Programowanie WWW

Model-View-Controller

Przypomnienie problemu

- Aplikacja do liczenia kredytów
- Klasa Kredyt
- Formatka do wprowadzania danych (czysty HTML)
- Skrypt liczący ratę (JSP wykorzystujące klasę Kredyt)

Klasa Kredyt

```
public class Kredyt {
    double procent;
    double kwota;
    double lat;

    public double getProcent() {
        return procent;
    }

    public void setProcent(double procent) {
        this.procent = procent;
    }
...
```

Klasa Kredyt cd

```
public double getKwota() { return kwota;}

public void setKwota(double kwota) { this.kwota = kwota; }

public double getLat() { return lat; }

public void setLat(double lat) { this.lat = lat; }

public double getRata() {
    double rata = kwota * (procent/12)/
                  (1-(1/Math.pow(1.0+procent/12,lat*12)));
    return rata;
}
//nie ma setRata() !!!
}
```

Formatka

```
<form action="result.jsp">
kwota: <input type="text" name="kwota"><br/>
ile lat: <input type="text" name="lat"><br/>
procent: <input type="text" name="procent"><br/>
<input type="submit"/>
</form>
```

Skryplet z setterami

```
<%
pl.edu.swsim.Kredyt kredyt = new pl.edu.swsim.Kredyt();
kredyt.setKwota(Double.parseDouble(
                        request.getParameter("kwota")));
kredyt.setLat(Double.parseDouble(
                        request.getParameter("lat")));
kredyt.setProcent(Double.parseDouble(
                        request.getParameter("procent")));
out.write("<p>Rata wynosi: "+kredyt.getRata()+"</p>");
%>
```

JSP z tagami

```
<jsp:useBean id="credit" class="pl.edu.swsim.Kredyt">
 <jsp:setProperty name="credit" property="kwota" param="kwota"/>
 <jsp:setProperty name="credit" property="procent" param="procent"/>
 <jsp:setProperty name="credit" property="lat" param="lat"/>
</jsp:useBean>
```

Rata: <jsp:getProperty name="credit" property="rata"/>

JSP z tagami

```
<jsp:useBean id="credit" class="pl.edu.swsim.Kredyt">
 <jsp:setProperty name="credit" property="kwota" param="kwota"/>
 <jsp:setProperty name="credit" property="procent" param="procent"/>
 <jsp:setProperty name="credit" property="lat" param="lat"/>
</jsp:useBean>
```

Rata: ${credit.rata}

Użycie tagów

- Unikamy kodu w Javie na stronie JSP
- Problem: niektóre rzeczy łatwiej napisać w Javie
- Zalety czystej Javy (względem tagów na stronie JSP)
  - Łatwiejszy dostęp do zasobów
  - Bieżąca kontrola poprawności kodu
  - Łatwiejsze debugowanie
  - Możliwość testowania poza serwerem WWW
  - Prosta obsługa błędów
- Plan na dziś:
  - Tworzenie servletów
  - Połączenie servletu i strony JSP

Tworzenie servletu

- Servlet to klasa dziedzicząca po HttpServlet
- Dwie podstawowe metody:
  - doGet(HttpServletRequest req, HttpServletResponse resp)
  - doPost(HttpServletRequest req, HttpServletResponse resp)
- Wypisanie tekstu HTML:
  - PrintWriter out = response.getWriter();
  - out.println("Hello");

Podpięcie servletu w projekcie

- Plik WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>pl.swsim.MyServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
</web-app>
```

Stworzenie servletu

- Tworzymy bezpośrednio kod servletu zamiast strony JSP

```
public class MyController extends HttpServlet {
public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException,
    ServletException {
        response.getWriter().println("Hello from GET");
}
public void doPost(HttpServletRequest request,
    HttpServletResponse response) throws IOException,
    ServletException {
        response.getWriter().println("Hello from POST");
}
```

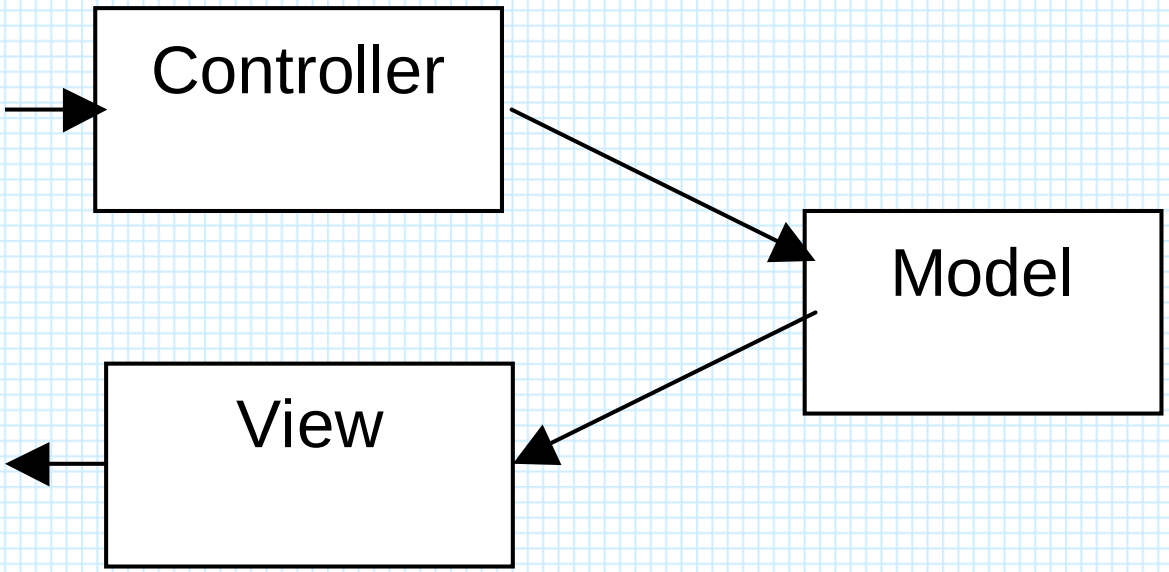
Tworzenie łańcucha servletów

- Servlet może pobrać z kontekstu obiekt RequestDispatcher:
  - getServletContext().getRequestDispatcher("url")
- Dwie metody:
  - forward(request, response)
  - include(request, response)
- Przekierowanie wywołania:
  - getServletContext().getRequestDispatcher("/form.jsp")
    .forward(request, response);
- Przed przekierowaniem można umieścić dane w request:
  - request.setAttribute("klucz",dowolnaDana);

Architektura MVC

- Controller – otrzymuje żądania od klienta i przesyła je do modelu
- Model – logika, dane, stan aplikacji
- View – prezentacja danych stworzonych w modelu

MVC model

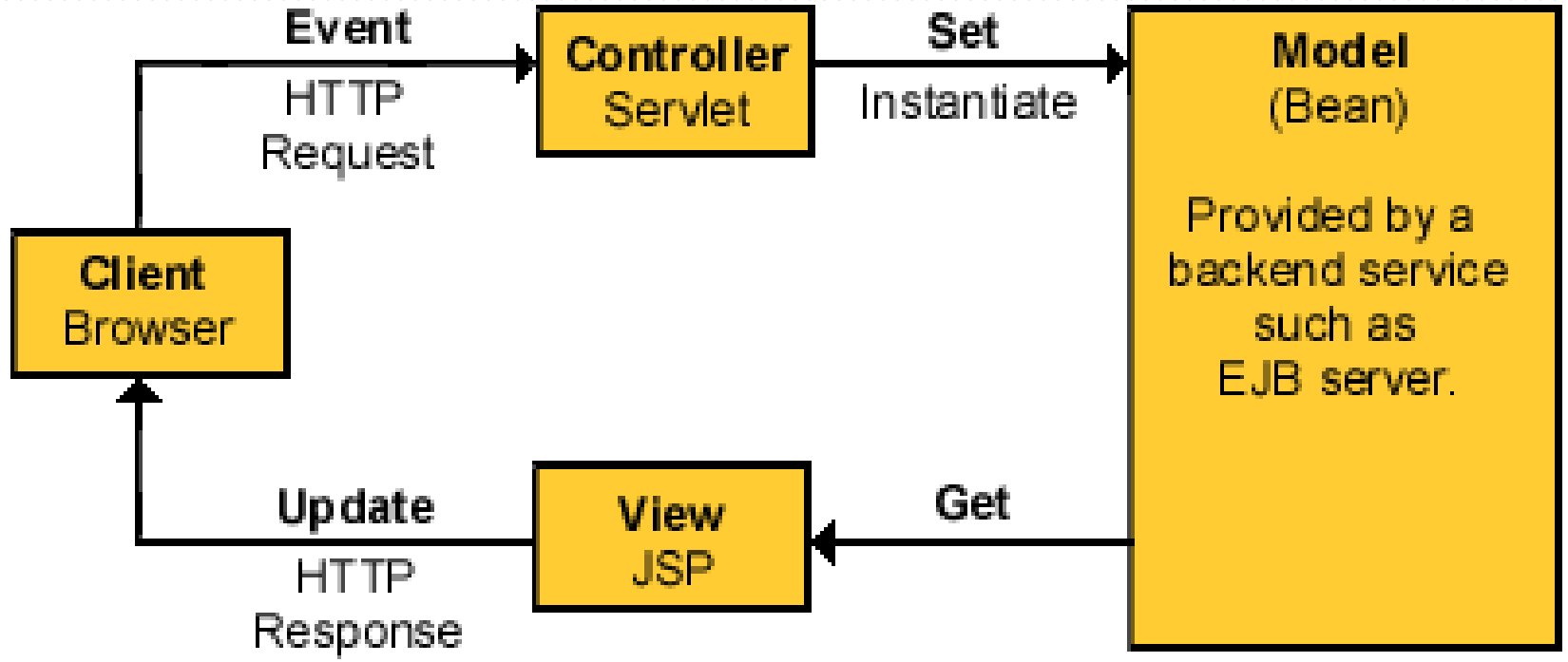


MVC i WWW

- Strony wysyłane "for request"
- Brak stałej komunikacji między Model i View
- View tworzy się głównie w HTML, resztę w innym języku (np. w Javie)


JAVA

MVC Model 2


Event
HTTP
Request
Controller
Servlet
Set
Instantiate
Model
(Bean)
Provided by a
backend service
such as
EJB server.
Client
Browser
Update
HTTP
Response
View
JSP
Get

Podstawy MVC

- Klient wysyła żądanie z parametrami
- Servlet analizuje parametry i uruchamia komponent
- Komponent tworzy beany
- Wypełnione beany są wysyłane do strony JSP
- Strona JSP jest przetwarzana na HTML i wysyłana do klienta

Przykład MVC

- Kontroler - servlet
  - Otrzymuje zgłoszenia od użytkowników
  - Przygotowuje dane (za pomocą Modelu)
  - Umieszcza dane w obiekcie request
  - Przekazuje sterowanie do Widoku
- Model – klasa Java
  - Udostępnia metody do tworzenia i modyfikacji danych
  - Jest niezależny od kontrolera i widoku (czyste przetwarzanie danych)
- Widok – strona JSP
  - Wydobywa dane z request
  - Prezentuje je użytkownikowi w HTML

Przykład z kredytem

- Klasa MyServlet - otrzymuje wszystkie wywołania użytkownika
- W zależności od typu
  - GET – wywołuje formatkę index.jsp
  - POST – oblicza kredyt, umieszcza wynik w request i wywołuje result.jsp
- Klasa Kredyt – bez zmian!
- Index.jsp – bez zmian!
- Result.jsp – nie trzeba już ładować beana

Metoda doGet

- Tylko przekierowuje do formularza

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException,
    ServletException {
    getServletContext()
          .getRequestDispatcher("/WEB-INF/jsp/form.jsp")
                .forward(req, resp);
}
```

Metoda doPost

- Tworzy obiekt klasy Kredyt
- Zasila go argumentami
- Wrzuca go do request
- Przekierowuje do strony result.jsp

Metoda doPost

```
protected void doPost(HttpServletRequest req,
    HttpServletResponse resp) throws ServletException, IOException {
Kredyt kredyt = new Kredyt();
kredyt.setKwota(Double.parseDouble(req.getParameter("kwota")));
kredyt.setLat(Double.parseDouble(req.getParameter("lat")));
kredyt.setProcent(Double.parseDouble(req.getParameter("procent")));

req.setAttribute("kredyt", kredyt);
getServletContext().getRequestDispatcher("/WEB-INF/jsp/result.jsp")
                                    .forward(req, resp);
}
```

Plik result.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>

Kwota: ${kredyt.kwota} <br/>
Rata: ${kredyt.rata}
```

Zalety rozwiązania

- W plikach jsp tylko formatowanie i używanie gotowych danych
- Logika tylko w klasach Javy
- Możliwa lepsza obsługa błędów!
  - Wyłapanie wyjątku NumberFormatException
  - Wrzucenie zmiennej "error" do requesta
  - Przekierowanie z powrotem do strony z formatką

Kod obsługi błędów

- Metoda doPost

```
try{
    kredyt.setKwota(Double.parseDouble(req.getParameter("kwota")));
    kredyt.setLat(Double.parseDouble(req.getParameter("lat")));
    kredyt.setProcent(Double.parseDouble(req.getParameter("procent")));
}catch(NumberFormatException ex) {
    req.setAttribute("error", "Podałeś nieprawidłowe dane - to nie liczby!");
    getServletContext().getRequestDispatcher("/WEB-INF/jsp/form.jsp")
                                        .forward(req, resp);
    return;
}
```

Umieszczenie błędu na formatce

```
<%@ page contentType="text/html; charset=UTF-8" %>

<form method="POST">
kwota: <input type="text" name="kwota"><br/>
ile lat: <input type="text" name="lat"><br/>
procent: <input type="text" name="procent"><br/>
${error}
<input type="submit"/>
</form>
```

Przykład z bazą danych

- Tabela samochodów (car)
- Możliwość obejrzenia samochodów i dodania nowego
- Docelowo możliwość wykonania wszelkich operacji na wierszach tabeli
  - Tworzenie (CREATE)
  - Oglądanie zawartości (RETRIEVE)
  - Aktualizacja (UPDATE)
  - Usuwanie (DELETE)
- W skrócie: CRUD

Data Access Objects (DAO)

- Główna idea: uniezależnić aplikację od źródła danych
- Interfejs DAO zapewnia wszystkie operacje na danych (tzw. CRUD – Create Retrieve Update Delete)
- Składniki:
  - interfejs DAO
  - źródło danych (DataSource)
  - obiekt transferowy (JavaBean)

Klasa transferowa

```
public class Car {
    private String make;
    private String model;
    private double price;
    ...gettery i settery...
}
```

Stworzenie tabeli

```
CREATE TABLE car (
  idc int NOT NULL auto_increment,
  make varchar(50),
  model varchar(50),
  price double default NULL,
  PRIMARY KEY  (`idc`)
);
```

Wstawienie danych

```
INSERT INTO car VALUES(1,'Fiat','Brava', 20000);
INSERT INTO car VALUES(2,'Fiat','Punto', 22000);
INSERT INTO car VALUES(3,'Ford','Fiesta',15600);
INSERT INTO car VALUES(4,'Ford','Focus',11000);
INSERT INTO car VALUES(5,'Opel','Astra',44500);
```

CarDAO

- Metody:
  - List<Car> getCars()
  - void addCar(Car car)
- Połączenie z bazą danych (DBCP)

```
public Connection getConnection() {
  try{
    Context initContext = new InitialContext();
    DataSource ds
          = (DataSource)initContext.lookup("java:/comp/env/jdbc/mydb");
    return ds.getConnection();
  }catch(Exception ec) { ec.printStackTrace(); }
  return null;
  }
```

Konfiguracja połączenia (xml)

```
<Context path="/web1" reloadable="true">
    <Resource name="jdbc/mydb" auth="Container"
    type="javax.sql.DataSource"
    maxActive="100" maxIdle="30" maxWait="10000"
    username="myuser" password="mypass"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/mydb?
                                        autoReconnect=true"/>
</Context>
```

getCars()

```
public List<Car> getCars() {
    List<Car> lista = new ArrayList<Car>();
    Connection con = getConnection();
    try{
    ResultSet rs = con.createStatement().executeQuery("select * from car");
    while(rs.next()) {
            Car car = new Car();
            car.setIdc(rs.getInt("idc"));
            car.setMake(rs.getString("make"));
            car.setModel(rs.getString("model"));
            car.setPrice(rs.getDouble("price"));
            lista.add(car);
    }
    con.close();
}catch(SQLException ec) {ec.printStackTrace();}
return lista;
}
```

insertCar()

```
public void insertCar(Car car) {
try{
    Connection con = getConnection();
    PreparedStatement pstmt = con.prepareStatement(
    "insert into car(make,model,price) values(?,?,?)");
    pstmt.setString(1, car.getMake());
    pstmt.setString(2, car.getModel());
    pstmt.setDouble(3, car.getPrice());
    pstmt.executeUpdate();
    con.close();
} catch(SQLException ec) {ec.printStackTrace();}
}
```

Controller

```
public class MyController extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException {
      processRequest(request, response);
   }
public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException {
      processRequest(request, response);
   }
protected void processRequest(
     HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException {
...
}
```

processRequest

```
String actionName = request.getParameter("action");
if(actionName.equals("carList")) {
    List carList = carDAO.getCars();
    request.setAttribute("carList", carList);
    destinationPage = "/carList.jsp";
} else {
    String error = "[" + actionName + "] is not a valid action.";
    request.setAttribute("error", error);
    destinationPage = "/error.jsp";
}
```

processRequest cd

```
// przekierowanie do wybranej strony jsp
RequestDispatcher dispatcher =
        getServletContext().
                getRequestDispatcher(destinationPage);
dispatcher.forward(request, response);
```

Strona carList.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<table>
   <tr>
     <th>Make</th><th>Model</th><th>Year</th>
   </tr>
   <c:forEach items='${carList}' var='car'>
     <tr>
     <td>${car.make}</td>
     <td>${car.model}</td>
     <td>${car.price}</td>
     </tr>
   </c:forEach>
  </table>
```

Strona error.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

${error}
```

Control flow

- Wszystkie zgłoszenia idą do serwletu MyController z parametrem action=...
- MyController przygotowuje dane, wrzuca do obiektu request i przesyła je do strony jsp
- Strona jsp jest wypełniana danymi przesyłana do użytkownika jako html

Konfiguracja zgłoszeń *.go

```
<web-app>
<!-- servlet definition -->
   <servlet>
      <servlet-name>Go</servlet-name>
      <servlet-class>car.servlet.MyController</servlet-class>
   </servlet>

   <!-- servlet mapping -->
   <servlet-mapping>
      <servlet-name>Go</servlet-name>
      <url-pattern>*.go</url-pattern>
   </servlet-mapping>
</web-app>
```

Czytanie requestURI

- requestURI zawiera pełną ścieżkę ze zgłoszenia
- String path = request.getRequestURI();
- Przykładowo:
  - /webcar/car/carList.go
- Rozpakowanie:

```
String path = request.getRequestURI();
actionName = path.substring(path.lastIndexOf("/")+1,
    path.lastIndexOf("."));
```

Kolejny krok: dodawanie

- Formatka dla nowego samochodu
- Akcja pokazująca formatkę (carInsert)
- Akcja zapisująca pola z formatki (carInsertSave)

Akcja dodaj samochód

```
if(actionName.equals("carInsert")) {
    destinationPage = "/carInsert.jsp";
    }
```

Formatka do edycji

```
<form action="carInsertSave.go">
make: <input type="text" name="make"><br/>
model: <input type="text" name="model"><br/>
price: <input type="text" name="price"><br/>
<input type="submit"/>
</form>
```

Obsługa zgłoszenia

```
if(actionName.equals("carInsertSave")) {
    Car car = new Car();
    car.setMake(request.getParameter("make"));
    car.setModel(request.getParameter("model"));
    car.setPrice(Double.parseDouble(request.getParameter("price"));
    carDAO.insertCar(car);

    List carList = carDAO.getCars();
    request.setAttribute("carList", carList);
    destinationPage = "/carList.jsp";
    }
```

Obsługa zgłoszenia

```
if(actionName.equals("carInsertSave")) {
    Car car = new Car();
    car.setMake(request.getParameter("make"));
    car.setModel(request.getParameter("model"));
    car.setPrice(Double.parseDouble(request.getParameter("price"));
    carDAO.insertCar(car);

    List carList = carDAO.getCars();
    request.setAttribute("carList", carList);
    destinationPage = "/carList.jsp";
    }
```

Obsługa zgłoszenia

```
if(actionName.equals("carInsertSave")) {
        Car car = new Car();
        car.setMake(request.getParameter("make"));
        car.setModel(request.getParameter("model"));
        car.setPrice(Double.parseDouble(request.getParameter("price"));
        carDAO.insertCar(car);

        destinationPage = "/carList.go";
}
```

Obsługa zgłoszenia

```
if(actionName.equals("carInsertSave")) {
        Car car = new Car();
        car.setMake(request.getParameter("make"));
        car.setModel(request.getParameter("model"));
        car.setPrice(Double.parseDouble(request.getParameter("price")));
        carDAO.insertCar(car);

        destinationPage = "/carList.go";
}
```

Obsługa zgłoszenia

```
if(actionName.equals("carInsertSave")) {
        Car car = new Car();
        JspRuntimeLibrary.introspect(car, request);
        carDAO.insertCar(car);

        destinationPage = "/carList.go";
}
```

Kolejne kroki

- Edycja samochodu
- Usunięcie samochodu
- Dodanie przycisków edycja i usuń w tabeli
- Akcje:
  - carList
  - carInsert
  - carEdit?id=X
    - jeśli id=0 dodaje nowy samochód
  - carDelete?id=X

Usunięcie samochodu

```
if(actionName.equals("carDelete")) {
    int idc = Integer.parseInt(request.getParameter("idc"));
    carDAO.deleteCar(idc);
    destinationPage = "/carList.go";
    }
```

Akcja edycji

```
if(actionName.equals("carEdit")) {
    String idcStr = request.getParameter("idc");
    int idc = Integer.valueOf(idcStr);
    Car car = carDAO.getCar(idc);
    request.setAttribute("car",car);
    destinationPage = "/carEdit.jsp";
    }
```

Formatka edycji

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<form action="carEditSave.go">
<input type="hidden" name="idc" value="${car.idc}">
make: <input type="text" name="make" value="${car.make}"><br/>
model: <input type="text" name="model" value="${car.model}"><br/>
price: <input type="text" name="price" value="${car.price}"><br/>
<input type="submit"/>
</form>
```

Zapis edycji

```
if(actionName.equals("carEditSave")) {
    Car car = new Car();
    car.setIdc(Integer.parseInt(request.getParameter("idc")));
    car.setMake(request.getParameter("make"));
    car.setModel(request.getParameter("model"));
    car.setPrice(Double.parseDouble(request.getParameter("price")));
    carDAO.updateCar(car);
    destinationPage = "/carList.go";
    }
```