

JavaServer Pages

Koncepcje i zastosowania

Marek Wojciechowski, Maciej Zakrzewicz
Politechnika Poznańska, Instytut Informatyki

Plan prezentacji

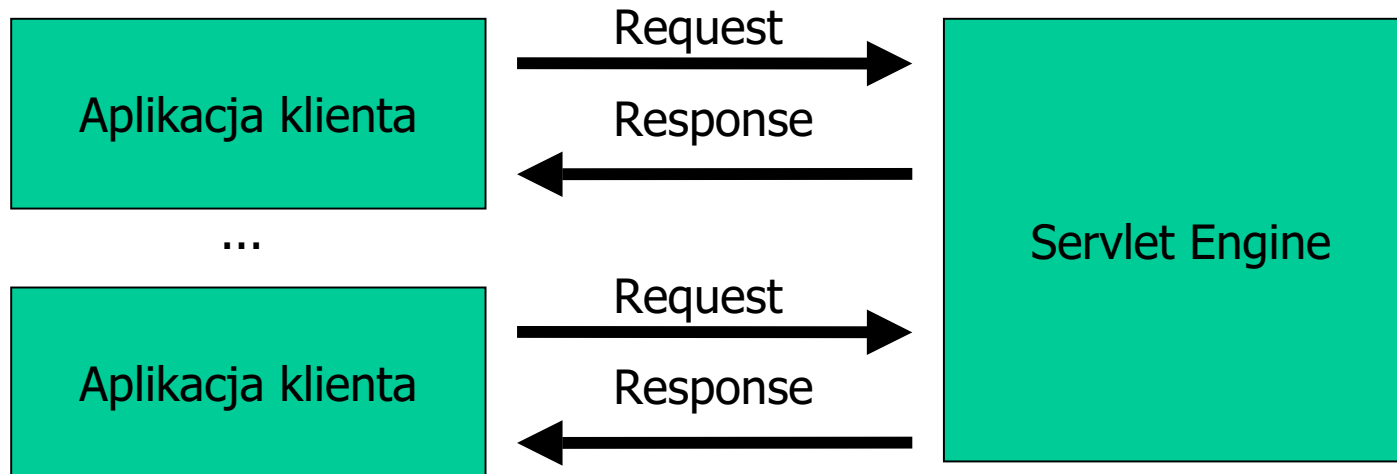
- 1) Architektura, zalety i wady serwletów
- 2) Architektura JavaServer Pages (JSP),
tworzenie dokumentów JSP
- 3) Serwlety i JSP w Oracle
- 4) Dostęp do baz danych z aplikacji JSP
- 5) Podsumowanie

Czym jest Serwlet?

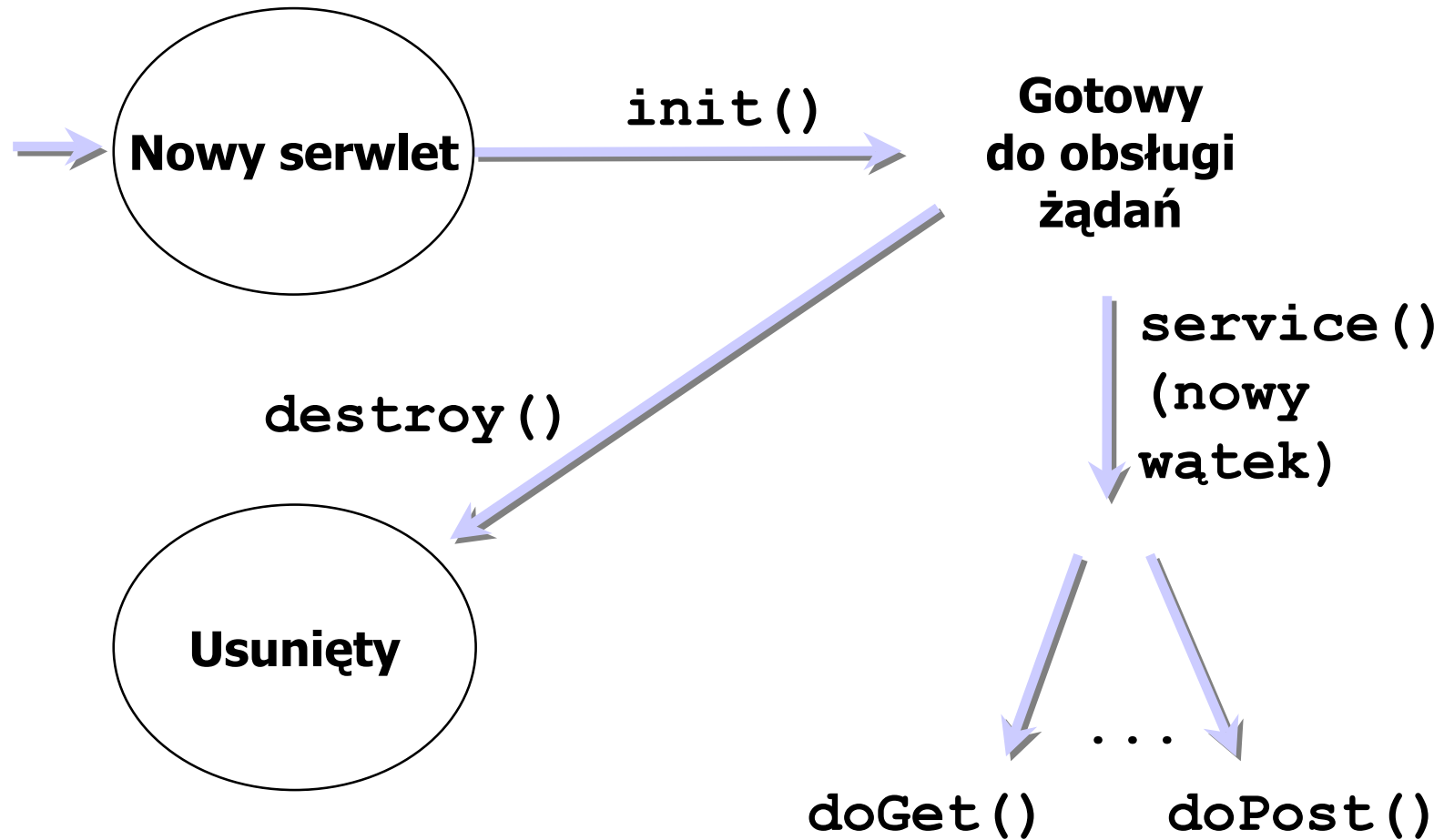
- Aplikacja w Javie pracująca po stronie serwera
- Najczęściej zadanie serwleta polega na dynamicznym generowaniu dokumentów HTML (serwlety HTTP)
- Serwlet oczekuje na żądania klientów i obsługuje je
- Serwlety stanowią alternatywę dla CGI
 - są szybsze i efektywniejsze
 - mogą obsługiwać wielu klientów jednocześnie
 - oszczędniej gospodarują zasobami
 - łatwiejsze do tworzenia
 - standardowe API (Java Servlet API)
- Serwlety mogą łączyć się z wieloma serwerami

Architektura serwletów

- Gdy klient odwołuje się do serwleta, serwletowi przekazywane są 2 obiekty
 - `ServletRequest` (komunikacja klient -> serwer)
 - `ServletResponse` (komunikacja serwer -> klient)
- Serwlet domyślnie jest wielowątkowy (chyba, że implementuje interfejs *SingleThreadModel*)



Cykl życia serwleta



Prosty serwlet - Przykład

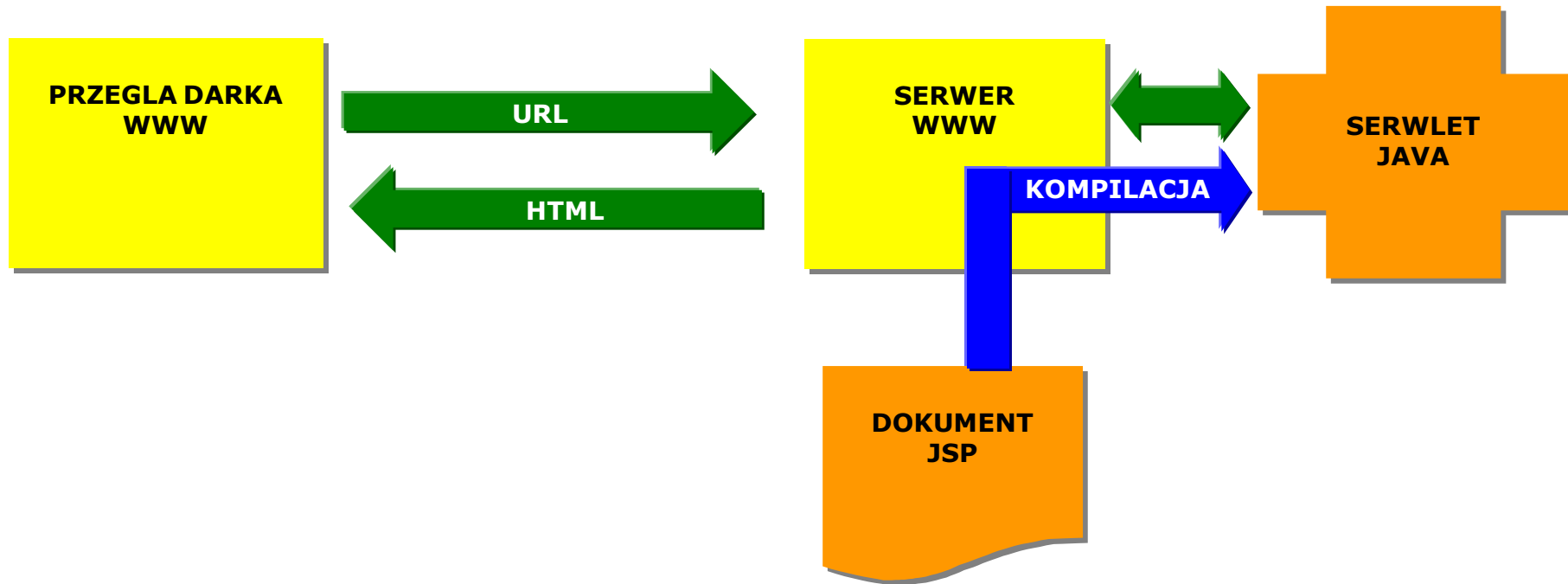
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        int term1, term2;
        res.setContentType("text/html");
        out = res.getWriter();
        out.println("<HEAD><TITLE> Serwlet!</TITLE></HEAD><BODY>");
        term1 = Integer.parseInt(req.getParameter("par1"));
        term2 = Integer.parseInt(req.getParameter("par2"));
        out.println("<H1> Multiplication result: " + term1 * term2);
        out.println("</H1></BODY>");
        out.close();
    }
}
```



Czym są JavaServer Pages?

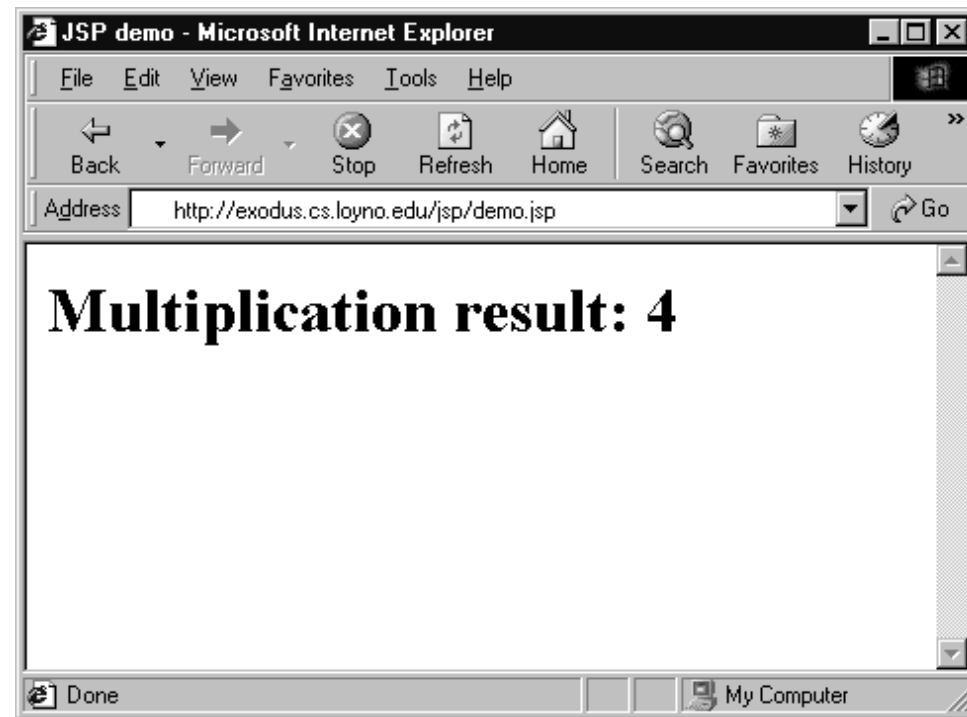
- Podstawowa wada serwletów Java: brak możliwości oddzielenia kodu w języku Java generującego dynamiczną zawartość dokumentu od części statycznej (HTML) odpowiedzialnej za ogólny wygląd strony
- Technologia JavaServer Pages (JSP) jest rozszerzeniem architektury serwletów Java
- Aplikacje JavaServer Pages mają postać dokumentów HTML (XML) zawierających zagnieżdżony kod w języku Java
- W odpowiedzi na żądanie użytkownika, serwer WWW wykonuje zagnieżdżony kod i zwraca wygenerowany dokument HTML
- Typowy sposób przetwarzania dokumentów JSP polega na ich translacji do serwletów, które są następnie kompilowane i uruchamiane przez serwer WWW

Architektura JSP



JSP - Przykład

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<%! int result; %>
<% result = 2*2; %>
<H1> Multiplication result:
<%= result %>
</H1>
</BODY>
</HTML>
```



Komponenty dokumentu JSP

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<%! int result; %>
<% result = 2*2; %>
<H1> Multiplication result:
<%= result %>
</H1>
</BODY>
</HTML>
```

Dyrektywy

(page - globalne właściwości dokumentu:
język, deklaracje import; include; taglib)

Deklaracje (globalnych zmiennych i metod)

Kod Java (ang. **Scriptlets**)

Wyrażenia (wyświetlane po wyznaczeniu ich wartości)

Generacja serwleta (1/2)

Dokument JSP:

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<%! int result; %>
<% result = 2*2; %>
<H1> Multiplication result:
<%= result %>
</H1>
</BODY>
</HTML>
```

Serwlet Java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet{
    int result;

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML><HEAD><TITLE>JSP demo");
        out.println("</TITLE></HEAD><BODY>");

        result = 2*2;
        out.println("<H1> Multiplication result:");
        out.println(result);
        out.println("</H1></BODY></HTML>");
    }
}
```

Generacja serwleta (2/2)

Dokument JSP:

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<%! int result; %>
<% result = 2*2; %>
<H1> Multiplication result:
<%= result %>
</H1>
</BODY>
</HTML>
```

Serwlet Java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet{
    int result;

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML><HEAD><TITLE>JSP demo");
        out.println("</TITLE></HEAD><BODY>");
        result = 2*2;
        out.println("<H1> Multiplication result:");
        out.println(result);
        out.println("</H1></BODY></HTML>");
    }
}
```

Obiekty dostępne w dokumentach JSP

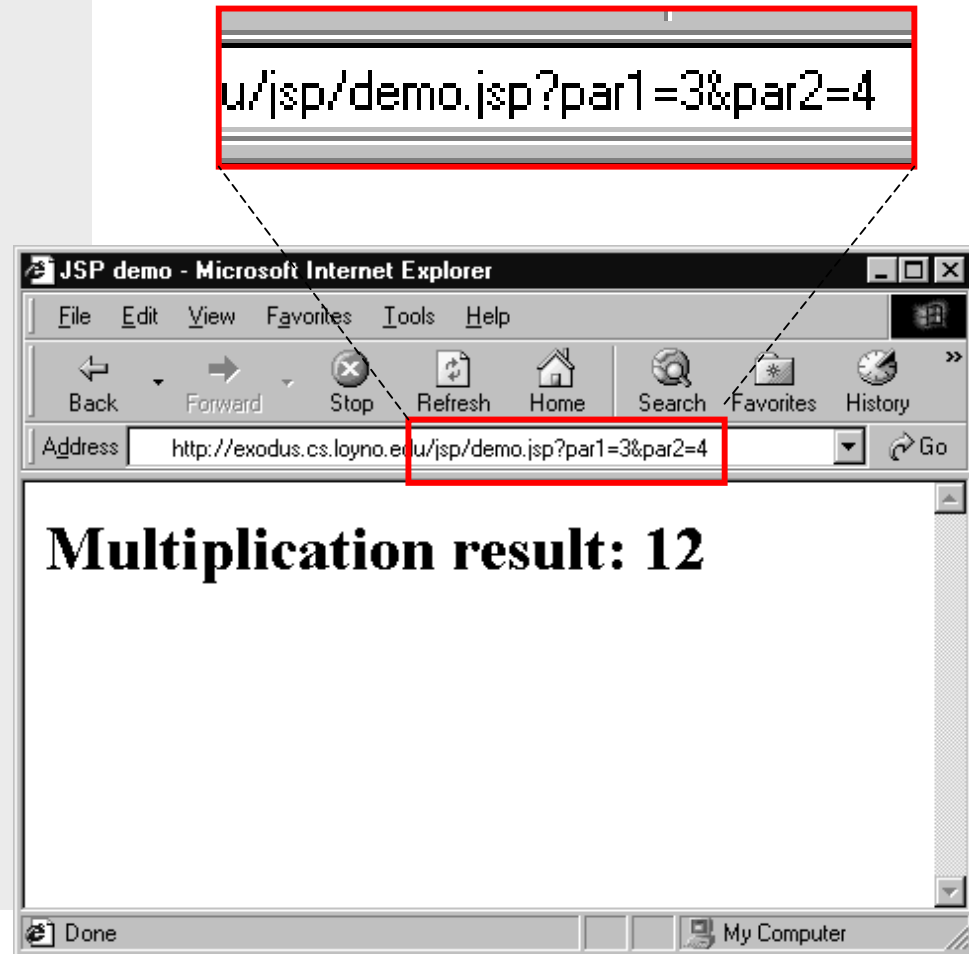
- Explicit objects – obiekty tworzone jawnie przez programistę:
 - Obiekty tworzone w kodzie Java zagnieżdżonym w dokumencie
 - Komponenty JavaBeans wykorzystywane w dokumencie
- Implicit objects – obiekty tworzone przez wewnętrzne mechanizmy JSP, dostępne standardowo w każdym dokumencie JSP:
 - page, request, response, pageContext, session, application, out, config, exception

Przekazywanie parametrów

- Dokumentowi JSP mogą być przekazane wartości parametrów wejściowych
- Parametry mogą być zakodowane w adresie URL lub wysyłane przez formularz HTML
- Implementacja mechanizmu przekazywania parametrów jest taka sama jak w przypadku serwletów: dostęp do wszystkich parametrów wywołania jest możliwy poprzez obiekt `request`
- Aby odczytać wartość parametru należy wywołać metodę `request.getParameter()`

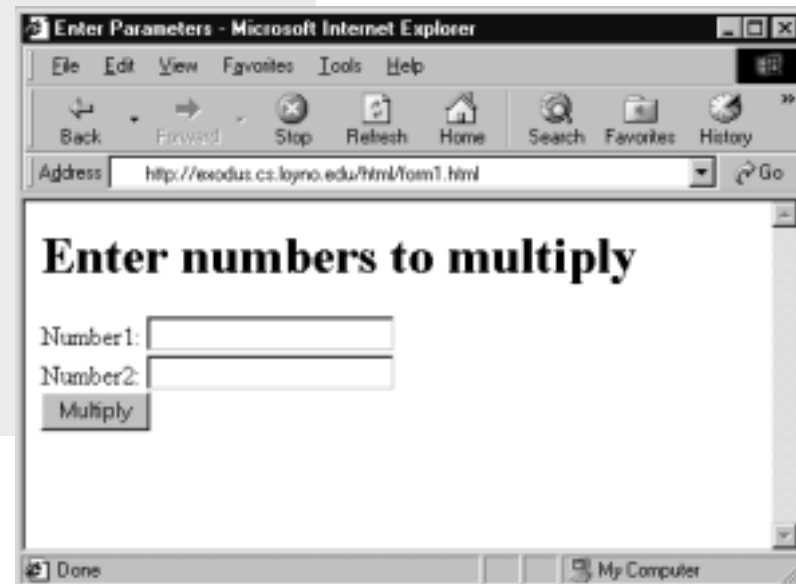
Przekazywanie parametrów - Przykład

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<%! int result;
    int term1, term2; %>
<% term1 = Integer.parseInt(
    request.getParameter("par1"))
    term2 = Integer.parseInt(
    request.getParameter("par2"))
    result = term1 * term2; %>
<H1> Multiplication result:
<%= result %>
</H1>
</BODY>
</HTML>
```



Przekazywanie parametrów do JSP za pomocą formularzy HTML

```
<HTML>
<HEAD>
<TITLE>Enter Parameters</TITLE>
</HEAD>
<BODY>
<H1>Enter numbers to multiply</H1>
<FORM ACTION='http://exodus.cs.loyno.edu/jsp/demo.jsp'>
Number1:
<INPUT TYPE='text' NAME='par1'><BR>
Number2:
<INPUT TYPE='text' NAME='par2'><BR>
<INPUT TYPE='submit' VALUE='Multiply'>
</FORM>
</BODY>
</HTML>
```



Komponenty JavaBeans

- Kod w języku Java zagnieżdżony w dokumencie JSP może osiągać duże rozmiary utrudniając pielęgnację aplikacji
- Technologia JavaBeans pozwala programistom na faktyczne oddzielenie kodu w języku Java od statycznej części dokumentu JSP (kod umieszczany jest w klasach języka Java)
- Dokument JSP może odwoływać się do komponentów JavaBeans (klas) za pomocą znaczników `<jsp:useBean>`, `<jsp:setProperty>` i `<jsp:getProperty>`
- Komponentem JavaBeans wykorzystywanym przez JSP może być dowolna skompilowana klasa w języku Java (posiadająca zbiór właściwości i metody do ich odczytu i zmiany)

JavaBeans - Przykład

```
public class taxCalc {
    double income;

    public void setIncome(double i) {
        income = i;
    }

    public double getTax() {
        double tax;

        if (income < 2450)
            tax = 0.05 * income;
        else if (income < 6100)
            tax = (0.07*(income-2450)+123);
        else
            tax = (0.09*(income-6100)+378);
        return tax;
    }
}
```

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<jsp:useBean
    id="incomeTax"
    class="taxCalc"/>
<jsp:setProperty
    name="incomeTax"
    property="income" />
<H1> The tax is:
<jsp:getProperty
    name="incomeTax"
    property="tax" />
USD</H1>
</BODY>
</HTML>
```

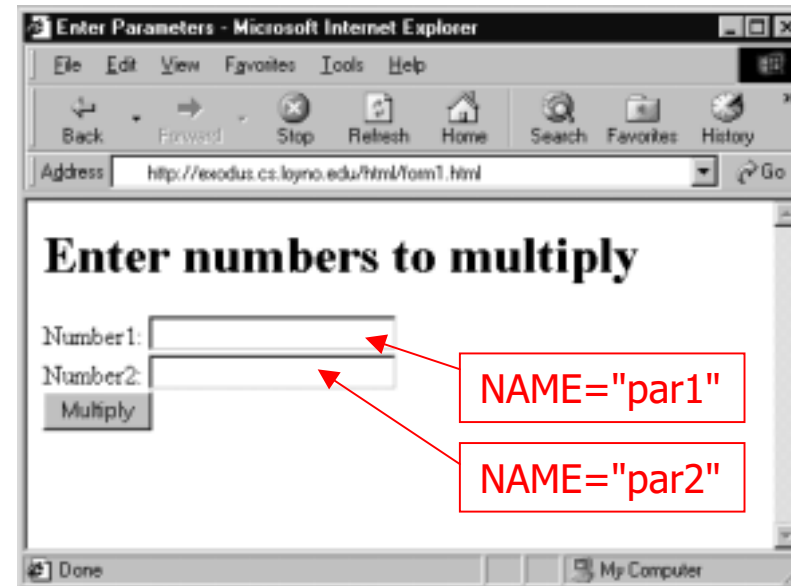
utworzenie obiektu
incomeTax klasy
taxCalc

wywołanie metody
setIncome()
obiektu incomeTax

wywołanie metody
getTax() obiektu
incomeTax;
wyświetlenie
wyniku

Przekazywanie parametrów JSP do komponentów JavaBeans - Przykład

```
<%@ page language="java" %>
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<jsp:useBean
    id="mul" class="Multi"/>
<jsp:setProperty
    name="mul" property="*" />
<H1> Multiplication result:
<jsp:getProperty
    name="mul" property="result" />
</H1>
</BODY>
</HTML>
```



```
public class Multi {
    int par1, par2;

    public void setPar1(int newPar1) {
        par1 = newPar1; }
    public void setPar2(int newPar2) {
        par2 = newPar2; }
    public int getResult() {
        return par1 * par2; }
}
```

Zasięg obiektów będących instancjami JavaBeans

- Obiekty w JSP posiadają swój zasięg (ang. scope) określający ich widzialność i czas życia
- Dla instancji JavaBeans zasięg można określić jawnie za pomocą atrybutu `scope` znacznika `<jsp:useBean>`
- Zasięg obiektu przybiera jedną z następujących wartości:
 - `scope= "page"`
 - `scope= "request"`
 - `scope= "session"`
 - `scope= "application"`
- Korzystanie z obiektów o zasięgu większym niż bieżący dokument pozwala na:
 - Oszczędniejsze gospodarowanie zasobami
 - Współdzielenie obiektów przez dokumenty obsługujące to samo żądanie

Serwlety i JSP w Oracle

- Serwlety i JSP są wspierane przez:
 - Oracle8i Release 8.1.7
 - Oracle Servlet Engine (OSE)
 - Oracle HTTP Server powered by Apache + Apache/Jserv + mod_jserv + mod_ose
 - Oracle9i Application Server
 - Oracle HTTP Server powered by Apache + Apache/Jserv + mod_jserv + mod_ose
- Możliwe strategie uruchamiania JSP w Oracle:
 - Uruchamianie w środowisku Apache/JServ poprzez Oracle HTTP Server (z użyciem mod_jserv)
 - Uruchamianie w środowisku Oracle Servlet Engine poprzez Oracle HTTP Server (z użyciem mod_ose)
 - Uruchamianie w środowisku Oracle Servlet Engine z wykorzystaniem OSE jako serwera HTTP

Dostęp do bazy danych z aplikacji JSP

- Dostęp do baz danych z aplikacji JSP realizowany jest na takich samych zasadach jak w przypadku innych aplikacji tworzonych w języku Java (serwletów, apletów, procedur składowanych)
- Dostęp odbywa się za pomocą interfejsu JDBC, z wykorzystaniem sterowników JDBC dostarczanych przez producentów systemów zarządzania bazami danych
- Operacje dostępu do bazy danych mogą być realizowane bezpośrednio poprzez interfejs JDBC (dynamiczny SQL) lub w oparciu o standard SQLJ (statyczny SQL)

Sterowniki JDBC

- sterownik typu most JDBC-ODBC
- sterowniki firmowe producentów baz danych
- sterowniki JDBC oferowane przez Oracle:
 - JDBC Thin
 - w 100% napisany w czystej Javie
 - może być pobrany przez sieć wraz z apletem Java
 - JDBC OCI
 - wykonuje wywołania OCI do "fabrycznego" sterownika, preinstalowanego po stronie klienta
 - wykorzystywany w aplikacjach języka Java, po stronie serwera lub w warstwie pośredniej
 - Server-side JDBC driver
 - wykorzystywany przez aplikacje Java uruchamiane w bazie danych
 - wysoce efektywny, ściśle zintegrowany z RDBMS

Dynamiczny SQL w języku Java

- Wywołania JDBC korzystają z "dynamicznego SQL"
- Treść poleceń SQL może być budowana dynamicznie w trakcie pracy programu (duża elastyczność)
- Polecenia SQL są analizowane w czasie pracy programu
- Na etapie kompilacji nie ma możliwości wykrywania błędów składniowych SQL oraz kontroli poprawności odwołań do obiektów bazy danych

JDBC - Przykład

```
import java.sql.*;
class DB1 {
    ... // definicje pól i metod

    public static double placaJDBC() throws SQLException
    {
        Connection conn = null;
        double maxplaca = 0;
        try {
            conn = new
                oracle.jdbc.driver.OracleDriver().defaultConnection();
            Statement stmt = conn.createStatement ();
            ResultSet rset = stmt.executeQuery ("SELECT
                max(placa_pod) from pracownicy");
            while (rset.next())
                maxplaca = rset.getDouble(1);
        } catch (SQLException e) {}
        return maxplaca;
    }
}
```

Statyczny SQL w języku Java

- standard SQLJ pozwala na zagnieżdżanie "statycznych" instrukcji SQL w kodzie w języku Java
- kod źródłowy w pliku *.sqlj jest prekompilowany do kodu w języku Java przez translator SQLJ
- polecenia SQL są zamieniane na równoważny kod języka Java (zawierający wywołania JDBC)
- w trakcie translacji weryfikowana jest poprawność polecenia SQL (składnia i odwołania do obiektów)
- zaletą jest zmniejszona liczba błędów w czasie pracy programu
- wadą jest mniejsza elastyczność aplikacji

SQLJ - Przykład

```
import java.sql.*;
class DB2 {
    ... // definicje pól i metod

    public static double placasQLJ() throws SQLException
    {
        double maxplaca = 0;
        try {
            #sql { SELECT nvl(max(placa_pod), 0.0)
                    into :maxplaca from pracownicy };
        } catch (SQLException e) {}
        return maxplaca;
    }
}
```

Dokument JSP korzystający z bazy danych - Przykład (1/2): JavaBean

```
import java.sql.*;

public class DbBean {
    Connection conn;

    public void openConnection(
        String connect_string)
        throws SQLException {

        DriverManager.registerDriver(
            new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection(
            connect_string, "scott", "tiger");
    }

    public String displayTable()
        throws SQLException {
        String result = "";
        Statement stmt;
        ResultSet rset;

        stmt = conn.createStatement();
        rset = stmt.executeQuery(
            "select ename,sal from emp");
```

```
        while (rset.next())
            result +=
                rset.getString("ename")+"<BR>";

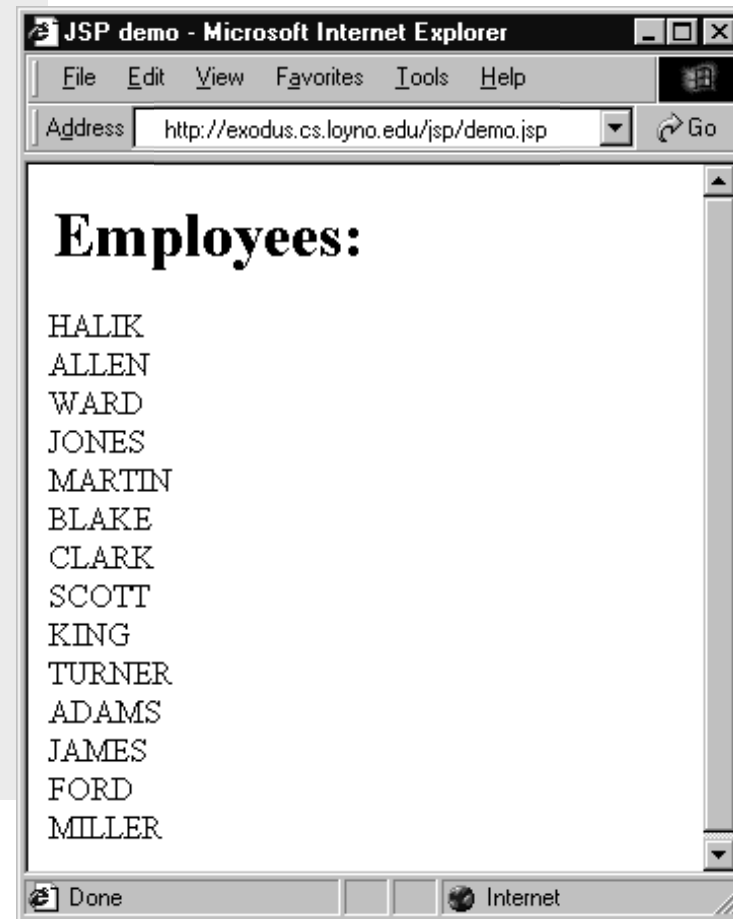
        rset.close(); stmt.close();

        return result;
    }

    public void closeConnection()
        throws SQLException
    {
        conn.close();
    }
}
```

Dokument JSP korzystający z bazy danych - Przykład (2/2): plik JSP

```
<HTML>
<HEAD>
<TITLE>JSP demo</TITLE>
</HEAD>
<BODY>
<%@ page language="java" %>
<jsp:useBean id="db" class="DbBean"/>
<% db.openConnection(
    "jdbc:oracle:thin:@srv1:1521:orcl");
    %>
<H1> Employees:</H1>
<%= db.displayTable() %>
<% db.closeConnection(); %>
</BODY>
</HTML>
```



JSP - Podsumowanie

- Technologia będąca rozszerzeniem architektury serwletów Java (oferuje taką samą efektywność)
- Separacja kodu wykonywalnego od statycznej części dokumentu (HTML)
- Pozwala na łatwy podział pracy między projektantów (grafików) i programistów języka Java
- Technologia szczególnie użyteczna w przypadku dynamicznie generowanych dokumentów, których układ graficzny jest często modyfikowany
- Zapewnia niezależność od platformy i bezpieczeństwo języka Java
- Umożliwia projektantom korzystanie z gotowych komponentów (JavaBeans)