DBS – 4. ročník
PL/SQL

Programování DB aplikací v jazyce JAVA

Konstrukce MVC aplikace

Základní pojmy:

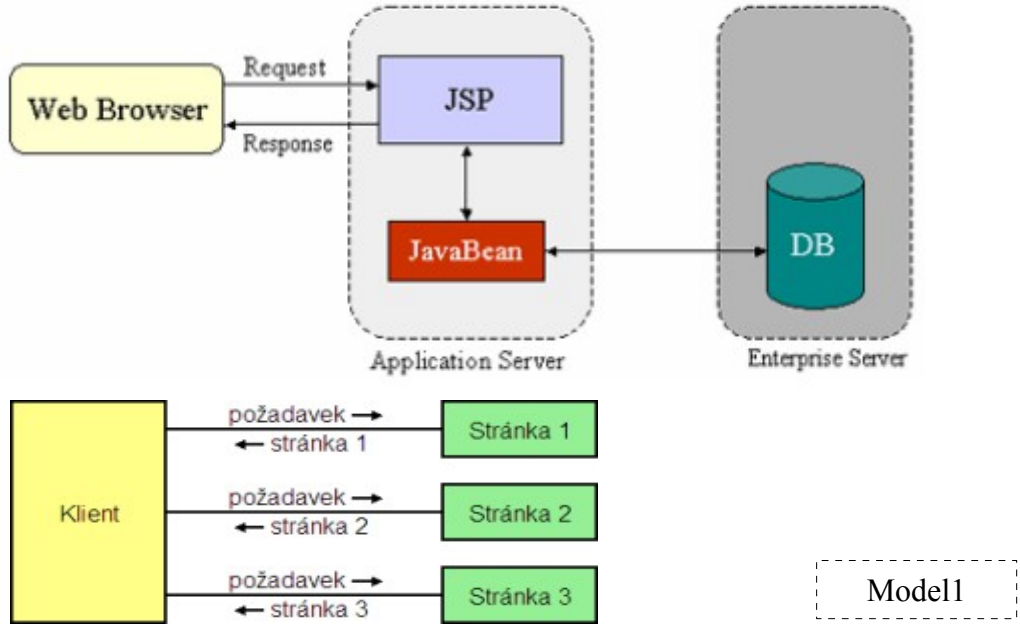
1. **Internet, http, www – viz. ICT**
2. **GUI (Graphical User Interface) – viz. sešit**
   je druh komunikace s počítačem mající podobu interaktivních grafických prvků. V dnešní době se nejčastěji skládá z oken zabírajících větší či menší část počítačového monitoru.
3. **J2EE (Java to Enterprise Edition)**
   - "enterprise" část Java technologie; aktuálně v1.5
   - jednou z částí servlety a JSP
4. **Kontejner**
   - prostředí pro běh servletů
   - Tomcat (Apache Jacarta projekt), aktuálně v5.5
   - OC4J – Oracle Contejner for Java
5. **Aplikační server**
   - řeší aplikační logiku
   - řeší podporu GUI u webové aplikace
6. **Servlet**
   - Java třída (objekt) která umí obsloužit HTTP požadavek
   - aktuální verze specifikace 2.4/2.5
7. **JavaServer Page (JSP)**
   - Java jako zapouzdřený HTML skriptovací jazyk
   - HTML s doplněnými JSP značkami (a kusy Java kódu) navenek stejná myšlenka jako PHP, ASP, …
   - kontejner interně přeloží JSP stránku na servlet
8. **Aplikační frameworky**
   - těsná integrace serveru, scriptovacího jazyka, knihoven a vývojového prostředí
   - Jakarta Struts (Apache)
   - Oracle Application Framework (ADF)
9. **Skriptovací jazyky**
   - PHP (PHP: Hypertext Preprocessor)
   - JSP (JavaServer Pages)
   - ASP/ASP.NET (Active Server Pages)

Architektura aplikace:

... je způsob rozdělení aplikace, aplikačních dat, procesů i datových toků do logických celků, stanovení struktury těchto komponent, vzájemných vztahů a interakcí mezi nimi.
V literatuře zabývající se platformou J2EE se v souvislosti s architekturou aplikace často objevují termíny **„Model 1" - základní** a **„Model 2" - třívrstvá architektura**.

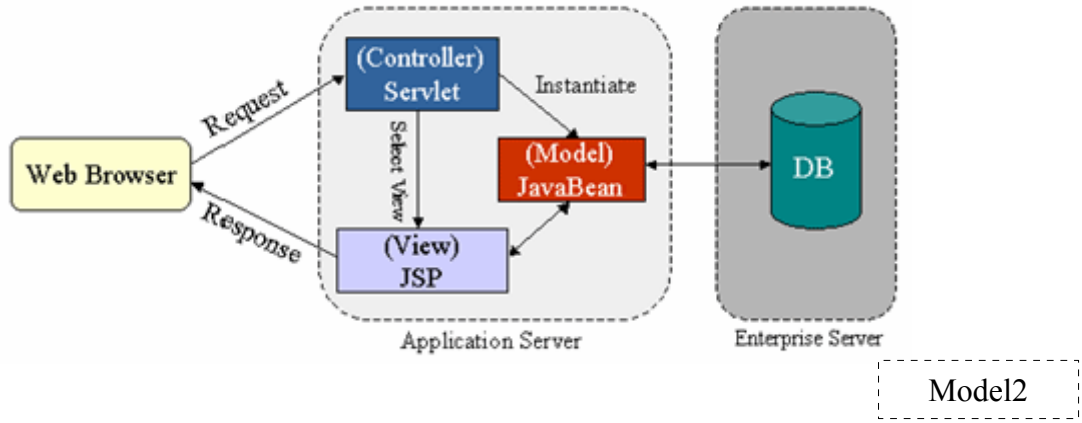
Model 1

je taková architektura, kdy prohlížeč přistupuje k jednotlivým stránkám přímo, otevírá je z URL, na kterém se skutečně nacházejí. Následující stránka k otevření je pak zpravidla určena pevnými odkazy umístěnými přímo v dokumentu. Řízení aplikace založené na Modelu 1 je decentralizované, protože aktuální stránka již definitivně určuje následující otevíranou stránku. Architektura Model 1 odděluje pouze datový model od uživatelského rozhraní s řídící logikou. Model 1 je vhodný pro statické webové prezentace, nanejvýš pro velmi jednoduché aplikace.



Model 2

Model-view-controller (MVC) (někdy také označovaná jako **Model-2**) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní.



Princip MVC

Obecně řečeno, vytváření aplikací s využitím architektury MVC vyžaduje vytvoření tří komponent, mezi které patří:

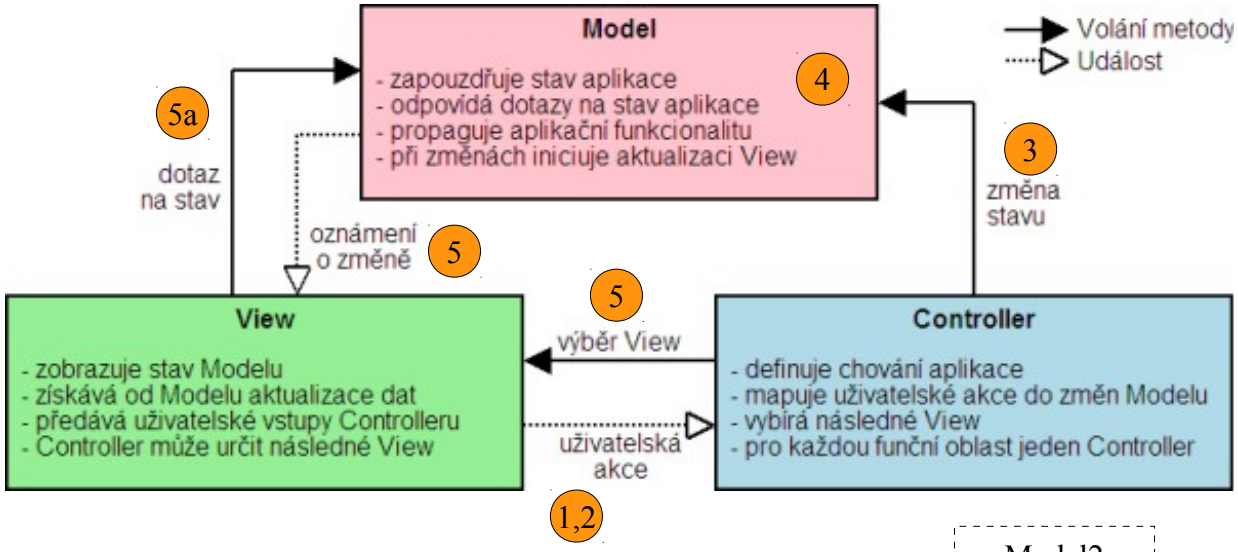
- **Model (model)**, což je doménově specifická reprezentace informací, s nimiž aplikace pracuje. *Model* je funkčním a datovým základem celé aplikace. Poskytuje prostředky jak pro přístup k datové základně a stavům aplikace, tak pro jejich ukládání a aktualizaci. Hovoří se o něm jako o softwarovém obrazu procesů reálného světa.
- **View (pohled)**, který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli. View zobrazuje obsah Modelu, zajišťuje grafický či jiný výstup aplikace. Přes Model přistupuje k datům a stavům aplikace a specifikuje, jak mají být prezentovány. Při změně stavu Modelu aktualizuje

zobrazení.

- **Controller (řadič)**, který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu. Controller definuje chování aplikace. Zpracovává veškeré vstupy a události pocházející od uživatele. Na jejich základě volá příslušné procesy Modelu, mění jeho stav apod. Podle událostí přijatých od uživatele i podle výsledků akcí v Modelu pak vybírá Controller vhodné View pro další zobrazení.

Ačkoliv může být koncept MVC realizován různým způsobem, obecně platí tento princip:
(viz obrázek)

1. Uživatel provede nějakou akci v uživatelském rozhraní (např. stiskne tlačítko).
2. Řadič obdrží oznámení o této akci z objektu uživatelského rozhraní.
3. Řadič přistoupí k modelu a v případě potřeby ho zaktualizuje na základě provedené uživatelské akce (např. zaktualizuje nákupní košík uživatele).
4. Model je pouze jiný název pro doménovou vrstvu. Doménová logika zpracuje změněná data (např. přepočítá celkovou cenu, daně a expediční poplatky pro položky v košíku). Některé aplikace užívají mechanizmus pro perzistentní uložení dat (např. databázi). To je však otázka vztahu mezi doménovou a datovou vrstvou, která není architekturou MVC pokryta.
5. Komponenta pohled použije zaktualizovaný model pro zobrazení zaktualizovaných dat uživateli (např. vypíše obsah košíku). Komponenta pohled získává data přímo z modelu, zatímco model nepotřebuje žádné informace o komponentě View (je na ní nezávislý). Nicméně je možné použít návrhový vzor pozorovatel, umožňující modelu informovat jakoukoliv komponentu o případných změnách dat (obrázek 5a). V tom případě se komponenta view zaregistruje u modelu jako příjemce těchto informací. Je důležité podotknout, že řadič nepředává doménové objekty (model) komponentě pohledu, nicméně jí může poslat příkaz, aby svůj obsah podle modelu zaktualizovala.
6. Uživatelské rozhraní čeká na další akci uživatele, která celý cyklus zahájí znovu.



MVC v praxi

Tento vzor poprvé popsal Trygve Reenskaug v roce 1979. Poprvé byl použit v jazyce Smalltalk, vyvíjeném v Xerox research labs. Touto implementací bylo inspirováno mnoho dalších projektů, např.:NeXTSTEP .

V současné době se koncept MVC užívá především jako architektura webových aplikací.

MVC je velice rozšířený a oblíbený přístup k návrhu aplikace. Používá se zejména v oblasti jazyka Java. Zde je Model obvykle realizován třídami definovanými v JavaBeans komponentách. View jsou generovány pomocí JavaServer Pages (JSP). Controller pak bývá implementován jako skupina servletů, místo mnoha různých servletů může být jeden hlavní servlet jako Front Controller.

Nejrozšířenějšími frameworky poskytujícími vlastní Controller jsou Apache Struts a J2EE Application Framework, ADF (Oracle)

Příklad MVC: Aplikace shop

Naše aplikace se bude skládat pouze z katalogu produktů a objektu nákupního koše (Cart), který bude mít každý návštěvník uložen v objektu session. Do tohoto koše budete moci přidávat položky a odebírat je (pouze všechny naráz, pokud byste je chtěli odebírat po jedné, budete si muset tuto vlastnost doprogramovat sami). Aplikace nepoužívá databázi, ale má zboží nadefinováno ve třídě `ActionServlet`, která slouží jako controler. S tím souvisí i poslední slabina aplikace, a to že předává data do nákupního košíku přes html formulář a ne pouze id požadovaného zboží.

Model

`CatalogItem` reprezentuje 1 řádek v katalogu. Další modely jsou třídy Cart a CartItem.

```
package com.patrny.shop;

public class CatalogItem {
    private int id;
    private String name;
    private String description;
    private int price;

    public CatalogItem() {
    }

    public CatalogItem(int id, String name, String description,
        int price) {

        this.id = id;
        this.name = name;
        this.description = description;
        this.price = price;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public int getPrice() {
        return price;
```

```
    }

    public void setPrice(int price) {
        this.price = price;
    }
}
```

Controler

Controlerem je servlet `ActionServlet`:

```
package com.patrny.shop;

import java.util.Vector;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;

public class ActionServlet extends HttpServlet {
    private static Vector catalogItems = new Vector();

    public void init(ServletConfig config) throws
ServletException {
         super.init(config);

        // inicializace věcí catalogu -
        // správně by se měly nahrávat z databáze
        catalogItems.add(
           new CatalogItem(1,"lžíce","nerezová lžíce",15));
        catalogItems.add(
           new CatalogItem(2,"stará lžíce","hliníková lžíce",5));
        catalogItems.add(
           new CatalogItem(3,"nůž","nerezový nůž",20));
        catalogItems.add(
           new CatalogItem(4,"vidlička","nerezová vidlička",15));
    }

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html; charset=iso-8859-2");
        HttpSession session = request.getSession();

        // nákupní koš
        Cart cart = (Cart)session.getAttribute("cart");
        if(cart==null) {
            cart = new Cart();
            session.setAttribute("cart", cart);
        }

        // zpracování requestů
        String action = request.getParameter("action");
        if(action==null) {
            request.setAttribute("catalogItems",
                catalogItems.elements());

            prepareCatalog(request, response);
```

```
        }
        else if(action.equals("addToCart")) {
            try {
                String name = enc(request.getParameter("name"));
                int price = Integer.parseInt(
                    request.getParameter("price"));
                int quantity = Integer.parseInt(
                    request.getParameter("quantity"));
                int id = Integer.parseInt(
                    request.getParameter("id"));

                cart.add(new CartItem(id, name, price, quantity));
            }
            catch(NumberFormatException e) {}
            prepareCatalog(request, response);
        }
        else if(action.equals("removeAllFromCart")) {
            cart.removeAll();
            prepareCatalog(request, response);
        }
        else if(action.equals("showCart")) {
            prepareShowCart(request, response);
        }
    }

    private void prepareCatalog(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        request.setAttribute("catalogItems",
         catalogItems.elements());

        getServletContext().getRequestDispatcher(
            response.encodeRedirectURL("/catalog.jsp")).
            forward(request, response);
    }

    private void prepareShowCart(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        getServletContext().getRequestDispatcher(
            response.encodeRedirectURL("/showcart.jsp")).
            forward(request, response);
    }

    private String enc(String param) {
        try {
            return new String(param.getBytes("iso-8859-1"),
                "iso-8859-2");
        }
        catch(java.io.UnsupportedEncodingException e) {}
        return param;
    }

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
```

```
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        processRequest(request, response);
    }
}
```

View

Všimněte si že ve view nedochází k ničemu jinému než k zobrazení dat (použitím custom tags):

```
<%@page contentType="text/html; charset=iso-8859-2"%>
<%@taglib uri="com.patrny.shop" prefix="shop" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
    <title>Katalog - vyberte si sami</title>
</head>
<body>
<h2>Železářství z druhého patra</h2>
<a href="Action?action=showCart">Obsah koše</a>
<table border="1">
<tr>
<td>Název</td><td>Popis</td><td>Cena</td>
<td>Po?et</td><td> </td>
</tr>
<shop:Catalog>
<form action="Action" method="get">

<tr>
<td>
<shop:CatalogItem column="name"/>
<input type="hidden" name="name"
 value="<shop:CatalogItem column="name"/>">
</td>
<td>
<shop:CatalogItem column="description"/>
</td>
<td>
<shop:CatalogItem column="price"/>
<input type="hidden" name="price"
 value="<shop:CatalogItem column="price"/>">
</td>
<td><input type="text" name="quantity"></td>
<td>
<input type="hidden" name="action" value="addToCart">
<input type="hidden" name="id"
value="<shop:CatalogItem column="id"/>">
<input type="submit" value="Přidat do koše">
</td>
</tr>

</form>
</shop:Catalog>
```

```
</table>

</body>
</html>
```