



# Architektúra internetových a intranetových systémov:

## Prezentačná vrstva

### Úvod + Serverovské stránky

Ľubor Šešera



# Prezentačná vrstva a WWW

- Použ. rozhranie = rozhranie pre používateľa na prácu so systémom
- Základné princípy moderného rozhrania:
  - projekt WorldWideWeb (CERN, Tim Berners-Lee)
- Spojenie 2 technológií: hypertext a internet
- Myšlienky
  - sieť (angl. web) hypertextových dokumentov uložených na serveroch
  - prezeranie pomocou univerzálneho prehliadača (angl. browser)
- Výsledky (1990, publikované 6.8.1991)
  - Spôsob identifikácie dokum./zdroja na internete: URI
  - Jazyk na prezentáciu dokumentov: HTML
  - Protokol na sieťovú komunikáciu: HTTP

# Ako dodať dynamiku?

- Pôvodný HTML: statický jazyk na prezzeranie dokumentov
  - Nevieme zobrazit' ani len zostatok účtu, lebo ten sa v čase mení
- Pokusy o dynamiku na stranu klienta
  - JavaApplets (Sun, 1995)
  - JavaScript (Netscape, 1995)
- Problémy:
  - Rozsah kódu prenášaný cez pomalé siete
  - Bezpečnosť
- Pozornosť sa sústredila na pridanie dynamiky na serveri
- Špecifikácia CGI (NCSA, 1993)
  - Spôsob, ako webový server má volať program v niektorom programovacom jazyku (napr. Perl)
  - Mále efektívne, zle škálovateľné, nízkoúrovňové odovzdávanie parametrov

# Serverovské stránky: JavaServlets

- Špecifikácia JavaServlets (Sun, 1997)
- Výhody:
  - Efektívnosť
    - Požiadavka sa spracuje vo vlákne (thread)
  - Škálovateľnosť
    - Servlet dokáže paralelne obslúžiť viac vlákien
  - Štruktúrovaný prístup k dátam
    - Web kontajner automaticky transformuje reťazec znakov na štruktúrovaný Java objekt
  - Jazyk Java
    - Ktorý sa stal silným a rozšíreným jazykom
    - Komponent (servlet) môže pristupovať k službám webového servera
  - Portabilitosť
    - Špecifikácia implementovaná väčšinou výrobcov web serverov

# Serverovské stránky: JavaServlets

- Nevýhody JavaServlets

- Výstupný HTML sa vytváral ako reťazec znakov do objektu typu stream

# Príklad servletu

```
public class MojServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                        HttpServletResponse response) {  
        String cisloUctu = request.getParameter ("idUctu");  
        ...  
        PrintWriter out = response.getWriter();  
        out.println ("<html>");  
        out.println ("<head>");  
        ...  
        out.println ("Číslo účtu: " + cisloUctu);  
    }  
}
```

Abstraktná trieda pre servlet

Objekt z požiadavky

Metóda pre spracovanie HTTP GET

Výstupné HTML sa vytvára ako reťazec znakov

# Serverovské stránky: JSP

- **JavaServer Pages (Sun, 1999)**
  - serverovská stránka = šablóna HTML stránky = HTML stránka + bloky kódu v jazyku Java
  - JSP stránka sa pri prvom prístupe kompiluje do JavaServlets
- **Active Server Pages (Microsoft, 1996)**
  - Vo webovom serveri IIS
  - Skriptovací jazyk: VBScript alebo Jscript
- **PHP (1995)**
  - Rozšírenie od PHP 3 (1998)

# Príklad jednoduchéj JSP

```
<table>
  <tr>
    <td> Číslo účtu: </td>
    <td>
      <%
        while ( ... ) { ... }
        ...
      %>
    </td>
  </tr>
  ...
</table>
```

Štandardné HTML

Vnorený kód v jazyku Java (scriptlet)



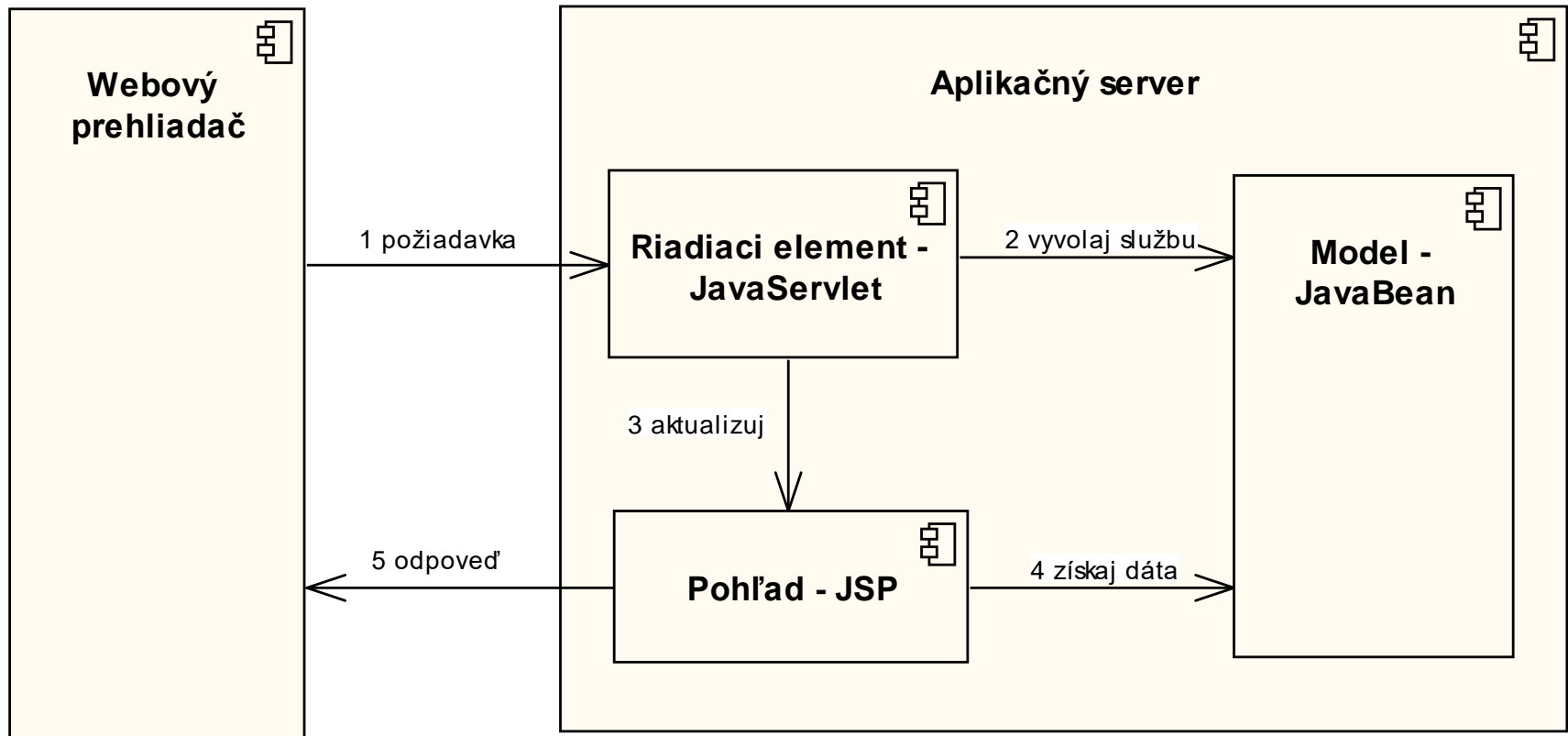
# Model –View - Controller

# Ako štruktúrovať kód?

- Java servlets
  - „Príliš procedurálne“ – výstupná HTML je utopená v kóde (*HTML inside Java*)
- JSP
  - „Príliš HTML“ – HTML stránka je prešpagetovaná kódom (*Java inside HTML*)
- Riešenie:
  - Vzor Model-View-Controller (MVC)
  - Využitie tak JSP (view) ako aj Java servlets (controller)

# MVC v JEE so servletom/JSP

cmp 3.24 MVC v JEE Model 2



# Príklad jednoduchkej JSP

```
<html>
<head>
  <title> Detail účtu </title>
</head>
<body>
....
<jsp:useBean id="ucet" scope="session" class="Ucet" />
<jsp:setProperty name="ucet" property="idUctu"
                                     param="idUctu" />
...

```

HTML značky

Prepojenie na Model

JSP značky

# Príklad jednoduchkej JSP

```
<div style="display:table">
  <div style="display:table-row">
    <div style="display:table-cell"> Číslo účtu:</div>
    <div style="display:table-cell">
      <jsp:getProperty name="ucet" property="idUctu"/>
    </div>
  </div>
  <div style="display:table-row">
    <div style="display:table-cell"> Stav:</div>
    <div style="display:table-cell">
      <jsp:getProperty name="ucet" property="stav"/>
    </div>
  </div>
</div>
</body> </html>
```

Dynamické získanie hodnoty atribútu z Modelu

# Príklad dvojúrovňovej JSP: log. obraz.

```
<jsp:useBean id="ucet" scope="session" class="Ucet" />
```

```
<my:screen>  
<my:title name="Detail účtu" />  
<my:form name="Položky účtu" />  
  <my:field label="Číslo účtu" value="{ucet.idUctu}" />  
  <my:field label="Stav" value="{ucet.stav}" />  
</my:form>  
</my:screen>
```

Zákazkové (používateľom definované)  
značky

Výraz v jazyku EL (JEE  
štandard)  
(dynamické vyhodnotenie)

Zákazkové značky si vývojár môže  
definovať pomocou JSTL JEE štandardu  
– detailnejšie pozri Učebnicu ASS

# Controller

# Serverovský Riadiaci element

Realizuje dve základné funkcie:

1. Spracovanie požiadavky, ktoré sa skladá z dvoch krokov:
  - Spracovanie (handle) udalosti a získanie dát z požiadavky.
  - Vyvolanie akcie Modelu s poslaním potrebných dát z požiadavky.
2. Rozhodnutie o tom, ktorý Pohľad sa má zobrazit' a poskytnutie mu základných informácií z Modelu.



# Typy serv. Riad. elementov (opak.)

- Page Controller
  - 1 Controller pre 1 stránku, presnejšie pre jeden typ požiadavky
- Front Controller
  - (1a) 1 všeobecný vstupný Controller (bezpečnosť,...)
  - (1b) Druhý Controller je špecifický podľa typu požiadavky
- Application Controller
  - Všeobecný controller pre (1b) a (2)
  - Zvyčajne implementuje stavový diagram

Detailnejšie v učebnici

# Stránkový riadiaci element

```
public class UcetDetailController extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                        HttpServletResponse response) {  
        try { Ucet ucet = getUcetService().  
            findUcet(request.getParameter("idUctu"));  
            request.setAttribute("ucet", ucet);  
            forward("ucetDetail.jsp", request, response);  
        } catch (AppException e) {  
            request.setAttribute("exception", e);  
            forward("error.jsp", request, response);  
        }  
    }  
}
```

1a. Spracovanie udalosti (HTTP GET) →

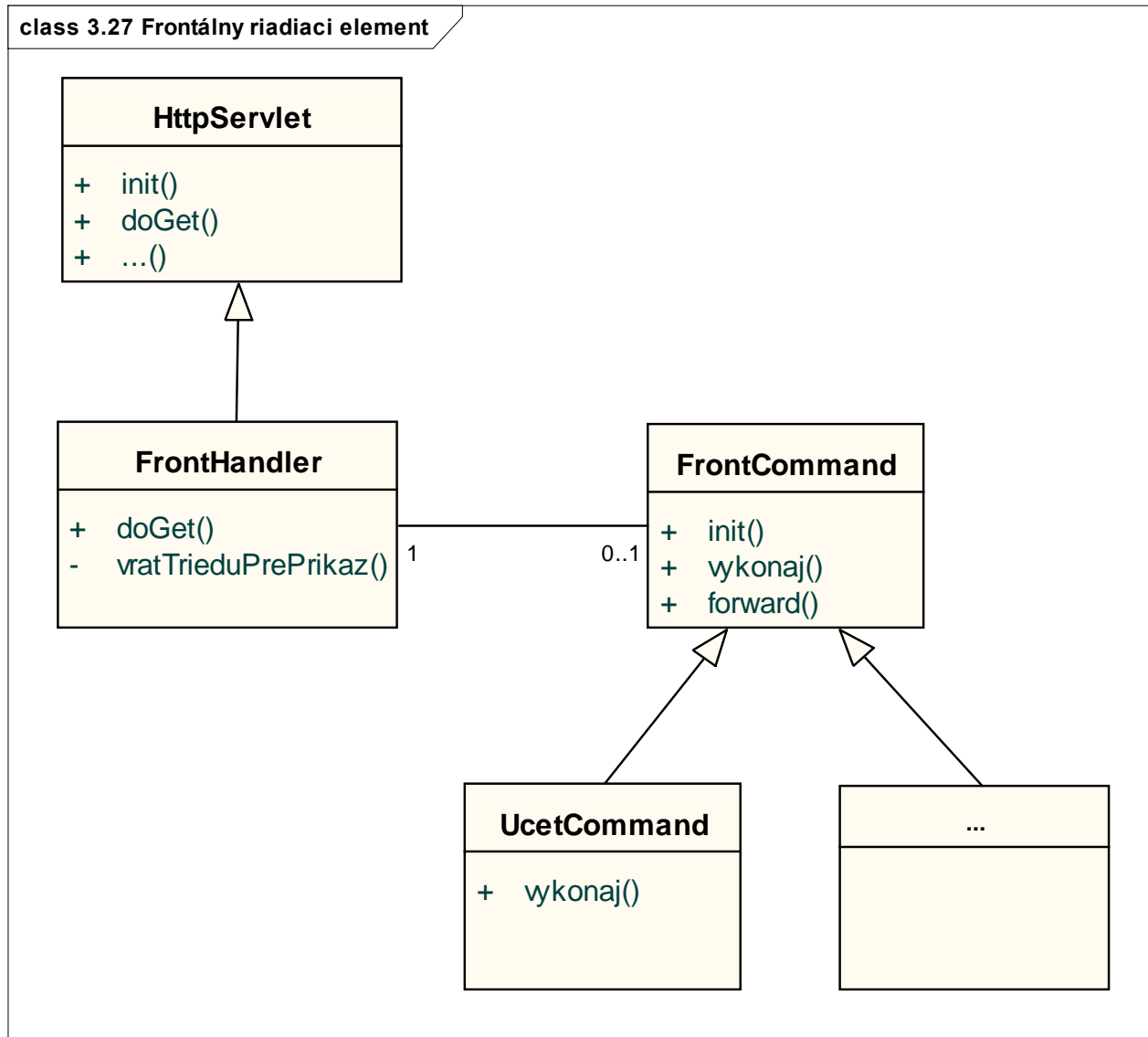
1a. Získanie dát z požiadavky →

1b. Vyvolanie akcie z Modelu →

2. Vyvolanie nasl. stránky →

Všetky kroky Riadiaceho elementu sú „nadrôtované“

# Frontálny riadiaci element



# Frontálny riadiaci element

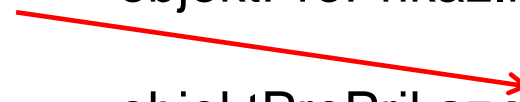
```
class FrontHandler extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                        HttpServletResponse response) throws ...
```

```
{  
    Class triedaPrePrikaz;  
    FrontCommand objektPrePrikaz;  
    try {  
        triedaPrePrikaz = vratTrieduPrePrikaz (request);  
        objektPrePrikaz = (FrontCommand)  
                           triedaPrePrikaz.newInstance ();  
        objektPrePrikaz.init (getServletContext(),  
                              request, response);  
        objektPrePrikaz.vykonaj();  
    } catch (Exception e) {throw new ApplicationException (e);  
    }  
}
```

Podľa parametra z  
požiadavky skonštruuje  
meno triedy




Vyvolá akciu  
Modelu



Žiadne „drôty“, ale všeobecný mechanizmus !

# Frontálny riadiaci element

```
private Class vratTrieduPrePrikaz (HttpServletRequest request) {  
    Class trieda;  
    Final String menoTriedy = (String) request.getParameter("command") +  
                                "Command";  
    try {  
        trieda = Class.forName (menoTriedy);  
    } catch (ClassNotFoundException e) {  
        trieda = UnknownCommand.class;  
    }  
    return trieda;  
}  
}
```



Podľa stringu  
naloží triedu

# Frontálny riadiaci element

```
class UcetCommand extends FrontCommand {  
    public void vykonaj () throws ServletException, IOException {  
        try { Ucet ucet = getUcetService().  
                                findUcet(request.getParameter ("idUctu"));  
            request.setAttribute("ucet", ucet);  
            super.forward("ucetDetail.jsp");  
        } catch (AppException e) {  
            request.setAttribute("exception", e);  
            forward("error.jsp", request, response);  
        }  
    }  
}
```

# Uvol'nenie

# Kto je to?



**Alan Key**

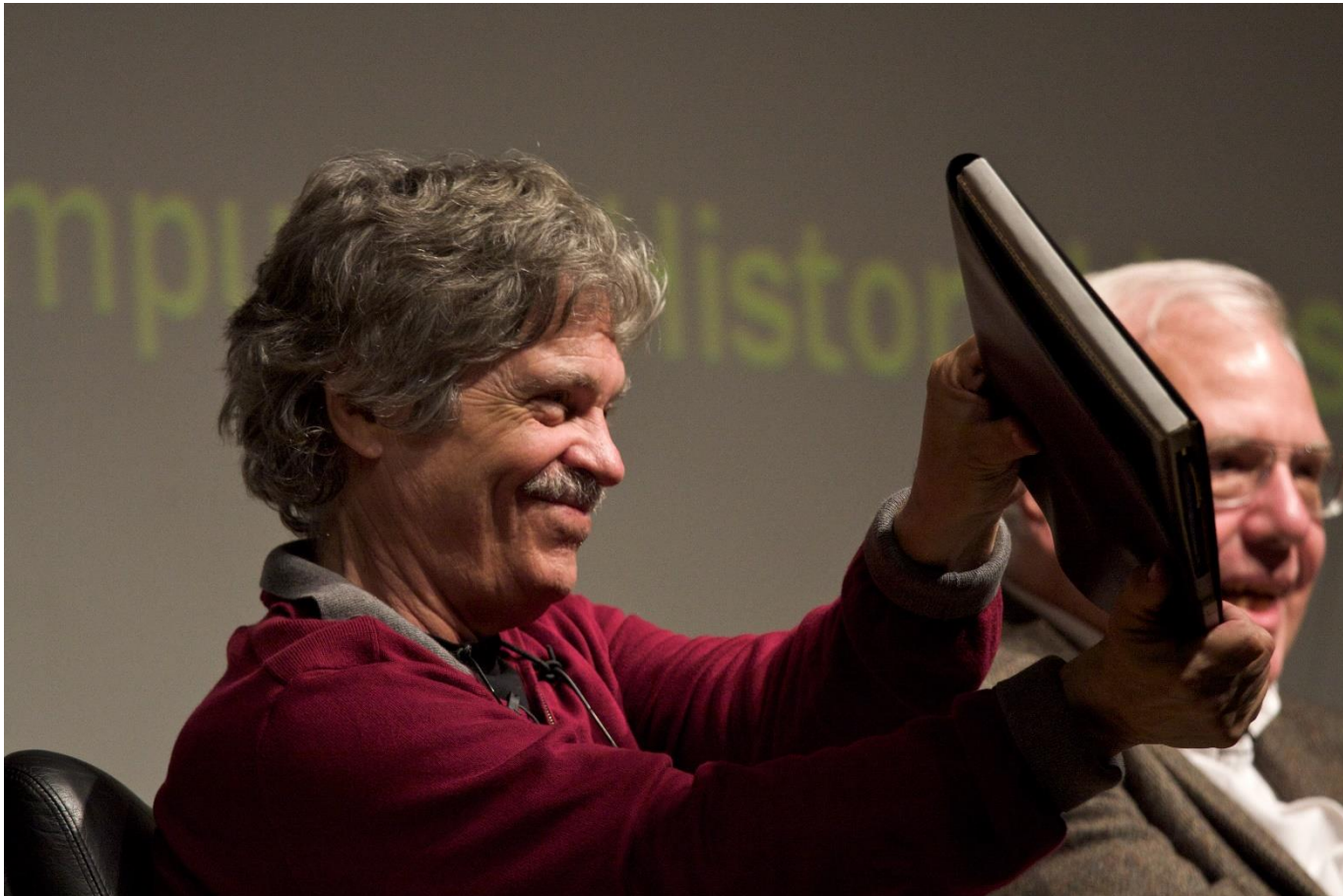
**\* 1940**

- Nositeľ Turingovej ceny (2003)
- PhD študent na University of Utah
  - V roku 1968 ho zaujali práce Seymoura Paperta a jazyk Logo na výuku programovania
  - 1969 PhD práca na tému vizuálne orientovaný programovací jazyk
- 1970 – prešiel do Xerox Parc v Palo Alto
- Tu vymyslel projekt Dynabook
  - Počítač pre deti
    - Predchodca tabletu a e-booku
  - Rozvinul koncept objektovo-orientovaného programovania (pôvodne zo Simula 67)
  - Tvorca moderného GUI na báze okien
    - Jazyk a prostredie Smalltalk



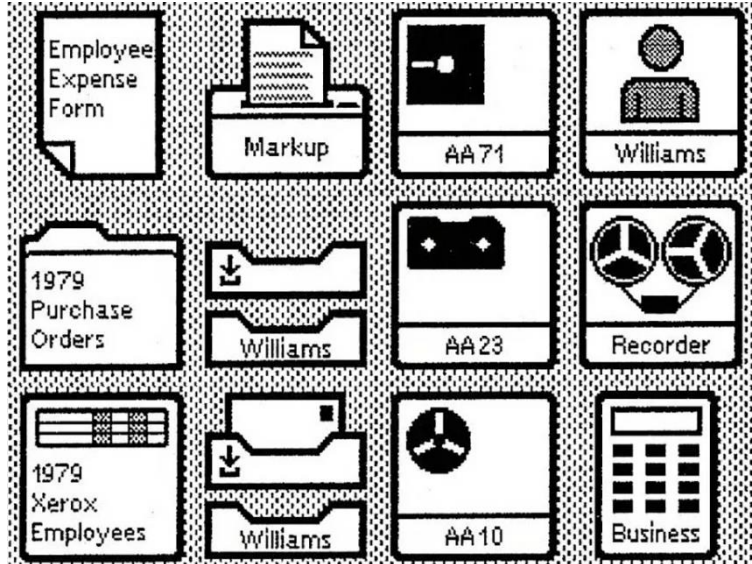
# Kto je to?

- Alan Kay s prototypem Dynabooku
  - Autor fotografie: Marcin Wichary, 2008, CC BY 2.0

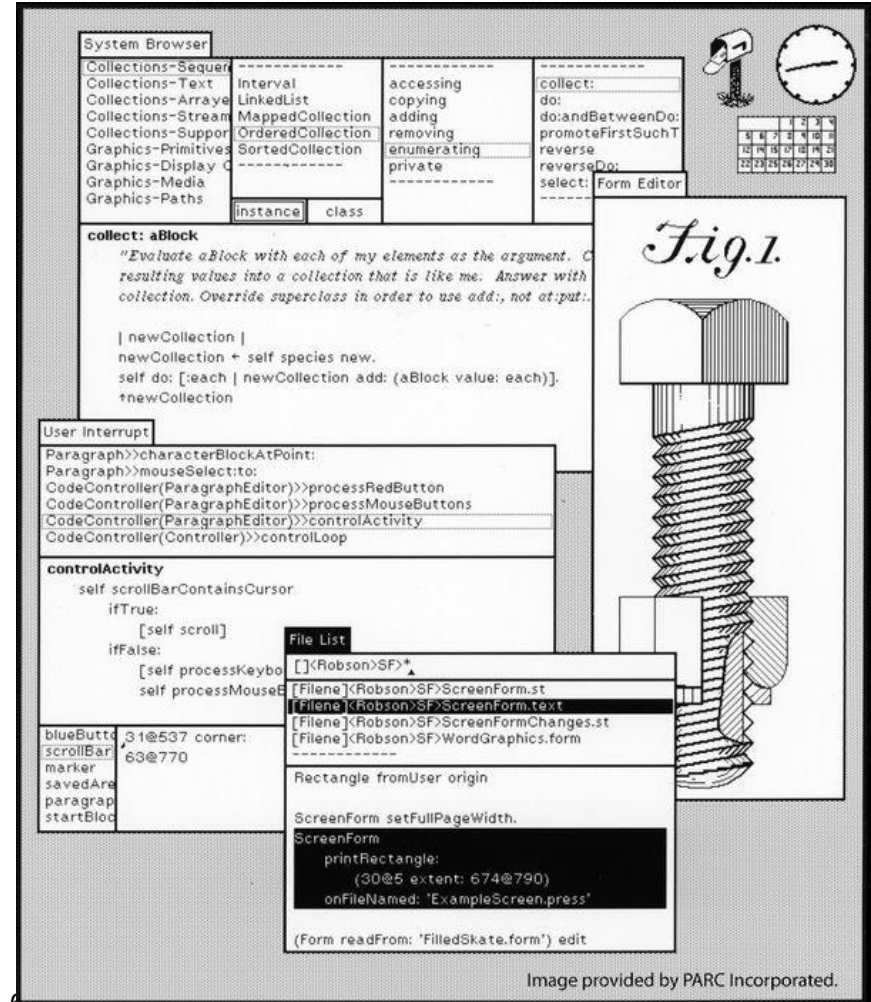


# Kto je to?

## Prvý GUI v Dynabooku



## GUI neskoršieho Smalltalku



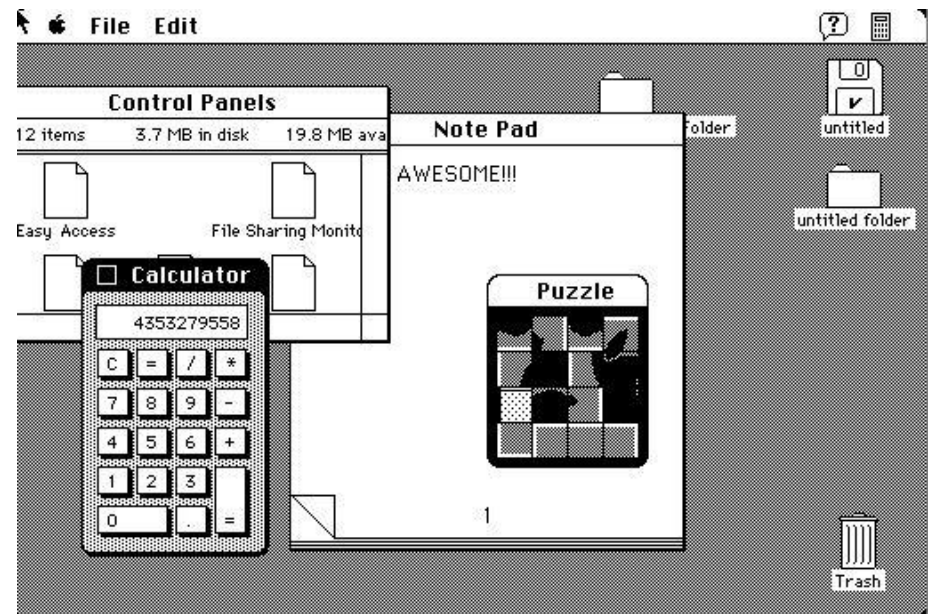


# Kto je to?

## ■ 1979 Jobs navštívil Xerox PARC

“It was one of those sort of apocalyptic moments. I remember within ten minutes of seeing the graphical user interface stuff, just knowing that every computer would work this way someday. It was so obvious once you saw it. It didn't require tremendous intellect. It was so clear,” said Jobs. “When I saw an Alto [that had

- Ponúkol 100 000 akcií Apple za zasvätenie do vývoja v Xerox Parc
- Apple prvý raz GUI a myš zakomponoval do počítača Apple Lisa (1983) – neúspešný produkt
- Apple Macintosh (1984) prvý masovo vyrábaný počítač s GUI



# JSF

# Prínosy JSF

- Väčšia podpora vzoru MVC
- Komponentový model pre Pohľad
  - Bohatšia funkcionálnosť než JSP
  - Rozširovateľnosť
- Model prepojený s Pohľadom
  - S Pohľadom je prepojený objekt typu Managed Bean
  - Objekt spravuje rámec JSF
- Všeobecný Riadiaci element
  - Poskytuje JSF automaticky

Na prednáške iba základné info o JSF.

Podrobnejšie info je možné nájsť v Učebnici

# Šablónový pohľad v JSF - Facelet

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
```

```
<f:view>
```

JSF značky

```
<head> <title> Vyhľadanie účtu </title> </head>
```

Zodpovedá <table>

```
<body> <h:form>
```

Zodpovedá  
<input type="text">

```
<h:panelGrid columns="2" title="Položky účtu">
```

```
<h:outputText value="Zadajte číslo účtu: "
```

styleClass= "display:table-cell" />

```
<h:inputText id="idUctuField" label="Číslo účtu"
```

Prepojenie na Model  
(pri submit sa  
automaticky vloží  
(setXXX) do Modelu)

value="#{ucetManBean.idUctu}"  
styleClass= "display:table-cell" />

# Šablónový pohľad v JSF - Facelet

```
<h:commandButton id="submit"
```

```
action="#{ucetManBean.vyhľadajUcet}"
```

```
value="Vyhľadaj účet"
```



```
</h:panelGrid>
```

```
</h:form>
```

```
</body>
```

```
</f:view>
```

```
</html>
```

Po submit z tlačidla sa  
automaticky vyvolá táto metóda  
Modelu

# Model v JSF – Managed Bean

**@ManagedBean (name="ucetManBean")**

**@SessionScoped**

public class **UcetManBean** {

Názov objektu použitý v Pohľade  
(najlepšie: rovnaké ako meno triedy)

private String idUctu;

private BigDecimal stavUctu;

Obsahuje všetky atribúty  
použité v Pohľade JSF stránky)

public UcetManBean() { }

public String getIdUctu () { return idUctu; }

public void setIdUctu (String idUctu) { this.idUctu = idUctu; }

public BigDecimal getStavUctu () { return stavUctu; }

public void setStavUctu (BigDecimal stavUctu)

{ this.stavUctu = stavUctu; }



# Model v JSF – Managed Bean

```
public String vyhladajUcet() {  
    try { UcetManBean ucet = getUcetService().findUcet(idUctu);  
        ...  
        return "ucetDetail";  
    } catch (AppException e) {  
        ...  
        return "ucetNenajdeny";  
    }  
}  
...  
}
```

Metóda vyvolávaná  
controllerom po  
submite z tlačidla

String pre nájdenie nasled.  
stránky

# Model v JSF – Managed Bean

- Objekt typu Managed Bean zabezpečuje JSF rámec automaticky.
  - JSF rámec ho štandardne vytvorí, keď prvý raz potrebuje hodnotu niektorej jeho vlastnosti.
  - Je však možné parametricky nastaviť aj iný čas vytvorenia objektu, napríklad pri štarte systému
- Po odoslaní stránky (submit) JSF rámec automaticky vyvolá setXXX metódy na nastavenie vlastností serverovského objektu typu Managed Bean
- Pri zobrazovaní stránky JSF rámec automaticky volá getXXX metódy serverovského objektu na získanie aktuálnych hodnôt pre serverovskú stránku.

# Navigačné pravidlá (faces-config.xml)

<navigation-rule>

<from-view-id> vyhladanieUctu.faces</from-view-id>

Aktuálny pohľad

<navigation-case>

<from-outcome>ucetDetail</from-outcome>

Návratový string  
z metódy

<to-view-id>/ucetDetail.faces</ to-view-id>

</navigation-case>

<navigation-case>

Nasl. pohľad

<from-outcome>ucetNenajdeny</from-outcome>

<to-view-id>/ucetNenajdeny.faces</ to-view-id>

</navigation-case>

</navigation-rule>

V JSF 2.0 navig. pravidlá už netreba  
písať. Framework automaticky pridá k  
výstupu funkcie príponu .xhtml a vyvolá  
takúto stránku