



Tworzenie aplikacji internetowych w języku Java

Podstawy JSP



Informacje na temat przedmiotu

- Wykładowca: Paweł Kasprowski (pawel@kasprowski.pl)
- Temat: Tworzenie aplikacji internetowych w języku Java
- Umiejętności wymagane:
 - Znajomość języka Java
 - Znajomość zasady działania relacyjnej bazy danych
 - Znajomość języka HTML
- Umiejętności uzyskiwane:
 - Tworzenie aplikacji internetowych
 - w języku Java
 - w środowisku Eclipse
 - z użyciem serwera Apache Tomcat



Plan ramowy

- Podstawy JSP
- Używanie modułów Javy w JSP
- Użycie tagów z biblioteki JSTL
- Tworzenie servletów
- Architektura MVC2
- Wprowadzenie do JSF



Aplikacja internetowa

Definicja:

- Aplikacja uruchamiana na serwerze WWW komunikująca się z użytkownikiem za pomocą przeglądarki internetowej



Zalety aplikacji internetowej

- Brak konieczności instalacji dodatkowego oprogramowania na komputerach klientów a co za tym idzie mniejsze wymagania co do jakości tych komputerów
- Łatwość aktualizacji aplikacji – w przypadku zmiany wersji nie trzeba aktualizować plików na każdym komputerze a wystarczy aktualizacja na serwerze
- Potencjalna możliwość zdalnego korzystania z aplikacji



Wady aplikacji internetowej

- Uboższy interfejs użytkownika (nie wszystko da się zrobić w przeglądarce)
- Większy nakład pracy projektowej, a co za tym idzie nieco większy koszt wykonania
- Większe obciążenie serwera (kosztem mniejszego obciążenia komputerów użytkowników)



Dynamiczne wstawki do tekstu html

Trzy najważniejsze rozwiązania:

- PHP – najpopularniejsze, własny język oparty początkowo na języku Perl
- ASP – propozycja Microsoft, wstawki w VisualBasic'u
- JSP – wstawki w języku Java



JSP – Java Server Pages

- Tworzenie dokumentów html ze wstawkami w Javie
- Dokumenty przekształcane są w servlety – klasy generujące html



Serwery JSP

- Przykładowe serwery:
 - Blazix
 - Tomcat
 - Websphere
- Standardowa funkcjonalność serwera WWW
- Kompilacja stron JSP do servletów



Tworzenie aplikacji internetowej

- Plik WAR
- Dowolne pliki html, jsp
- Katalog WEB-INF
 - Pliki konfiguracyjne (web.xml)
 - Klasy (serwlety i inne) – katalog classes
 - Biblioteki (jary) – katalog lib
- Elementy w WEB-INF nie są bezpośrednio widoczne przez WWW



Przygotowanie środowiska

- Instalacja Java JDK
- Instalacja Apache Tomcat (darmowy)
 - katalog główny: webapps/ROOT
 - aplikacje: webapps/<nazwa_aplikacji>
 - konfiguracja: conf/server.xml
- Instalacja Eclipse
- Instalacja plug-in'u Sysdeo do Eclipse'a



Nasze narzędzie pracy





Historia Eclipse

- Założyciele (2001): Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft, Webgain (Board of Stewards)
- Kolejni członkowie to m.in.: HP, Oracle, SAP, Ericsson, Intel
- Licencja CPL (Common Public License)
- Aktualna wersja: 3.4 (Ganymede)
- Podstawowe narzędzie pracy w wielu firmach programistycznych!



Rozszerzalność

- Eclipse jest narzędziem uniwersalnym:
 - Eclipse IDE for Java Developers
 - Eclipse IDE for C/C++ Developers
 - PHPEclipse
- Istnieją dedykowane wersje Eclipse'a (komercyjne):
 - Red Hat Developer Studio
 - IBM Data Studio
 - Borland JBuilder
- Są to Eclipse'y rozszerzone o własne wtyczki firmy



Przykładowe wtyczki

- Amateras HTMLEditor
- Sysdeo TomcatPlugin
- DBViewer Plugin
- AzzuriClay – modelowanie baz danych
- HibernateTools – dostęp do baz danych
- Omondo – EclipseUML
- Visual Editor – tworzenie GUI
- PHPEclipse
- Tworzone z użyciem: Plug-In Development Environment



Instalacja wtyczek

- Help->Software Updates
- Przebranie pakietu do katalogu plugins



Stworzenie projektu

- New Tomcat project: aplikacja
- Katalog /webapp
- Plik index.jsp
- Start Tomcat'a
- Sprawdzenie działania
 - <http://localhost:8080>
- Stworzenie dokumentu startowego (index.jsp)
- Sprawdzenie działania
 - <http://localhost:8080/aplikacja>



Najprostszy JSP

```
<HTML>
```

```
<BODY>
```

```
Hello! The time is now <%= new java.util.Date() %>
```

```
</BODY>
```

```
</HTML>
```



Wstawki w kodzie (scriptlet'y)

```
<HTML>
<BODY>
<%
    System.out.println( "Evaluating date now");
    java.util.Date date = new java.util.Date();
%>
...
Hello! The time is now <%= date %>
</BODY>
</HTML>
```



Scriptlet tworzący tekst

```
<HTML>
<BODY>
<%
    System.out.println( "Evaluating date now");
    java.util.Date date = new java.util.Date();
%>
...
Hello! The time is now
<%
    out.print(date);
%>
</BODY>
</HTML>
```



Formularze

- Możliwość wysyłania danych do serwera
- Proste pola edycyjne
- Pola bardziej zaawansowane – radio, checkbox, select
- ACTION – gdzie wysłać wprowadzone dane



Tworzenie formularzy

- Najprostszy formularz:

```
<form action="http://adres.pl/zarejestruj">  
  ...  
  Imię: <input type="text" name="imie"><br>  
  ...  
  <input type="submit" value="Wyślij">  
</form>
```

Imię:

Wyślij



Znaczniki

```
<input type="text">
```

```
<input type="submit">
```

```
<input type="image">
```

```
<input type="password">
```

```
<input type="radio">
```

```
<input type="checkbox">
```

```
<input type="file">
```

```
<input type="hidden">
```



Radiobutton

- Wybór jednej z opcji

```
<form>  
  <input type="radio" name="sex" value="m">Mężczyzna<br>  
  <input type="radio" name="sex" value="k">Kobieta  
</form>
```

Mężczyzna
 Kobieta



Checkbox

- Wybór wielu opcji

```
<form>
  Lubię:<br>
  <input type="checkbox" name="banany">Banany<br>
  <input type="checkbox" name="ananasy">Ananasy
</form>
```

Lubię:
 Banany
 Ananasy



Właściwości pola INPUT

<INPUT

TYPE=text|password|checkbox|radio|submit|reset|file|hidden|image,

NAME= string,

VALUE=string,

CHECKED, SIZE=n,

MAXLENGTH=n,

ALIGN=top|middle|bottom|left|right

>



Pole select

- Wybór jednej z wielu opcji drop-down

```
<form>
<select name="samochody">
<option value="volvo">Volvo
<option value="saab">Saab
<option value="fiat">Fiat
<option value="audi">Audi
</select>
</form>
```

Two examples of a select dropdown menu. The top example shows the menu closed with 'Volvo' selected. The bottom example shows the menu open with 'Volvo' selected and other options visible.

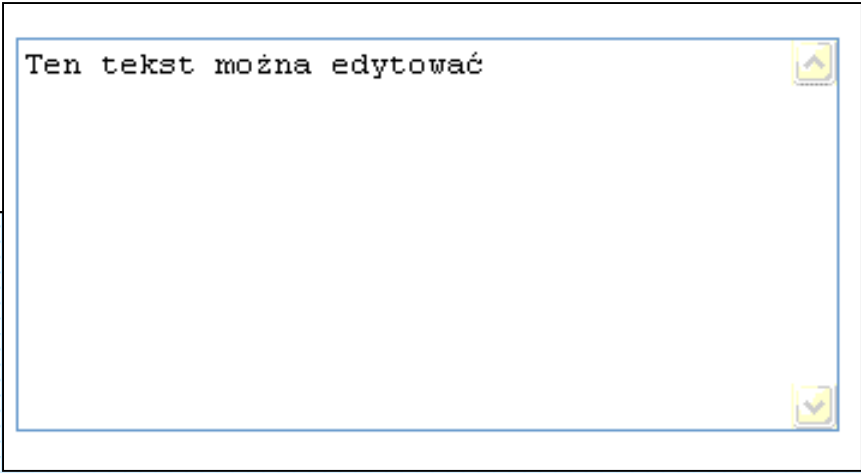
Volvo ▾
Volvo ▾
Volvo
Saab
Fiat
Audi



Pole textarea

- Edycja większego tekstu

```
<form>  
<textarea rows="10" cols="40">  
Ten tekst można edytować  
</textarea>  
</form>
```

A screenshot of a Java Swing text area widget. It is a rectangular box with a blue border. Inside, the text "Ten tekst można edytować" is displayed in a monospaced font. In the top-right and bottom-right corners, there are small yellow square buttons with black arrows pointing up and down, respectively, indicating scrollable content.

Ten tekst można edytować



Metody wysyłania danych

- Metoda GET

Odpowiednik:

`http://xyz.pl/start?imie=Adam&nazwisko=Kowalski`

- Metoda POST

Parametry wewnątrz wysyłanego pakietu – brak ograniczeń na wielkość parametrów



Obiekty dostępne w jsp

- out
 - out.write(...)
- request
 - request.getParameter(...)
 - request.getRemoteHost()
- response
 - response.sendRedirect(...)



Odczytywanie danych

- Obiekt request, metoda `getParameter(„nazwa”)`

```
<h1>Witaj <%=request.getParameter("imie")%></h1>
<p>
<%
String imie = (String)request.getParameter("imie");
if(imie.equals("Paweł"))
    out.print("Ja też mam na imię Paweł!");
%>
</p>
```



Możliwości JSP

- Mieszanie kodu i html'a

```
<TABLE BORDER=2>
  <%
  for ( int i = 0; i < 6; i++ ) {
    %>
    <TR>
      <TD>Numer</TD>
      <TD><%= i+1%></TD>
    </TR>
  }
  %>
</TABLE>
```




Możliwości JSP

- Mieszanie kodu i html'a

```
<TABLE BORDER=2>
  <%
  for ( int i = 0; i < 6; i++ ) {
    %>
    <TR>
      <TD>Numer</TD>
      <TD><%= i+1%></TD>
    </TR>
  }
  %>
</TABLE>
```

Numer	1
Numer	2
Numer	3
Numer	4
Numer	5
Numer	6



Użycie pakietów Javy

```
<HTML>
<BODY>
<%
    System.out.println( "Evaluating date now" );
    java.util.Date date = new java.util.Date();
%>
Hello! The time is now <%= date %>
</BODY>
</HTML>
```



Dyrektywa page

```
<%@ page import="java.util.*" %>
```

```
<HTML>
```

```
<BODY>
```

```
<%
```

```
    System.out.println( "Evaluating date now" );
```

```
    Date date = new Date();
```

```
%>
```

```
Hello! The time is now <%= date %>
```

```
</BODY>
```

```
</HTML>
```



Importowanie kilku pakietów

```
<%@ page import="java.util.*,java.text.*" %>
```

```
<HTML>
```

```
<BODY>
```

```
<%
```

```
    System.out.println( "Evaluating date now" );
```

```
    Date date = new Date();
```

```
%>
```

```
Hello! The time is now <%= date %>
```

```
</BODY>
```

```
</HTML>
```



Pierwsza aplikacja

- Kalkulator kredytowy
- Obliczanie raty kredytu
- Część prezentacyjna (JSP)
- Część "logiki biznesowej" (java)



Kalkulator kredytowy

- Obliczenie wysokości miesięcznej raty spłaty kredytu na podstawie jego kwoty, oprocentowania rocznego i liczby rat miesięcznych

$$rata = kwota * \frac{oprocent / 12}{1 - \frac{1}{(1 + oprocent / 12)^{liczba_rat}}}$$



Parametryzacja

- Dokument index.jsp zawiera formularz w którym użytkownik wpisuje parametry kredytu
 - kwota
 - na ile lat
- Formularz wysyłany jest do dokumentu result.jsp, który oblicza ratę



Formularz

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




Plik result.jsp

- Odczytanie parametrów:

```
String strKwota = request.getParameter("kwota");  
String strLat = request.getParameter("lat");
```
- Konwersja na liczbę (w Javie niezbędna!):

```
double kwota = Double.parseDouble(strKwota);  
int lat = Integer.parseInt(strLat);
```
- Wykonanie obliczenia:

```
double rata = kwota * (procent/12)/  
(1-(1/Math.pow(1.0+procent/12,lat*12)));
```



Stworzenie aplikacji

- Stworzenie aplikacji kredyt
- Index.jsp – formularz
- Result.jsp - obliczenie



Wady rozwiązania

- Oprocentowanie jest określone na stałe wewnątrz pliku HTML – zmiany muszą być dokonywane bezpośrednio w nim
- Sposób obliczenia raty także jest w pliku HTML
- Zmiana wyglądu strony – zmiana w pliku HTML
- A więc: łatwo popełnić błąd, który spowoduje, że aplikacja przestanie działać prawidłowo
- Dalsza rozbudowa aplikacji (np. ściąganie oprocentowania z bazy danych) bardzo utrudnione



Rozwiązanie dwuwarstwowe

- Rozdzielenie części „prezentacyjnej” od części obliczeniowej
- Programista (grafik) tworzący interfejs WWW nie musi znać wzoru na obliczanie kredytu
- Programista tworzący kod obliczający kredyt nie musi wiedzieć gdzie będzie on używany



Klasa Kredyt

```
package pl.kurs; //pakiet jest konieczny!
public class Kredyt {
    double procent = 0.05;
    double kwota;
    double lat;

    public Kredyt(double kwota, double lat) {
        this.kwota = kwota;
        this.lat = lat;
    }

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

Kredyt.java



Nowy result.jsp

```
<%  
Kredyt kredyt = new Kredyt(  
    Double.parseDouble(request.getParameter("kwota")),  
    Double.parseDouble(request.getParameter("lat"))  
);  
out.write("<p>Rata wynosi: "+kredyt.getRata()+"</p>");  
%>
```



Nowy result.jsp [2]

```
<%  
Kredyt kredyt = new Kredyt(  
    Double.parseDouble(request.getParameter("kwota")),  
    Double.parseDouble(request.getParameter("lat"))  
    );  
%>  
<p>Rata wynosi: <%=kredyt.getRata()%></p>
```



Dodanie konstruktora

```
package pl.kurs;
public class Kredyt {
    double procent = 0.05;
    double kwota;
    double lat;

    public Kredyt(double k, double l) { kwota = k; lat = l; }

    public Kredyt(String k, String l) {
        kwota = Double.parseDouble(k);
        lat = Double.parseDouble(l);
    }

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

Kredyt.java



Dodanie konstruktora

```
package pl.kurs;
public class Kredyt {
    double procent = 0.05;
    double kwota;
    double lat;

    public Kredyt(double k, double l) { kwota = k; lat = l; }

    public Kredyt(String k, String l) {
        this(Double.parseDouble(k), Double.parseDouble(l));
    }

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



Nowy result.jsp [3]

```
<%  
Kredyt kredyt = new Kredyt(  
    request.getParameter("kwota"),  
    request.getParameter("lat"))  
    );  
%>  
<p>Rata wynosi: <%=kredyt.getRata()%></p>
```

Brak jakiegokolwiek kodu obliczającego cokolwiek!



Obsługa wyjątków

- Gdy coś nie zadziała tak jak trzeba – należy to przewidzieć i "wyłapać" w programie
- Jeśli nie wyłapiemy wyjątku – rezultaty mogą być niespodziewane
- Wyłapywanie wyjątków:

```
try{  
    ...tu kod, który może wygenerować wyjątek...  
}catch(Exception ex) {...tu kod obsługujący...}
```



Miejsce obsługi wyjątków

- Strona HTML
 - zaleta – można bezpośrednio wypisywać komunikaty
 - wada – strona staje się nieczytelna
- Kod klasy
 - zaleta – kod HTML jest czytelny
 - wada – w kodzie HTML często i tak trzeba zareagować na wyjątek więc należałoby go i tak "podać" do HTMLa



try-catch w result.jsp

```
<%  
try{  
    Kredyt kredyt = new Kredyt(  
        request.getParameter("kwota"),  
        request.getParameter("lat"))  
    );  
    %>  
    <p>Rata wynosi: <%=kredyt.getRata()%></p>  
    <%  
}catch(NumberFormatException ex) {out.write("Nie udało się  
obliczyć raty!");  
%>
```



try-catch w result.jsp

```
<%  
try{  
    Kredyt kredyt = new Kredyt(  
        request.getParameter("kwota"),  
        request.getParameter("lat")  
    );  
    out.write("<p>Rata wynosi: "+kredyt.getRata()+"</p>");  
}catch(NumberFormatException ex) {out.write("Nie udało się  
obliczyć raty!");  
}%>
```



Dyrektywa errorPage

- Ustawienie strony do obsługi błędu
`<%@ page errorPage="/error.jsp" %>`
- Strona error.jsp – dostęp do obiektu exception
`<%@ page isErrorPage="true" %>`
`<% response.setStatus(500); %>`
`<%= exception.getClass().getName() %>`
- Wada: reakcja polega tylko na poinformowaniu o błędzie