JSP dizajn antipaterni

ili kako ne treba programirati

•••

MATEMATIČKI FAKULTET, UNIVERZITET U BEOGRADU
PROFESOR: PROF.DR.VLADIMIR FILIPOVIĆ
STUDENT: ANA SIMIJONOVIĆ
BEOGRAD, 27.09.2016.

Uvod

- Loše prakse pri korišćenu JSP strana u MVC arhitekturi
- JSP zaduženja
- Javina zaduženja
- Prepuštanje modela tag-jeziku ili jeziku na višem nivou?

Mane korišćenja tag-jezika na server strani

- Model usko vezan sa pogledom (održavanje i dodavanje funkcionalnosti teško zato što model ne može biti izolovan od pogleda)

- Logika modela se održava u tag-jeziku ili skripti (Java je superiornija, fleksibilnija, robusnija u odnosu na tag jezik kao jezik višeg nivoa)

Mane korišćenja tag-jezika na server strani

- Zaduženja nisu razdvojena (programeri moraju biti dobri dizajneri, i dizajneri dobri programeri)

- Nakon početne implementacije, izbori za implementaciju arhitekture modela su ograničeni

- Efikasnost počinje da opada, strane postaju sve teže za editovanje, i model i UI postaju kompleksniji sa svakom revizijom i ne mogu se refaktorisati lako, sve to zahvaljujući tom usko povezanom UI i modelu
- UI redizajniranje zahteva dosta više vremena od očekivanog
- Uska povezanost UI i modela sele logiku modela i kontrolera u pogled

Teški JSP-ovi

- Čak i kada je programer iskusan, i zna za postojanje ovakvih ili sličnih antipaterna, dešava se da u nekom momentu pođe stranputicom
- Deluje da bi programeru bilo lakše da piše samo JSP sa skriptama nego JSP, komandni bean i kontroler
- Naravno, ukoliko se odlučimo za drugo rešenje, to se na kraju mnogo više isplati

Primer teške JSP strane

```
<%@ page import="java.sql.*" %>
<%
  // instance variables for connection
  ResultSet result;
  Connection connection = null;
  Statement statement = null;
  String url = "jdbc:db2:board";
%>
```

❶ **Model specific initialization**

❷ **View specific HTML**

```
<HTML>

<HEAD>
<TITLE>Message Board Posts</TITLE>
</HEAD>

<BODY BGCOLOR=#C0C0C0>

<H1>All Messages</H1>
<P>

<TABLE border="1">

<TR>
  <TD>Subject</TD>
  <TD>Author</TD>
  <TD>Board</TD>
</TR>
```

<% ❸ **Model specific database access**

```
// Establish a connection
try {
  Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

  // connect with default id/password
  connection = DriverManager.getConnection (url);

  // set and execute SQL statement
  statement = connection.createStatement();
  result = statement.executeQuery
              ("SELECT subject, author, board from posts");

  // print the results
  // retrieve data from the database and print on the result page

  while (result.next()) {
      out.println("<TR> <TD>" + result.getString(1) + "</TD>");
      out.println("<TD>" + result.getString(2) + "</TD>");
      out.println("<TD>" + result.getString(1) + "</TD></TR>");
  }
  result.close();
  statement.close();
  connection.close();
} catch (Throwable theException) {
  out.println("Connection or output print failed.");
}
```

```
</TABLE>
<P>
</BODY>
</HTML>
```

❹ **View specific HTML**

- Specifikacija modela unutar <% %>, zatim specifikacije pogleda (html), i zatim opet model, pa opet pogled

- U prethodnom primeru vidi se i lako se misaono izdvaja UI, ali je poimanje modela mnogo otežano

- Prosto, tag-jezik ne dovoljava modelu da bude strogo i čisto odvojen, u meri u kojoj bi to Java dozvolila

Rešenje

- Kako da se to poboljša i iskoristi u svoju korist?
  - Razdvojiti model i view bolje

- Napraviti klase u Javi, u JSP-u bean-tag unutar body taga, onde gde je stajao dinamički sadržaj;
  - u kontroleru izvršiti sve potrebne komande i poslati JSPu kao odgovor

- Korišćenjem ovog metoda, teške JSP strane refaktorišemo i pretvaramo u pravi MVC dizajn patern

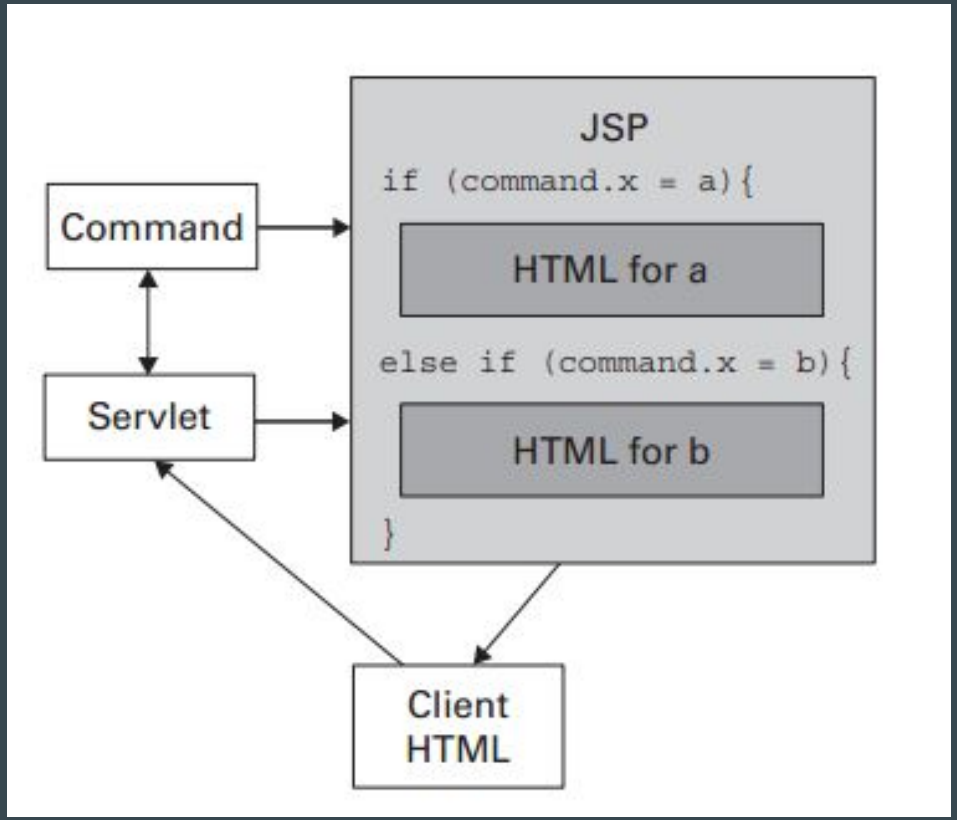
- Pokidamo kod na tri dela: JSP za view, komande na model i servlet za kontroler

- Za timove koji grade puno UI, može pomoći izgrađivanje uzorka i obrasca koji može da se koristi za ove tipove i UI

Sjedinjene JSP strana

- Još jedna loša praksa sa kojom se programer može susresti pri korišćenju JSP strana
- Ovde se MVC arhitektura ne poštuje na drugi način
- Ukoliko se za dizajn izbere dobar patern, ne znači da se više ne mora voditi računa o arhitekturi i njenoj pravilnoj implementaciji
- Pitanje koje ovde postavljamo jeste da li forsirati donošenje odluka u JSP-u ili kontroleru, kada je odluka bitna kao npr. koja će se strana prikazati korisniku ukoliko je neki izraz ispunjen

- Antipatern sjedinjenje JSP strana, u kome jedna JSP strana vraća puno različitih rezultata - previše odluka se dovodi u JSP-u
- Sjedinjenje JSP strane ima manje bočnih efekata od prethodnog antipaterna, ali je svakako bolje da se iz početka refaktoriše, nego da se to ostavlja za kasnije
- Lakši su za kreniranje, ali je i dalje održavanje, ponovno korišćenje, dizajniranje sa odmicanjem vremena sve teže i teže


JSP
if (command.x = a){
HTML for a
else if (command.x = b){
HTML for b
}
Command
Servlet
Client
HTML

- Nema razdvajanje zaduženja
  kombinovanje koda i HTML skripti zahteva programera koji ima dobre kako programerske, tako i dizajnerske veštine
- Nemogućnost ponovnog korišćenja
  samim tim što sadrži u sebi kontrolnu logiku
- Nedostatak validacije grešaka i oporavak od istih
  u ovakvom dizajnu softvera, teško je uključiti dobro lečenje i prikazivanje grešaka

Primer sjedinjenih JSP-ova

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">

<HTML>
<BODY>
  <FORM METHOD="post" ACTION="/servlet/bbs.CompoundJSPController">
  <P>Please complete the form.</P>
  <P>board
   <BR>

   <INPUT TYPE="text" NAME="board" ID="board" SIZE="20"
      MAXLENGTH="20">
  </P>
  <INPUT TYPE="hidden" NAME="mlname" ID="mlname" VALUE="HTML">
  <P>
  <INPUT TYPE="submit" NAME="Submit" ID="Submit" VALUE="Submit">

  <INPUT TYPE="reset" NAME="Reset" ID="Reset" VALUE="Reset">

    
  </P>
  </FORM>
</BODY>
</HTML>
```

```
package bbs;

// Imports
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Import for commands used by this class
import bbs.CompoundJSPCommand;

public class CompoundJSPController
  extends javax.servlet.http.HttpServlet
  implements Serializable {
```

```
/**
 * DoPost
 * Pass post requests through to performTask
 */
public void doPost(
  HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
    performTask(request, response);
}
```

```
/**************************************************************
* Process incoming requests for information
*
* @param request encapsulates the request to the servlet
* @param response encapsulates the response from the servlet
*/
public void performTask(
  HttpServletRequest request,
  HttpServletResponse response) {

  try {
    String board=request.getParameter("board");
    CompoundJSPCommand postList = new CompoundJSPCommand();
    postList.setBoard(board);
    postList.initialize();
    postList.execute();
```

```
    request.setAttribute("CompoundJSPCommand", postList);

    ServletContext servletContext = getServletContext ();
    RequestDispatcher dispatcher =
      servletContext.getRequestDispatcher("/JSP/CompoundJSPResults.jsp");
    dispatcher.forward(request, response);

  } catch (Throwable theException) {
    theException.printStackTrace();
  }
}

  /**
   * DoGet
   * Pass get requests through to performTask
   */
```

```
<HTML>

<jsp:useBean id="CompoundJSPCommand"  class="bbs.CompoundJSPCommand"
   scope="request"></jsp:useBean>

<% if (CompoundJSPCommand.getSize() == 0) { %>
<HEAD>
<TITLE>Choose another board! </TITLE>
</HEAD>
<p>
```

❶ **Decision 1**

```
There were no posts for board <%=CompoundJSPCommand.getBoard()%>

<BODY BGCOLOR=#C0C0C0>
  <FORM METHOD="post" ACTION="/servlet/bbs.CompoundJSPController">
<P>Please complete the form.</P>
<P>board
      <BR>

  <INPUT TYPE="text" NAME="board" ID="board" SIZE="20" MAXLENGTH="20" >
</P>
<INPUT TYPE="hidden" NAME="mlname" ID="mlname" VALUE="HTML">
<P>
  <INPUT TYPE="submit" NAME="Submit" ID="Submit" VALUE="Submit">

  <INPUT TYPE="reset" NAME="Reset" ID="Reset" VALUE="Reset">

    
</P>
</FORM>
```

```
<% } else { %>
```

❷ **Decision 2**

```
<HEAD>
<TITLE>Message Board Posts</TITLE>
</HEAD>

<BODY BGCOLOR=#C0C0C0>
<H1>All Messages</H1>
<P>

<p>
Board: <%=CompoundJSPCommand.getBoard()%>
<TABLE border="1">

<TR>
  <TD>Subject</TD>
  <TD>Author</TD>
</TR>
      <% for (int _i=0; _i < CompoundJSPCommand.getSize(); _i++) { %>
      <TR> <TD><%=CompoundJSPCommand.getSubject(_i) %></TD>
      <TD><%=CompoundJSPCommand.getAuthor(_i) %></TD>
      </TR>
      <% } %>
</TABLE>
<P>
<% } %>

</BODY>
</HTML>
```

Rešenje

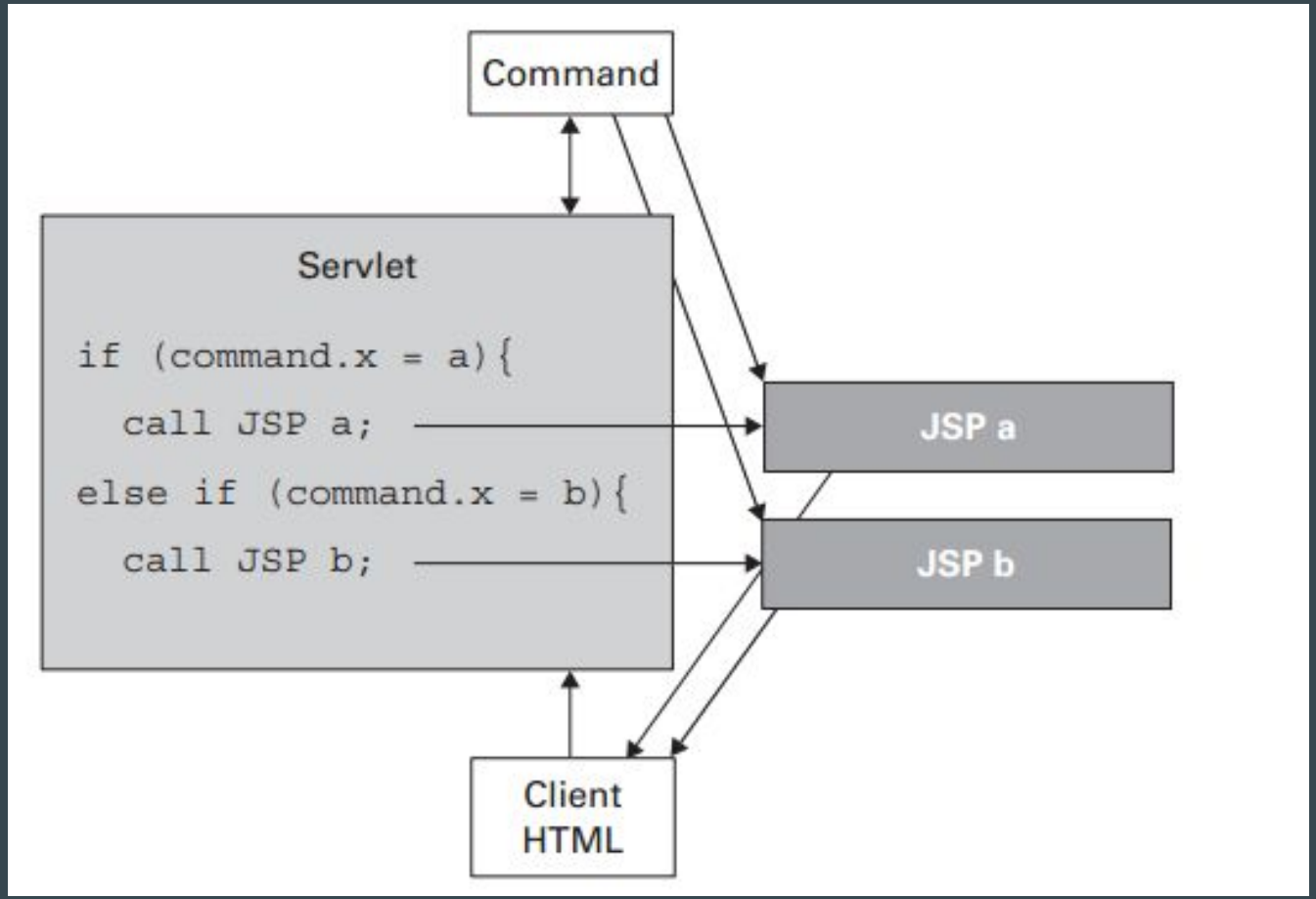
- Razdvojiti sjedinjeni JSP u dva fajla
- Premestiti trenutak odluke u kontroler
- Time:
  - komandni deo se neće promeniti
  - očuvano je razdvajanje zaduženja
  - mnogostruko je upotrebljivije

```
String resultsPage;
if (postList.getSize() > 0) {
  resultsPage = "/JSP/BoardResults.jsp";
} else {
  resultsPage = "/JSP/NotFoundResults.jsp";
}
RequestDispatcher dispatcher =
servletContext.getRequestDispatcher (resultsPage);

dispatcher.forward(request, response);
```

❶ Decision 1

❷ Decision 2


Command
Servlet
if (command.x = a){
call JSP a;
else if (command.x = b){
call JSP b;
JSP a
JSP b
Client
HTML

- Kako refaktorisati sjedinjene JSP strane
  - Razdvojiti JSPove u različite strane, bez suvišnog Java koda u njima
  - Pomeriti trenutak odluke u kontroler
  - Odluku pomeriti posle izvršavanja komande a pre slanja rezultata
  - Poslati rezultat tačnom JSPu kome treba

- Refaktorisanje ne bi trebalo biti teško ukoliko je prethodno loše rešenje zasnovano na odlukama na komandnom sloju

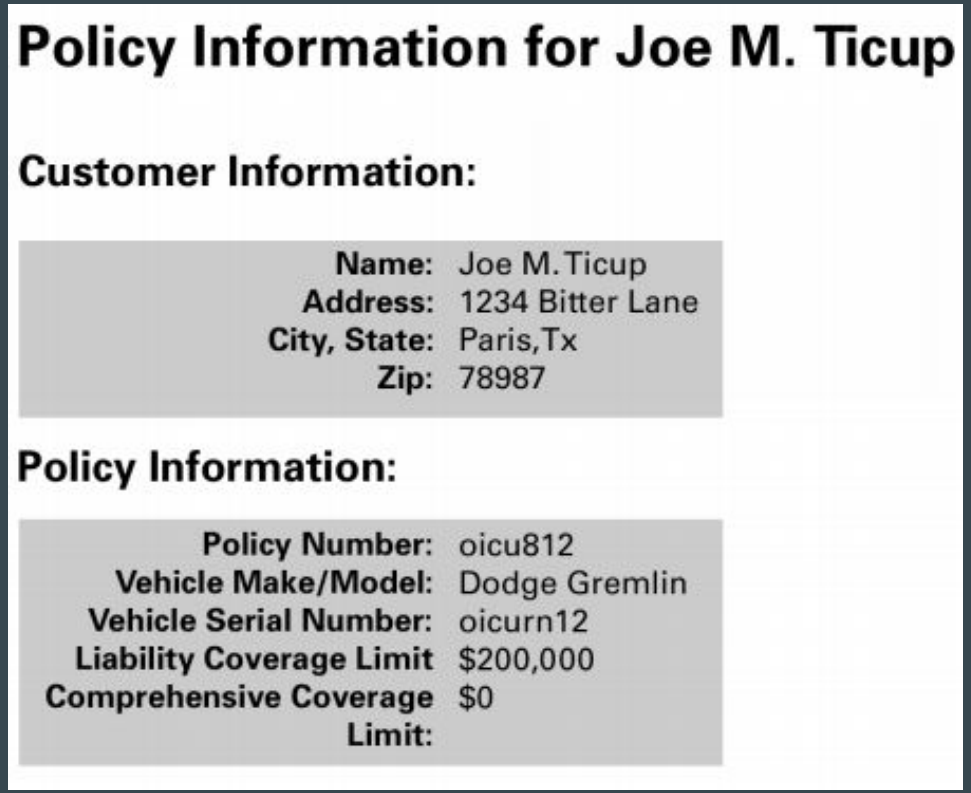
Grubo i fino deljenje komandi u grupe

- Programerima koji se tek susreću sa Komandnim dizajn paternom često se dešava da greše pri deljenju modela u komande, zato što je za takav posao potrebno iskustvo

- Ukoliko je granualnost previše gruba, ponovno korišćenje je nemoguće

- Ukoliko je granualnost previše fina, određene prednosti poput redukcije u tzv. Kružnom povezivanju koda, nisu moguće

Primer



Policy Information for Joe M. Ticup

Customer Information:

Name:	Joe M.Ticup
Address:	1234 Bitter Lane
City, State:	Paris,Tx
Zip:	78987

Policy Information:

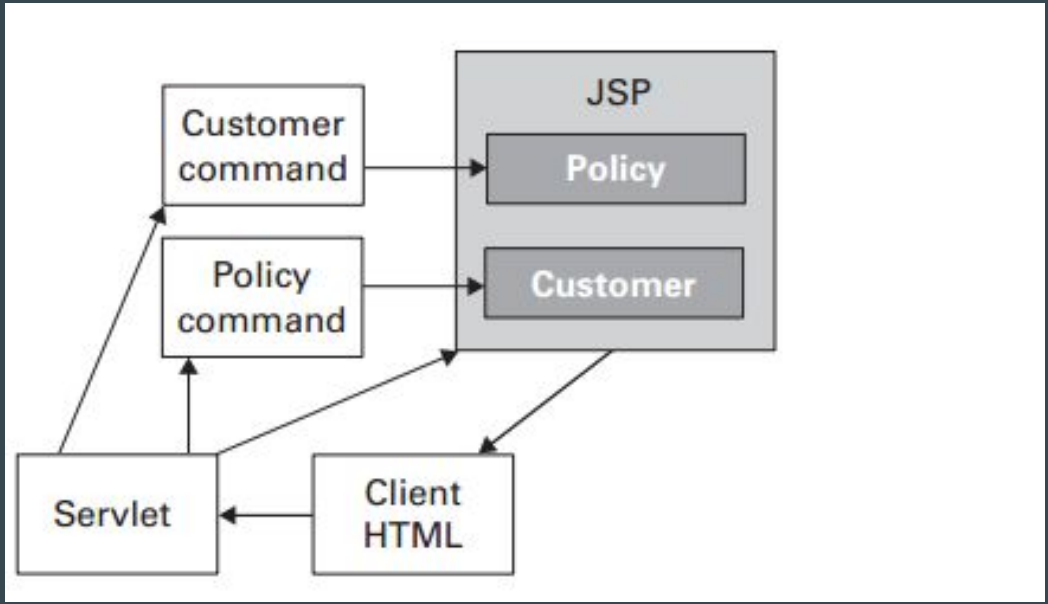
Policy Number:	oicu812
Vehicle Make/Model:	Dodge Gremlin
Vehicle Serial Number:	oicurn12
Liability Coverage Limit	$200,000
Comprehensive Coverage Limit:	$0

- Da li da izaberemo da pokupimo sve informacije o polisi osiguranja zajednmo sa dodatnim podacima o korisniku ili odvojiti ta dva zahteva za prikazivanje na UI?

- Ukoliko ih sasvim razdvojimo tako da pretpostavimo da će nam svaki put kada nam zatreba polisa osiguranja, trebati i informacije o korisniku, dobijamo problem grube granualnosti - time se okrećemo više ka UI, a ne ka modelu kako bi trebalo

- Komadni sloj je organizovan oko UI pa je redizajniranje UI teže, u odnosu na slučaj da su komande grupisane po nekoj logici

Rešenje


JSP
Customer command
Policy
Policy command
Customer
Servlet
Client HTML

- Ukoliko su grupe komandi korišćene i indivualno i paketno, koristiti pakete individualnih komandi
- Sasvim je u redu koristiti komande koje vraćaju nešto više informacija nego što je vama potrebno, što sprečava stalno refaktorisanje
- Ukoliko se slične komande koriste stalno, refaktorisati tako da se kombinuju u grupu

Zaključak

- Programiranje u Javi se može znatno zakomplikovati i postati gorko, ali znajući korake koje treba preduzeti u tom slučaju, problem se brzo rešava uz par ožiljaka

Hvala na pažnji!