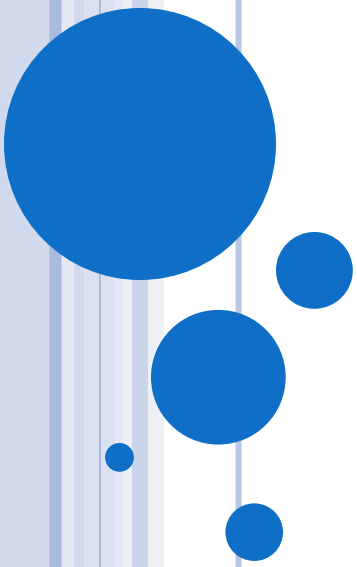


Bitter Servlets

Mladen Lazić, 1088/2015

Pogrešan početak

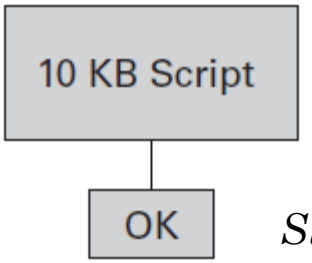
- Osnovni gejtvej za većinu neotpornih Java aplikacija na strani servera je **servlet**. On predstavlja jednostavan omotač oko servisa koji je implementiran u Javi i kome se pristupa preko HTTP-a.

- Projekti koji su nespretno implementirali ovaj prvi korak imaju malo šanse za uspeh.

Rani antipatern: Magično dugme

- Prilikom proučavanja ovog anitpaterna uočava se isto napravljena aplikacija na mnogo različitih mesta.
- Predstavlja najjednostavniji antipatern: Nestruktuirana aplikacija zavisi od jedne procedure koja je inicirana nekim događajem interfejsa.
- Dugme je „magično“ jer rado većinu posla bez ijedne funkcije.
- Povezan je sa grafičkim alatima koji su počeli da se koriste u programiranju 1990-ih godina.
- Predstavlja najgori oblik programiranja vođenog događajima.

Rani antipatern – Magično dugme



Slika 1. Magično dugme

- Ako logika nije dobro definisana tada aplikacija može lako da usvoji interfejs kao dizajn. Skriptovi koji su programirani u obliku osnovnih funkcija su napravljeni u obliku akcija koje se iniciraju pritiskom na dugme smešteno unutar interfejsa. Najveći događaji, kao što su OK ili Submit obično privlače najviše pažnje i postaju glavna tačka programa. Bez pravilne organizacije ono što što iniciraju može rasti nekontrolisano i raditi previše stvari.

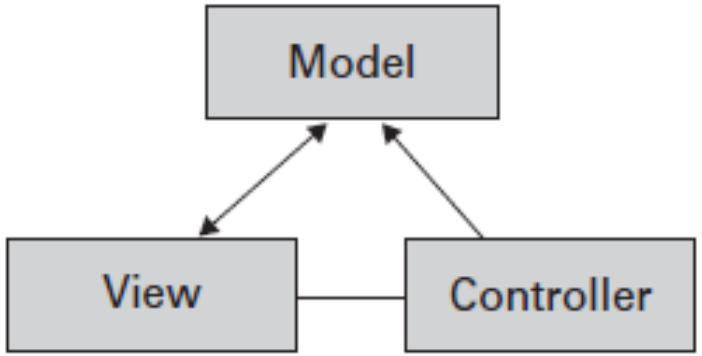
Razvoj korišćenjem uzorka Model-View-Controller

- Najbolji način za dizajniranje aplikacija ovog tipa je korišćenje najpoznatijeg i najranijeg uzorka za projektovanje koji se zove *Model-View-Controller* (MVC).

- Kod ovog uzorka **modeli** predstavljaju poslovnu logiku pomoću koje se upravlja aplikacijom. Korisnički interfejsi se nazivaju **pogledi** i predstavljaju promenljive aspekte modela. Vezom između korisnika i ostalih ulaza i izlaza se upravlja pomoću **kontrolera**.

Mane razdvajanja modela i pogleda

- MVC je uzorak koji promoviše jasno razdvajanje korisničkog interfejsa (*pogleda*) i poslovne logike (*modela*). Kotroler pomaže u rukovanju ulazima i izlazima.
- Prednosti su lakše razumevanje koda, poboljšana fleksibilnost i čitljiviji kod.



Slika 2. Model-View-Controller

```
import java.util.*;
import java.text.*;

public class BitterAccount {

  public String balance;
  public String fieldValue;
  public int radioButtons;
  public float balanceNumber;

public BitterAccount() {
    balance="0";
  }

public String buttonPressed() {
    radioButtons = userInterface.getRadioState();
    if (radioButtons == 1) {
      fieldValue = userInterface.getEntryField();
      balanceNumber = balanceNumber +
        Float.valueOf(fieldValue.trim()).floatValue();

      balance = Float.toString(balanceNumber);
      return balance;
    }  else if (radioButtons == 2) {
      fieldValue = userInterface.getEntryField();
      balanceNumber = balanceNumber -
        Float.valueOf(fieldValue.trim()).floatValue();

      balance = Float.toString(balanceNumber);
      return balance;
    } else if (radioButtons == 3) {
      return balance;
    }
  }

}
```

View logic

Model logic

View logic

Model logic

View and model logic

Model logic

View logic

Model logic

View and model logic

Model logic

View logic

Model logic

Slika 3. Implementacija računa u banci bez korišćenja MVC uzorka

```
public class Account {
 private float balance;
 public Account(float openingBalance) {
    setBalance(openingBalance);
  }
  public void setBalance(float amount) {
    this.balance=amount;
  }
  public float(get.balance) {
    return this.balance
  }
  public float debit(float amount) {
    setBalance(this.balance-amount);
    return (this.balance);
  }
  public float credit(float amount) {
    setBalance(this.balance+amount);
    return (this.balance);
  }
}
```

● **Model logic**

Slika 4. Implementacija računa u banci sa korišćenjem MVC uzorka

- Ovakvim refaktorisanjem se dobija čitljiviji kod, koji je lakše menjati i proširivati. Upravljanje akcijama koje se iniciraju događajima je ostavljeno za kontrolere.

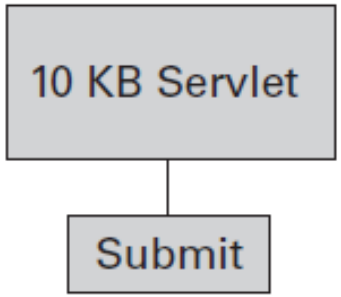
Antipatern: Magični servlet

- **Magični servlet** je najčešće korišćen Java antipatern na serverskoj strani.
- Prilikom korišćenja MVC uzorka za projektovanje glavni cilj je da se napravi jasno razdvajanje između modela podataka i pogleda. Kontroler uspostavlja vezu između njih tako da ni jedan nema informaciju o internim detaljima drugog.
- Kada se arhitektura podeli na klijent i server, lakše se pravi razlika između modela i pogleda- na klijentskoj strani je smeštena cela logika pogleda, a na serverskoj strani su modeli. Kontroleri su podeljeni, a većina njihove implementacije je na serverskoj strani i unutar softvera za podršku.

- Za razliku od većine klijent-server interfejsa, HTML ima paketno orijentisanu arhitekturu. Iz HTML koda klijenta šaljemo zahteve i čekamo odgovore. Kada je zahtev obrađen dobija se povratni interfejs koji je različit od početnog korišničkog interfejsa. Na žalost, mnogi programeri uglavnom se koncentrišu na razdvajanje početnog HTML pogleda, potpuno zaboravljajući da generišu povratni interfejs. Servlet u tom slučaju postaje zamršena gomila modela i povratnih pogleda.

Korišćenje servleta kao modela?

- U sledećem primeru servlet pruža interfejs servisa na serverskoj strani. Kada se pritisne **Submit** prvo se poziva neka forma servleta.



Slika 5. Magični servlet

- Programeri primećuju da podelom na HTML i Java servlet dolazi do jasnog razdvajanja na korisnički interfejs i poslovnu logiku.

  primer HelloWorld.java

Upadanje u zamku „Magičnog servleta“

- U boljim dizajnima, servlet je jedina tačka interfejsa između logike na serverskoj strani i korisničkog interfejsa. Na žalost, monolitne serverske skripte koje rade skoro sve su i dalje čest izbor.
- Sledeći program prikazuje Java program koji je uzet iz stvarnog života i koristi se za štampanje strana za neku oglasnu tablu. Logika servleta je potpuno odvojena od HTML-a (koji nije prikazan), ali krije velike probleme.

  primer PostList.java

Problem	Diskusija
Ne postoji razdvajanje	Korisnički interfejs i model održavaju različiti ljudi, a model i pogled su veoma povezani.
Teško održavanje	Izolovane promene unutar modela ili pogleda mogu lako da izazovu značajne promene u ostalim delovima sistema.
Loša pouzdanost	Pošto promene ne mogu da budu izolovane, jedna promena može da uništi ostale delove sistema.
Slaba ponovna upotrebljivost	Posto aplikacija nije struktuirana, ne postoje logične jedinice koje je moguće ponovo upotrebiti unutar aplikacije. Umesto toga, ponovna upotrevljivost se ostvaruje kopiranjem delova koda.

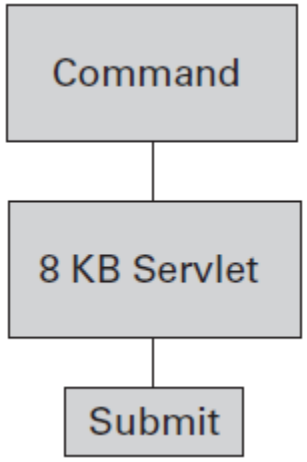
Tabela 1. Česti problemi do kojih dolazi zbog upotrebe „Magičnih servleta“

Rešenje: Refaktorisanje korišćenjem komandi

- Težak deo kod antipaterna je njihovo prepoznavanje. Kada je identifikovan antipatern sledeći korak je refaktorisanje ili primena odgovarajućeg uzorka za projektovanje.

- Biće opisan primer refaktorisanja, korak po korak. Svakim korakom će se refaktorisani kod približavati Model-View-Controller uzorku za projektovanje.

Razbijanje modela



Slika 5. Razbijanje komandi servleta

- Razbijanje modela je prvi korak prilikom refaktorisanja „Magičnog servleta“. **Komanda** je uzorak za projektovanje koji omogućava jasno pravljenje omotača oko poslovne logike ili modela.
- Napravljen je kao model za program. Cela poslovna logika ili njeni delovi su smešteni unutar servleta.
- Napravljen je kontroler za program. Rukuje ulaznom formom.
- Proveravaju se ulazni podaci.

Pravljenje omotača modela sa objektima komandi

- Komanda predstavlja tanak omotač modela. Njihova upotreba je dobra jer nije bitna implementacija koja se nalazi ispod sloja komande. Interfejs je isti bez obzira na mesto pristupanja.
- Arhitektura komande ne uzima u obzir nikakve pretpostavke o strukturi modela, što dovodi do sledećih prednosti:
  - Komanda može biti generisana upotrebom „čarobnjaka“
  - Generičke komande mogu da budu nasleđene tako da enkapsuliraju upravljanje pozivima procedura.
  - Pogodne su za enkapsuliranje undo/redo arhitekture
  - Mogu da sadrže višestruke zahteve
  - Interfejs je uvek isti i to olakšava čitanje i održavanje koda

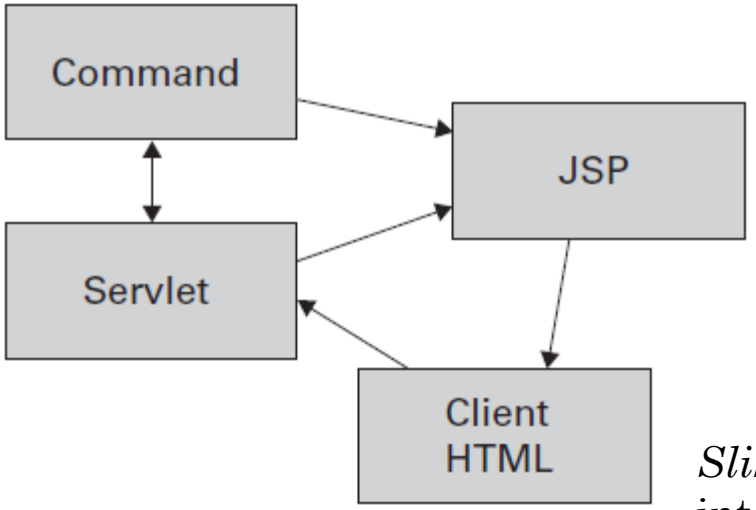
Odvajanje logike modela

- Sledeći program pokazuje objekt komandi koji vraća slogove iz tabele.

- *primer* PostListCommand.java

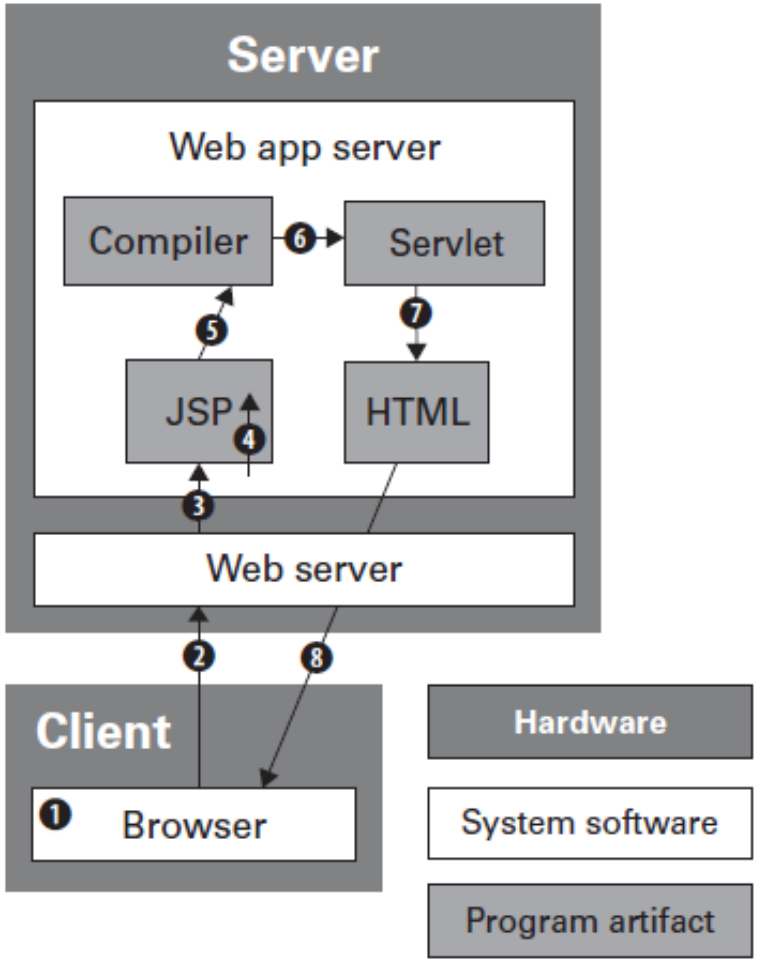
Odvajanje povratka

- Interfejsi koji se koriste na internetu su paketno orijentisani. Interfejs za Model-View-Controller je interaktivan. Postoje dve različite komunikacije unutar modela: inicijalni zahtevi i povratak. Pošto je povratna strana dinamički napravljena neophodno je da bude eksplicitno predstavljena u arhitekturi.



Slika 6. Izgled MVC arhitekture za internet aplikacije

- HTML strana koja se vraća korisniku je napravljena preko **JSP**-a. On predstavlja serversku stranu koja je nastala od HTML-a. JSP dozvoljava da Java kod i pokazivači na dinamički sadržaj budu dodati na stranu servera pored postojećih HTML tagova. JSP se kompajlira i izvršava na serveru. Rezultat izvršenog JSP-a je HTML stranica koja se vraća klijentu.
- Ovakva arhitektura pruža sledeće prednosti:
  - JSP podržava dinamički sadržaj
  - Jasno razdvajanje
  - JSP može da se koristi za izolovanje pogleda od kontrolera i modela
  - JSP koristi otvorene standarde
  - JSP podržava korišćenje alata i automatskih generatora



Slika 7. Prikaz zahteva za JSP komercijalne internet aplikacije

❶korisnik zahteva JSP

❷pretraživač šalje GET ili POST

❸Veb server pušta JSP zahtev do servera aplikacije

❹Veb server aplikacije počinje da procesira JSP

❺prvi put kad je JSP zahtev poslat JSP se kompajlira

❻JSP se kompajlira kao servlet

❼Server veb aplikacije izvršava JSP i pravi HTML

❽HTML se vraća korisniku

Korišćenje JSP-a za povratak

- Sledeći primer sadrži povratak u formi JSP-a.

  primer PostListResults.jsp