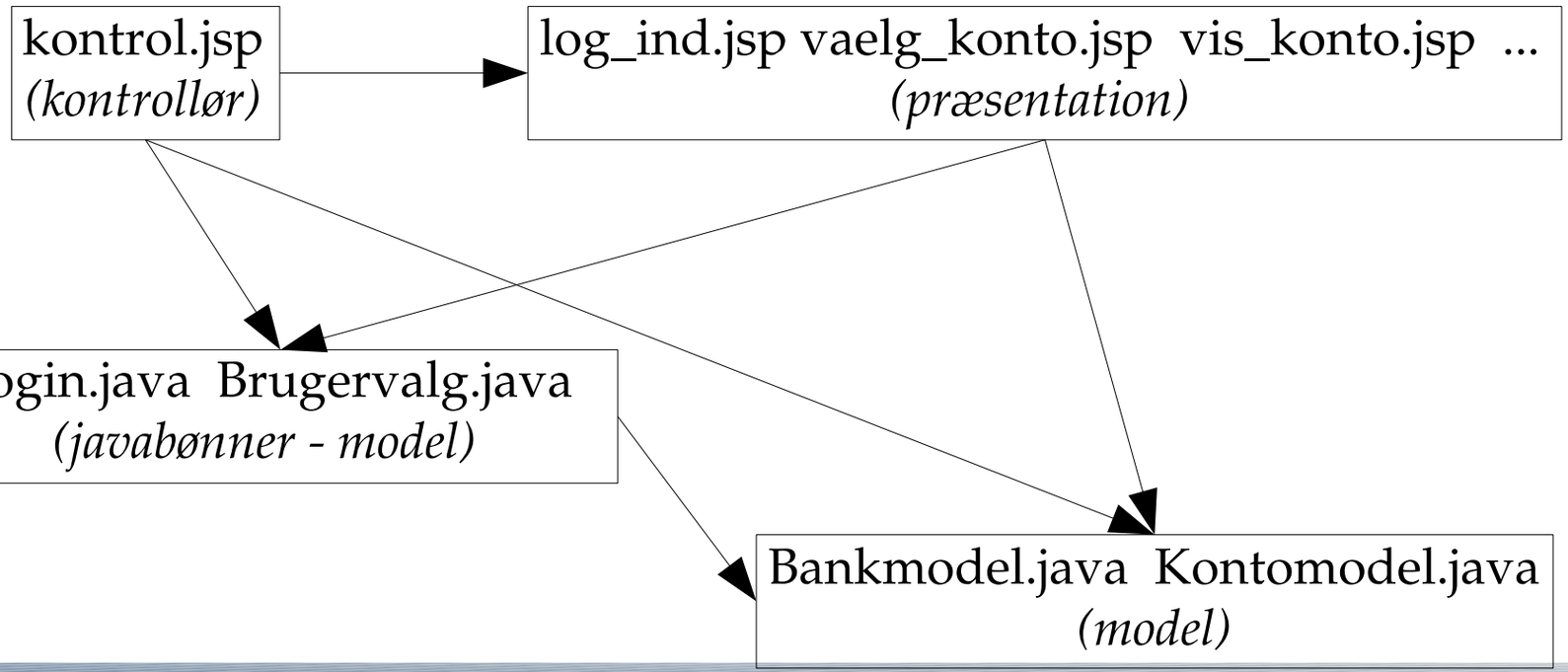
MVC-eksempel

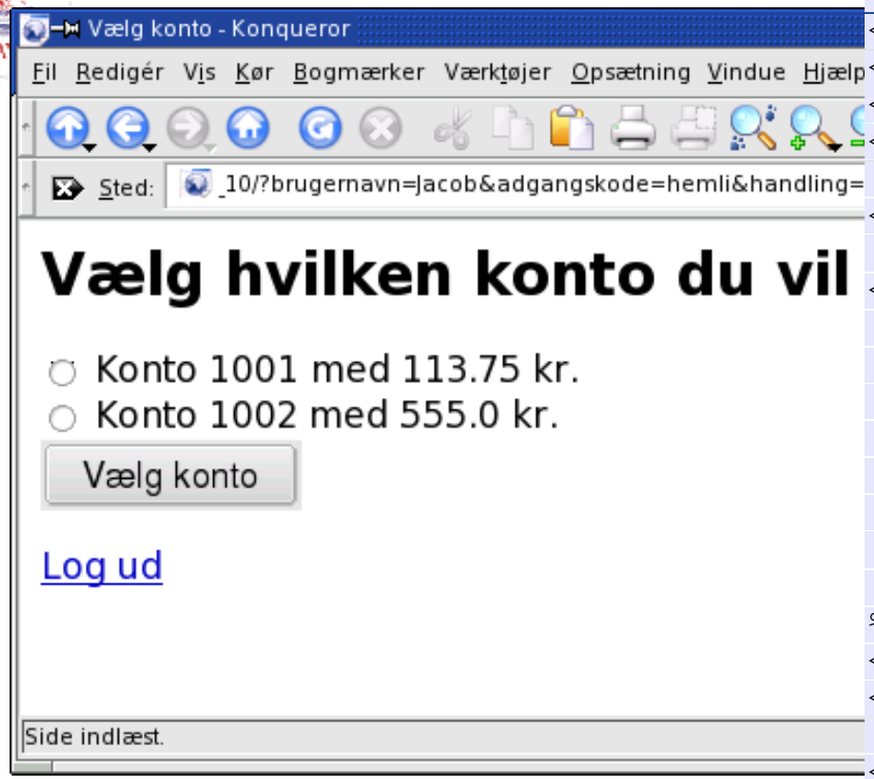



Log ind - Konqueror
Fil Redigér Vis Kør Bogmærker Værktøjer Opsætning Vindue Hjælp
Sted: http://javabog.dk:8080/JSP/kode/kapitel_10/
Log ind
Brugernavn: Jacob
Adgangskode: *****
log ind
Vink: Brugere 'Jacob', 'Preben' og 'Søren' har adgangskoden 'hemli'.
Side indlæst.


Vælg konto - Konqueror
Fil Redigér Vis Kør Bogmærker Værktøjer Opsætning Vindue Hjælp
Sted: _10/?brugernavn=Jacob&adgangskode=hemli&handling=log+ind
Vælg hvilken konto du vil bruge
Konto 1001 med 113.75 kr.
Konto 1002 med 555.0 kr.
Vælg konto
Log ud
Side indlæst.

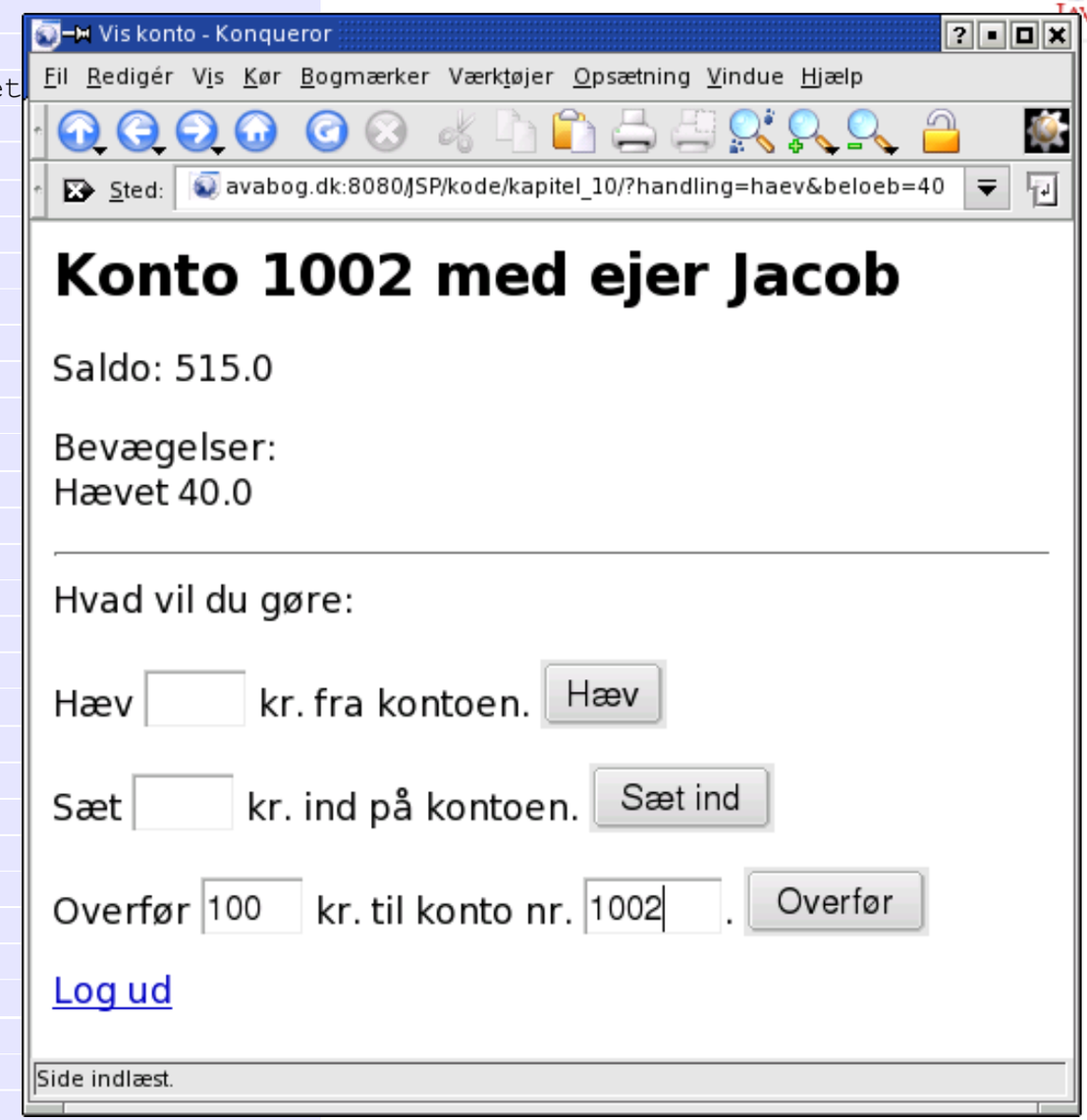

Vis konto - Konqueror
Fil Redigér Vis Kør Bogmærker Værktøjer Opsætning Vindue Hjælp
Sted: avabog.dk:8080/JSP/kode/kapitel_10/?handling=haev&beloeb=40
Konto 1002 med ejer Jacob
Saldo: 515.0
Bevægelser:
Hævet 40.0
Hvad vil du gøre:
Hæv kr. fra kontoen. Hæv
Sæt kr. ind på kontoen. Sæt ind
Overfør 100 kr. til konto nr. 1002 . Overfør
Log ud
Side indlæst.


kontrol.jsp
(kontrollør)
log_ind.jsp vaelg_konto.jsp vis_konto.jsp ...
(præsentation)
Login.java Brugervalg.java
(javabønner - model)
Bankmodel.java Kontomodel.java
(model)




```
<%@include file="logintjek.jsp" %>
<jsp:useBean id="valg" scope="session" class="bank.Brugervalg" />
<html>
<head><title>vaelg_konto</title></head>
<body>
<h1>Vælg hvilken konto du vil bruge</h1>

<form method="post" action="kontrol.jsp">

<%
  for (java.util.Iterator i=valg.konti.iterator(); i.hasNext(); )
  {
    bank.Kontomodel k = (bank.Kontomodel) i.next();
    %>
      <input type="radio" name="kontovalg" value="<%= k.getId() %>">
      Konto <%= k.getId() %> med <%= k.getSaldo() %> kr.<br>
    <%
  }
%>
<input type="submit" value="Vælg konto">
</form>

<p><a href="kontrol.jsp?adgangskode=x">Log ud</a>
</body>
</html>
```

```
<%@page import="java.util.*" %>
<%@include file="logintjek.jsp" %>
<jsp:useBean id="valg" scope="session" class="bank.Brugervalg" />
<html>
<head><title>vis_konto</title></head>
<body>
<h1>Konto <%= valg.konto.getId() %> med ejer <%= valg.konto.get
Saldo: <%= valg.konto.getSaldo() %><p>
Bevægelser:<br>
<%
  Iterator i= valg.konto.getBevægelser().iterator();
  while (i.hasNext()) {
    %>
      <%= i.next() %><br>
    <%
  }
%>
<hr>
Hvad vil du gøre:<p>

<form method="post" action="kontrol.jsp">
<input type="hidden" name="handling" value="haev">
Hæv <input type="text" name="beloeb"> kr. fra kontoen.
<input type="submit" value="Hæv">
</form>
<p>

<form method="post" action="kontrol.jsp">
<input type="hidden" name="handling" value="saet_ind">
Sæt <input type="text" name="beloeb"> kr. ind på kontoen.
<input type="submit" value="Sæt ind">
</form>
<p>

<form method="post" action="kontrol.jsp">
<input type="hidden" name="handling" value="overfoer">
Overfør <input type="text" name="beloeb"> kr. fra denne konto
til konto nummer <input type="text" name="tilKontoId">.
<input type="submit" value="Overfør">
</form>
```



kontrol.jsp

```
        import="java.util.*, bank.*" %>

<%-- Nedenstående objekt har globalt virkefelt (svarer til en singleton) --%>
<jsp:useBean id="bank" class="bank.Bankmodel" scope="application" />

<%-- Nedenstående objekter har sessionen (brugeren) som virkefelt --%>
<jsp:useBean id="login" class="javabog.Login" scope = "session"/>

<jsp:useBean id="valg" class="bank.Brugervalg" scope="session">
  <%-- Når denne bønne oprettes skal setBankmodel(bank) kaldes --%>
  <jsp:setProperty name="valg" property="bankmodel" value="<%= bank %>" />
</jsp:useBean>

<%-- Denne side er den eneste, der sætter egenskaber (ændrer på modellen) --%>
<jsp:setProperty name="login" property="*" />
<jsp:setProperty name="valg" property="*" />

<%
  String brugernavn = login.getBrugernavn();
  if (!login.isLoggetInd()) {                    // er bruger logget korrekt ind?
    application.log("Bruger "+brugernavn+" skal logge ind.");
    valg.konto = null;
    valg.konti = null;
    response.sendRedirect("log_ind.jsp");
  } else {                                       // login er ok

    if (valg.konto == null) {                    // har han valgt en af sine konti?
      // nej - find alle brugerens konti og læg dem i valg.konti
      if (valg.konti == null)
      {
        valg.konti = new ArrayList();
        for (Iterator i=bank.konti.iterator(); i.hasNext();)
        {
          Kontomodel k = (Kontomodel) i.next();
          if (k.getEjer().equalsIgnoreCase(brugernavn)) valg.konti.add(k);
        }
      }

      if (valg.konti.size() == 0) {
        application.log(brugernavn+" skal oprette konto.");
        response.sendRedirect("ingen_konto.jsp");
      } else if (valg.konti.size() == 1) {
```

fejl.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1" isErrorPage="true" %>
<jsp:useBean id="login" class="javabog.Login" scope = "session"/>
<html>
<head><title>fejl</title></head>
<body>
Der skete en fejl: <%= exception.getLocalizedMessage() %>.
<p>
Gå tilbage og prøv igen.
<%
  application.log( (Exception)exception, "Fejl for "+login.getBrugernavn());
%>

<p><a href="kontrol.jsp?adgangskode=x">Log ud</a>
</body>
</html>
```

Frontkontrol

- En Frontkontrol (eng.: Front Controller) modtager alle anmodninger og beslutter hvad der skal ske
  - kontrol.jsp er en frontkontrol. (Den er det ikke helt, fordi man kan kan rette i URLen og på den måde få fat i de andre sider uden om kontrol.jsp.)
  - redigere web.xml sådan, at alle forespørgsler dirigeres til én servlet eller JSP-side, der agerer Frontkontrol.
  - De rigtige sider kan ligge skjult for klienten i WEB-INF/-mappen

Eksempel på web.xml

```
<!-- Frontkontrol: Alt der starter med /bank sendes til kontrol.jsp -->

  <servlet>
    <servlet-name>Frontkontrol</servlet-name>
    <jsp-file>/WEB-INF/bank/kontrol.jsp</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>Frontkontrol</servlet-name>
    <url-pattern>/bank/*</url-pattern>     <!-- Bemærk: * i URL-mønster -->
  </servlet-mapping>
```

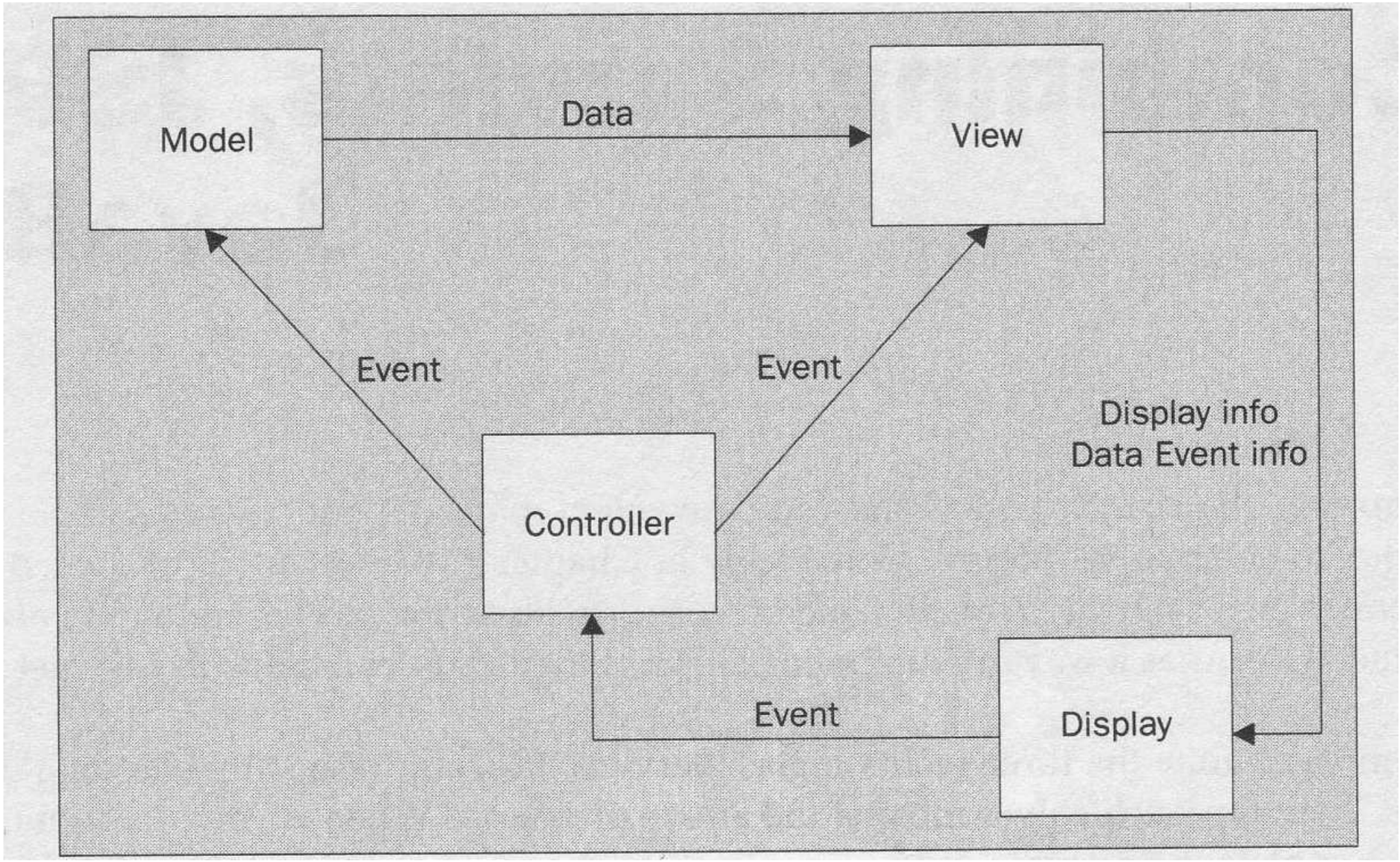
Brugervalg.java

```
package bank;
import java.util.*;
public class Brugervalg
{
  /** Egenskab 'bank' sættes fra JSP-siden kontrol.jsp når bønnen oprettes */
  Bankmodel bank;
  public void setBankmodel(Bankmodel b) { bank = b; }

  /** Liste over denne brugers konti */
  public ArrayList konti;

  /** Den konto, brugeren har valgt at arbejde med lige nu */
  public Kontomodel konto;

  /** Egenskab 'kontovalg' sættes fra JSP-side vaelg_konto.jsp*/
  public void setKontovalg(int nr) {
    konto = null;
    for (Iterator i=konti.iterator(); i.hasNext(); ) {
      Kontomodel k = (Kontomodel) i.next();
      if (k.getId() == nr) konto = k;
    }
    if (konto == null)
      throw new IllegalArgumentException("Konto-ID "+nr+" er ukendt");
  }

  /** Streng der beskriver en handling brugern ønsker at udføre */
  public String handling;
  public void setHandling(String h) { handling = h; }

  /** Beløbet handlingen (hæv/sæt ind/overfør) drejer sig om */
  double handlBeløb;
  public void setBeloeb(double b) { handlBeløb = b; }

  /** Hvis der skal foretages en overførel, hvilken konto er det til */
  Kontomodel ovfTil;

  /** Egenskab 'tilKontoId' sættes fra JSP-siden vis_konto.jsp */
  public void setTilKontoId(int nr) {
    ovfTil = null;
    for (Iterator i=bank.konti.iterator(); i.hasNext(); ) {
      Kontomodel k = (Kontomodel) i.next();
```

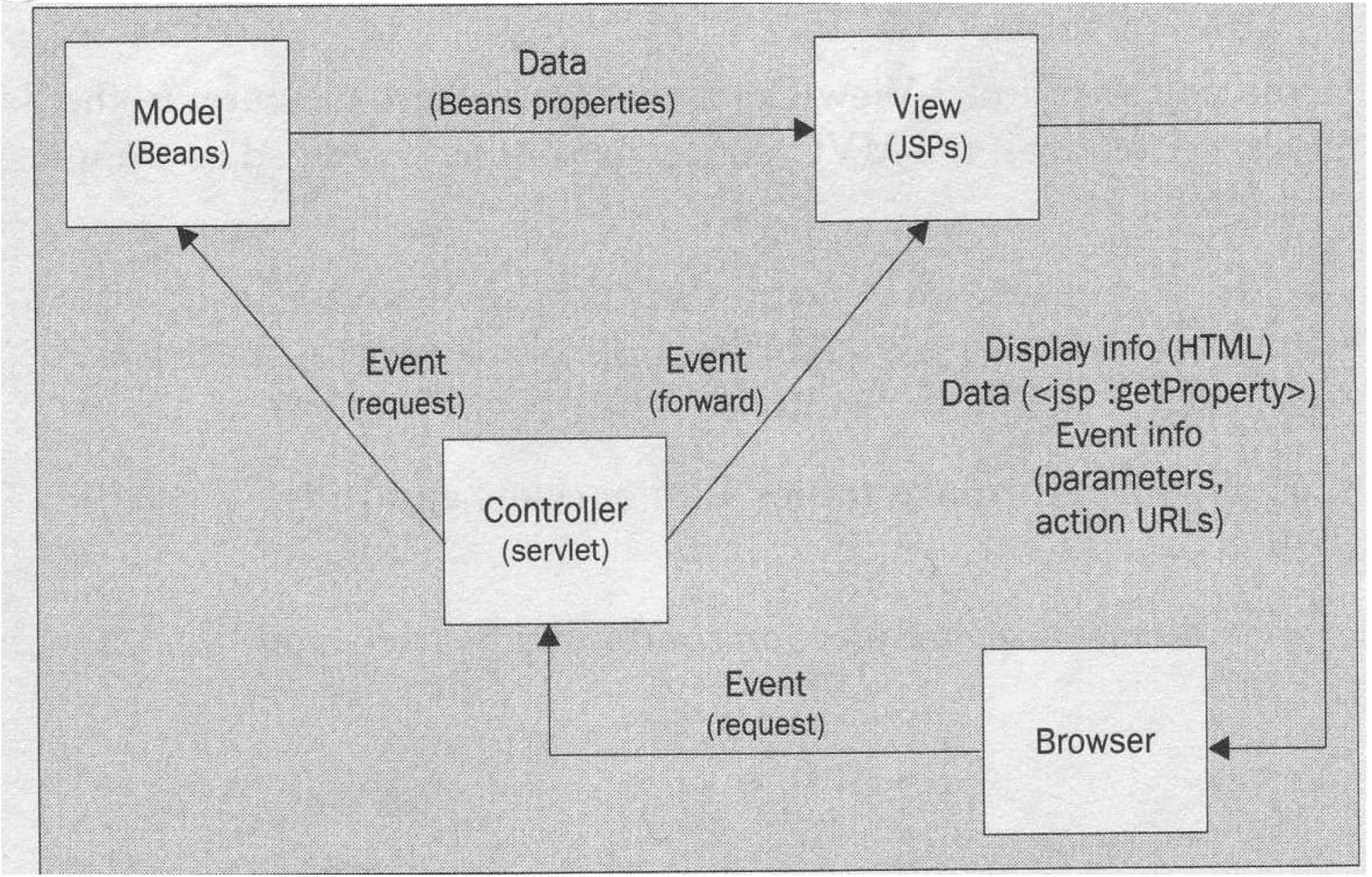

```
package bank;
import java.util.*;
public class Kontomodel
{
  private String ejer;
  private int id;
  private double saldo;
  private List bevægelser = new ArrayList();

  public Kontomodel(String ejer1, int id1) { ejer = ejer1; id = id1; }
  public Kontomodel(String ejer1, int id1, double saldo1)
    { ejer = ejer1; id = id1; saldo = saldo1; }

  public String getEjer()       { return ejer; }
  public int getId()          { return id; }
  public double getSaldo()      { return saldo; }
  public List getBevægelser() { return bevægelser; }

  public String toString() { return ejer + ": "+saldo+" kr"; }

  public void overfør(Kontomodel til, double beløb)
  {
    if (beløb<0) throw new IllegalArgumentException(
                                    "Beløb kan ikke være negativt eller nul");
    saldo = saldo - beløb;
    til.saldo = til.saldo + beløb;// privat variabel kan ændres i samme klasse

    String ændring = "Overført "+beløb+" fra "+ejer+" til "+til.ejer;
    bevægelser.add(ændring);
    til.bevægelser.add(ændring);
  }

  public void hæv(double beløb)
  {
    if (beløb<0) throw new IllegalArgumentException(
                                    "Beløb kan ikke være negativt eller nul");
    saldo = saldo - beløb;
    bevægelser.add("Hævet "+beløb);
  }

  public void indsæt(double beløb)
  {
    if (beløb<0) throw new IllegalArgumentException(
```

Lidt mere om MVC

Kilde: Professional JSP, 2. udg., Wrox Press Ltd.



MVC og implementation-teknologier

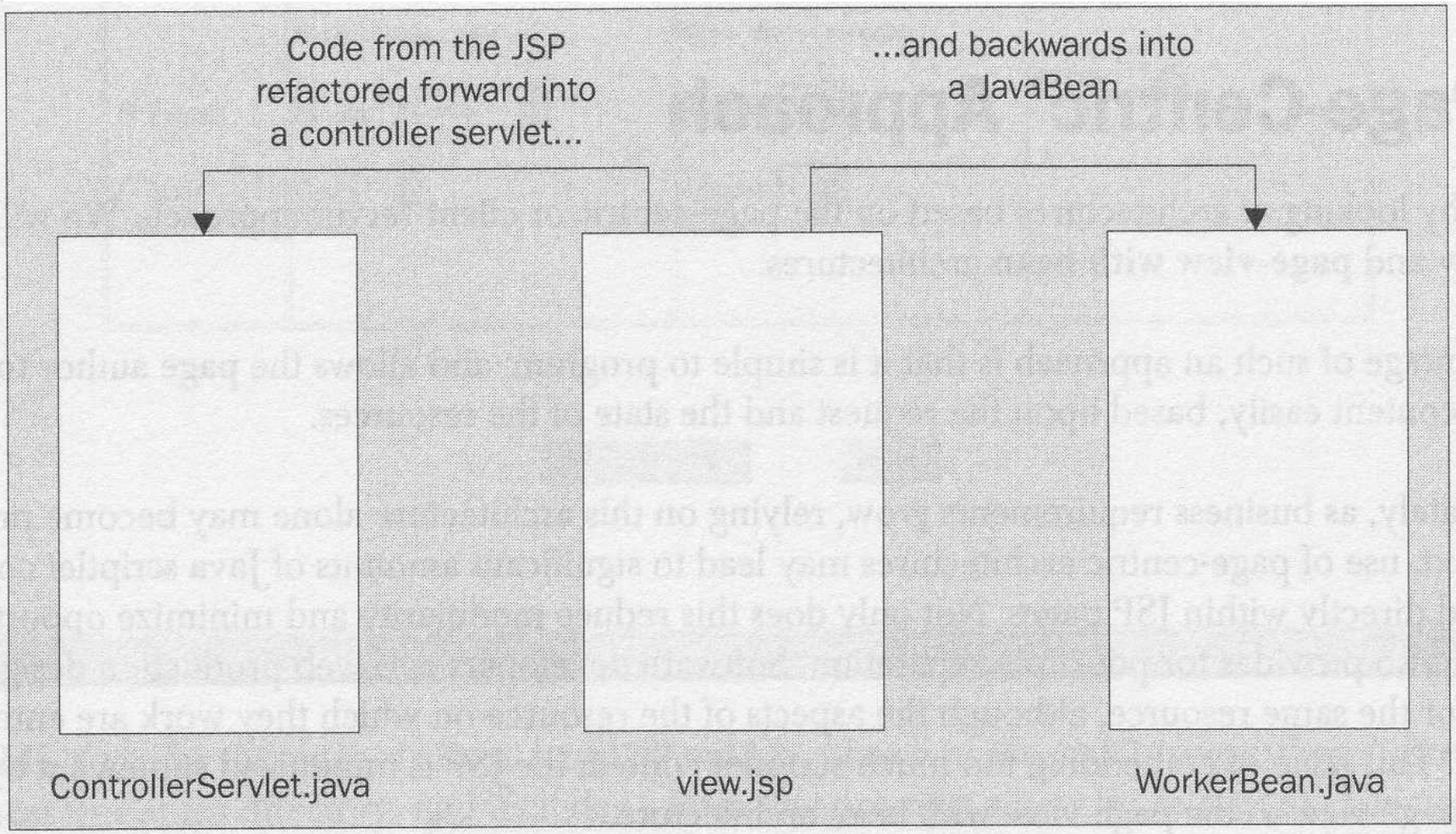


MVC

- **Model**
  - 1 - State components
    - Current model values
    - Methods to change the values (may include some business logic)
    - Implementation: Java Beans applicable
    - Protocol-independent (HTTP, RMI, ...)
  - 2 - Action components
    - Define allowable changes to state (business logic)
    - Implementation:
      - best not to put in controller
      - rather: create a layer of *action beans*
- **View**
  - The presentation logic of an application
  - Presents state from the Model
  - Provides the UI for the specific protocol (here, HTTP/browser)
  - Separation from Model allows different UI's
  - Implementation: JSP (http)
    - Interaction with model eased by using Java Beans

- **Controller**
  - 'Glue' for the MVC-architecture
  - Receiving events
  - Determining and invoking the appropriate handler
  - Invoking appropriate response (view)
  - Implementation: servlets (general programming)
  -
  - Controller acts as *dispatcher*
  - Handles these tasks:
    - Security – authentification, authorisation
    - Event identification (e.g. using parameter in request)
    - Preparing the model – ensure availability of required components
    - Processing events – invoke appropriate event handler
    - Handling errors – forward control to an error page
    - Triggering response – forward control to response generator

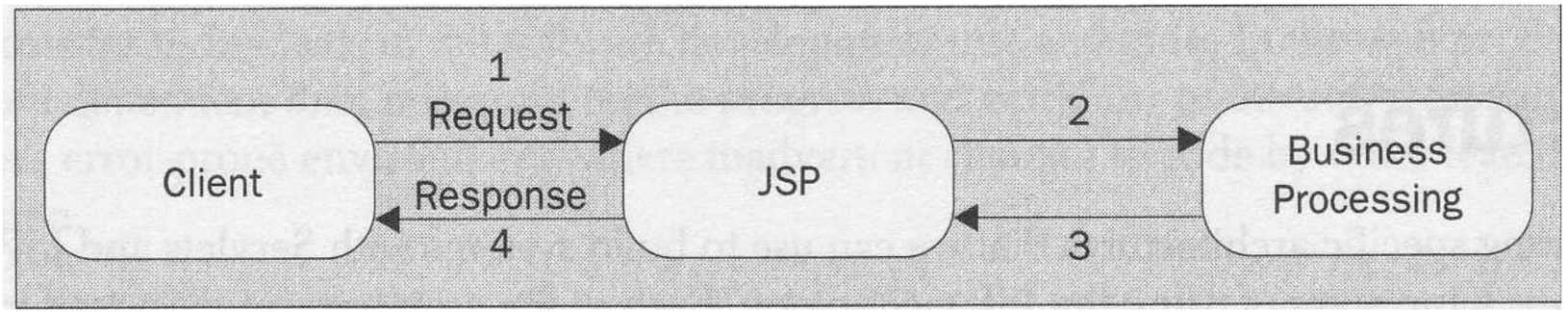
Code refactoring – Role separation



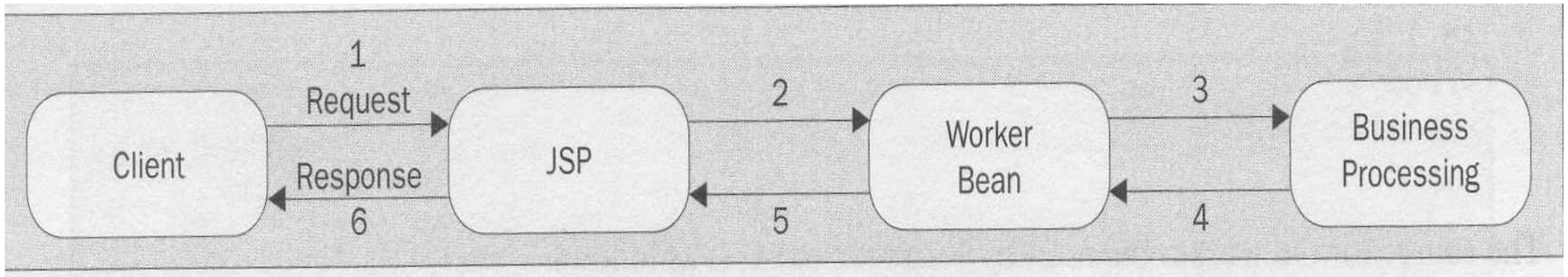
Another way to think about where code should be localized and encapsulated is that our JSP page should reveal as little as possible of our Java code implementation details.

Page-centric (Client-server) architecture

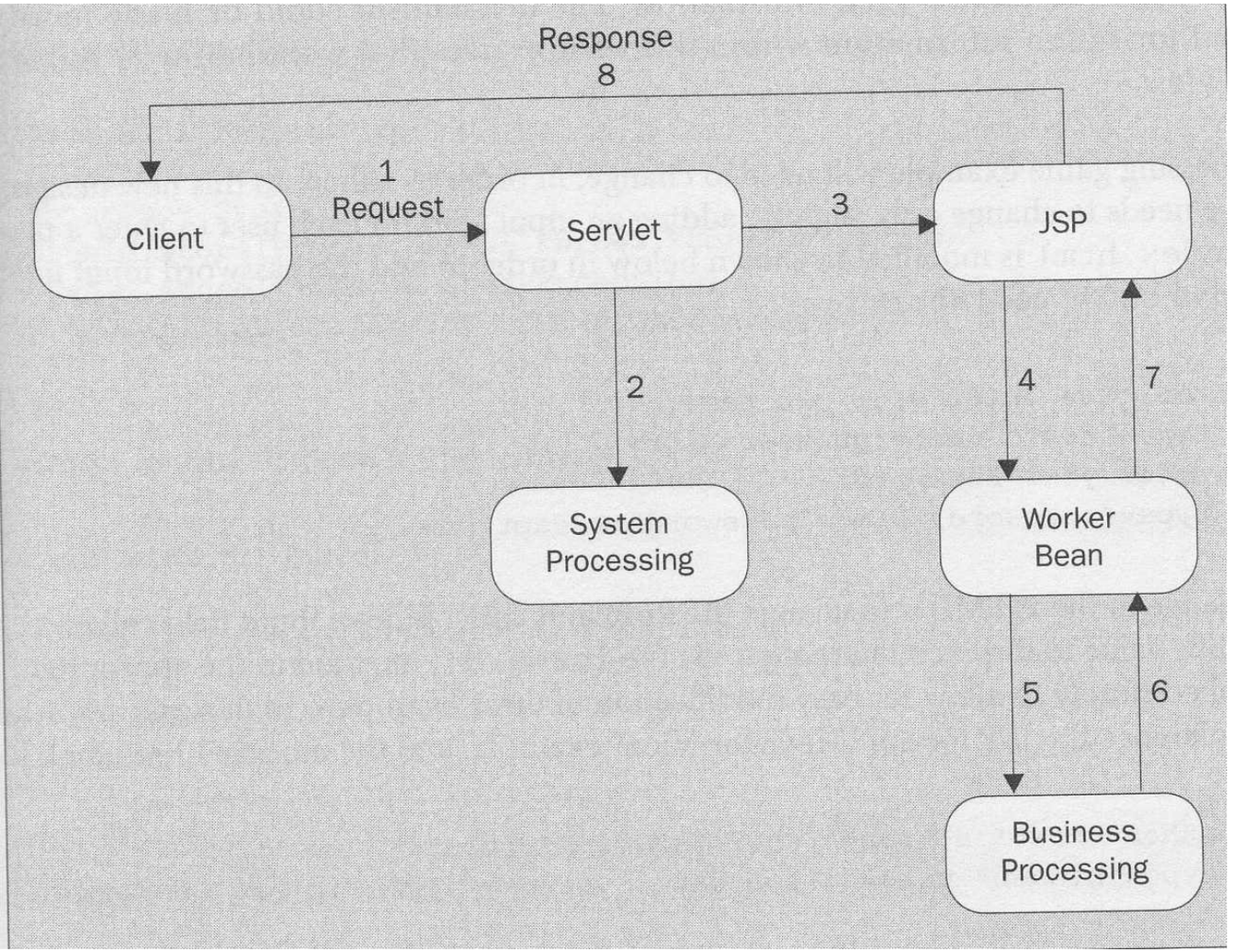
1. Simple Page-view architecture
   - easy for getting started; small applications



2. Factoring-out business logic



The 'Dispatcher' (mediator, controller) architecture



- **Mediating servlet** (controller)
  - Processes requests
  - Creates beans
  - Manges navigation (decides which JSP the request should be forwarded to next)
  - has common tasks: e.g. authentification
- **JSP (presentation)**
  - Has no processing logic
  - Retrieves objects/beans
  - Presents next view