Witryny i Portale Internetowe

TECHNOLOGIA JAVA 2 ENTERPRISE EDITION

PRZEMYSŁAW SOŁTAN
email: kerk@moskit.ie.tu.koszalin.pl

03.04.2003

Co powinieneś znać?

- Podstawy HTML
- Programowanie obiektowe
- Programowanie w Javie
- Bazy danych

Cel !

- Tworzenie stron WWW w oparciu o
  - Java Server Pages
  - servlety
- Korzystanie z usług technologii J2EE
- Wykorzystanie BAZ DANYCH
- Instalowanie i obsługa aplikacji webowych na wzorcowych serwerach
  - Tomcat
  - J2EE server

Zagadnienia poruszane na zajęciach

- Serwery WWW i serwery aplikacji
- Technologia Java 2 Enterprise Edition
- JSP i serwlety
- JDBC i SQL
- Java Bean i Enterprise Java Bean
- Transakcje
- Bezpieczeństwo
- XML

Literatura

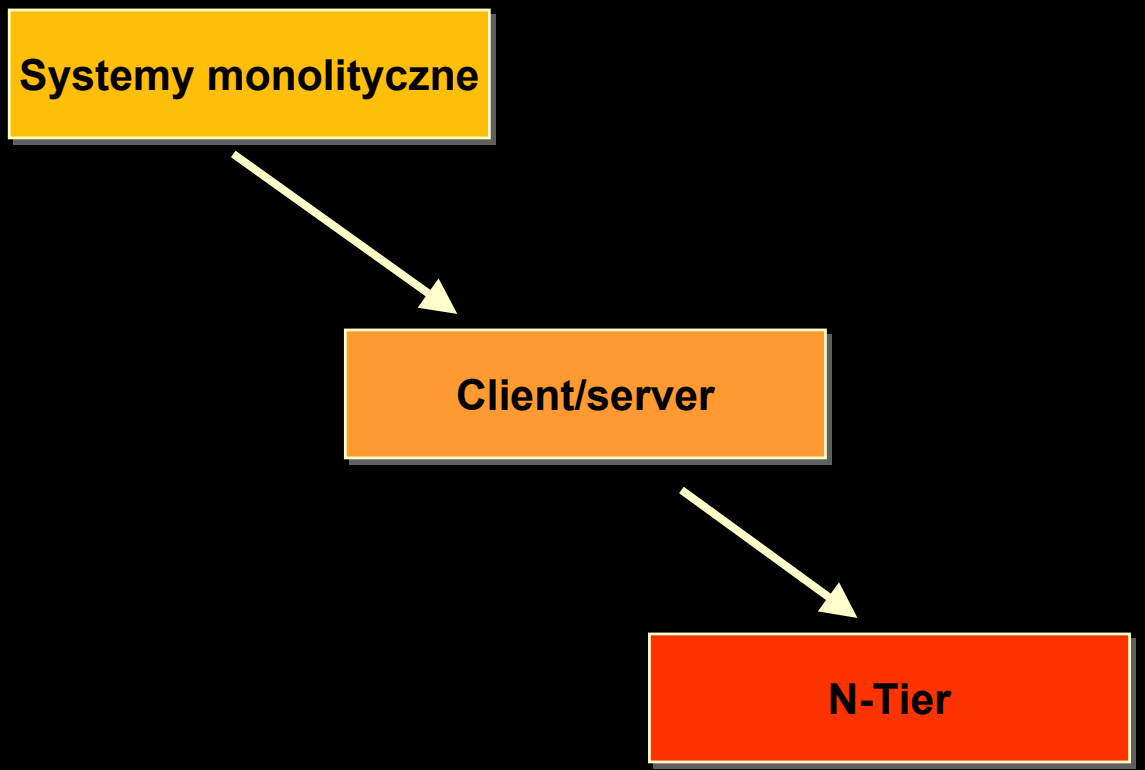
- „**Java Server Pages** - Java Server Pages oraz inne komponenty JavaPlatform” Wojciech Romowicz, Helion 2001
- „**Java Server Pages** - podręcznik z przykładami” James Goodwill, Helion 2001
- „**Java Server Pages** od podstaw” Maneesh Sahu, Translator 2001

- „**Thinking in Java** - edycja polska” Bruce Eckel, Helion 2001
- „**Java** - aplikacje bazodanowe” Michał Grochala, Helion 2001
- „Po prostu **XML**” Elizabeth Castro, Helion 2001
- „**Core** - Servlets and JavaServer Pages” Marty Hall, Prentice Hall & Sun Microsystems - **www.coreservlets.com**

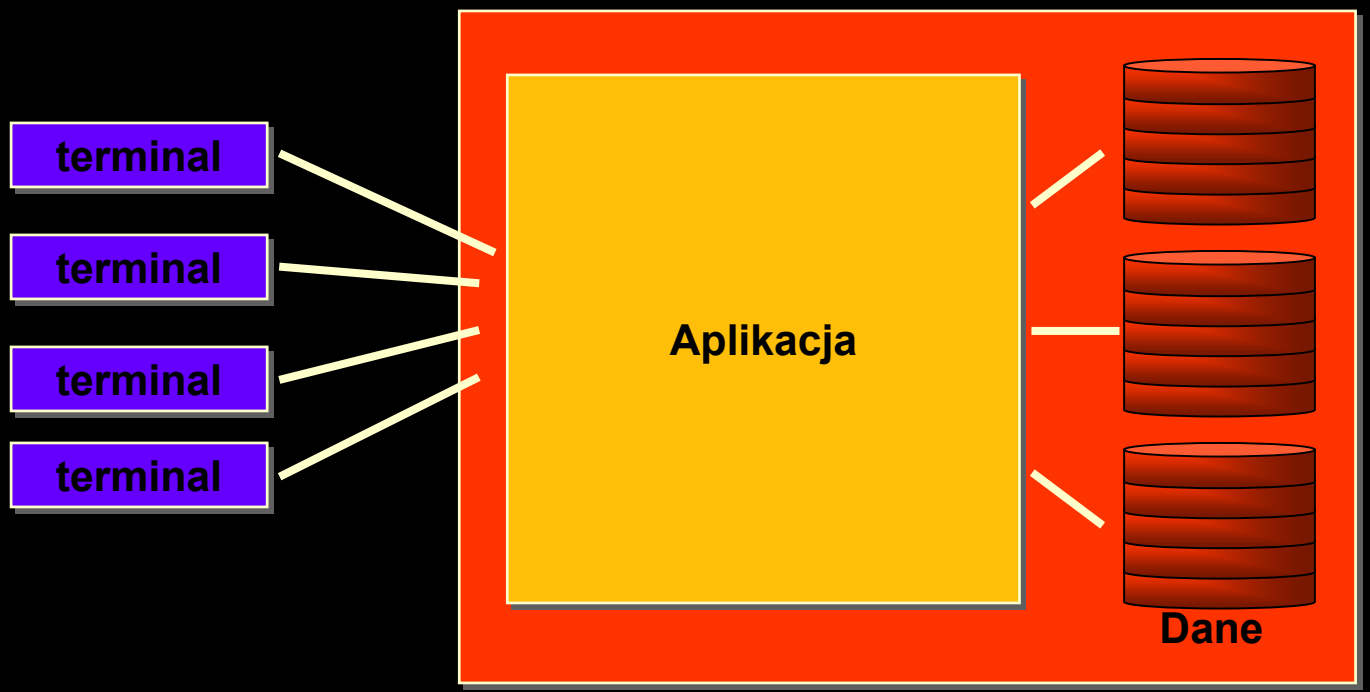
- Sun Microsystems - **www.java.sun.com**

Systemy wielowarstwowe N-tier

Ewolucja systemów wielowarstwowych



Systemy monolityczne

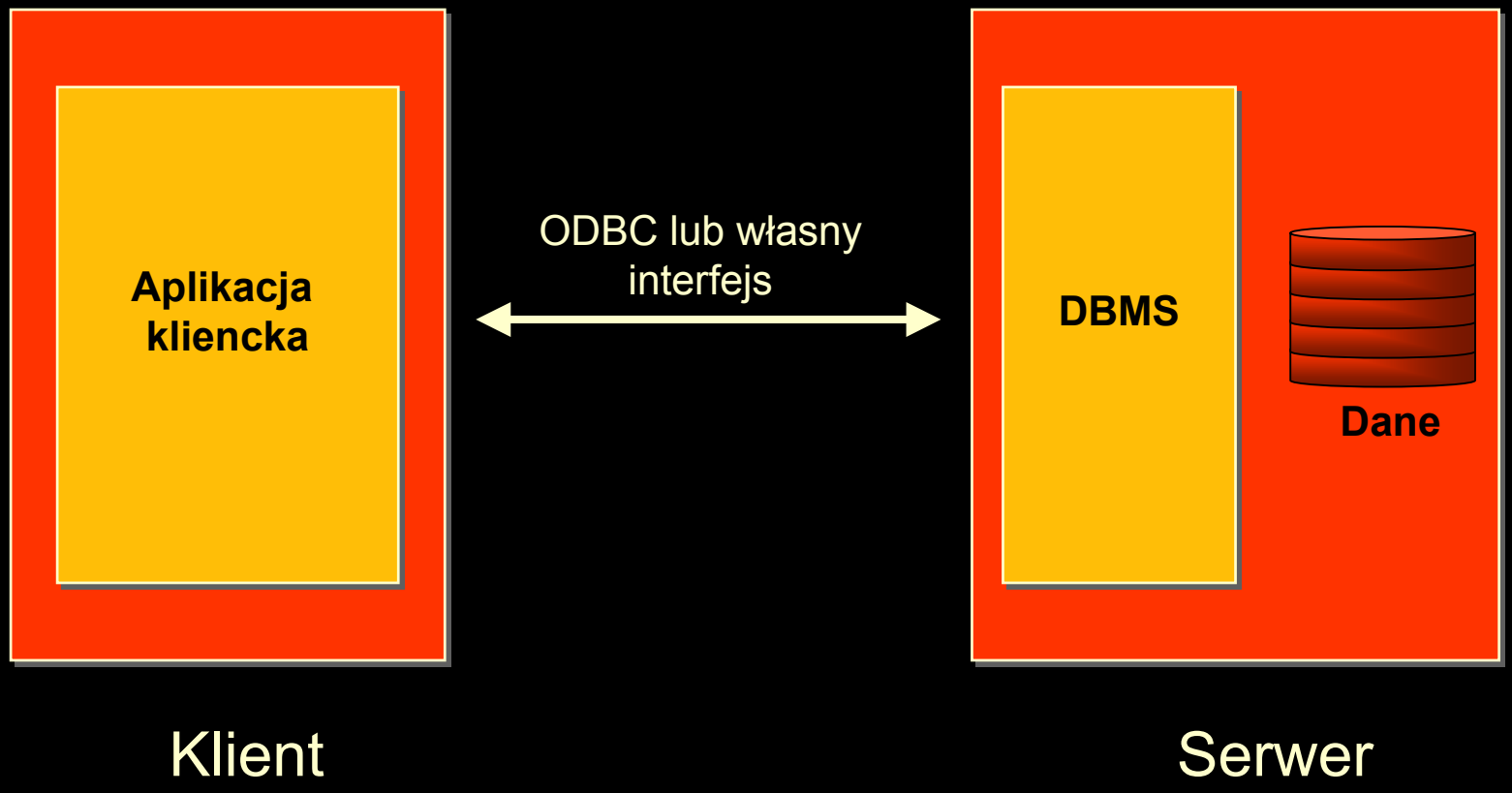


MainFrame

Systemy monolityczne

- Systemy zawierające całość logiki prezentacji, biznesu i dostępu do danych. Brak możliwości wymiany danych z innymi aplikacjami (integrowanie danych)
- Niestandardowy dostęp do danych (uzależnienie od wybranego systemu baz danych)
- Wielokrotne tworzenie (programowanie) tej samej funkcjonalności systemu
- Duże koszty i słaba wydajność

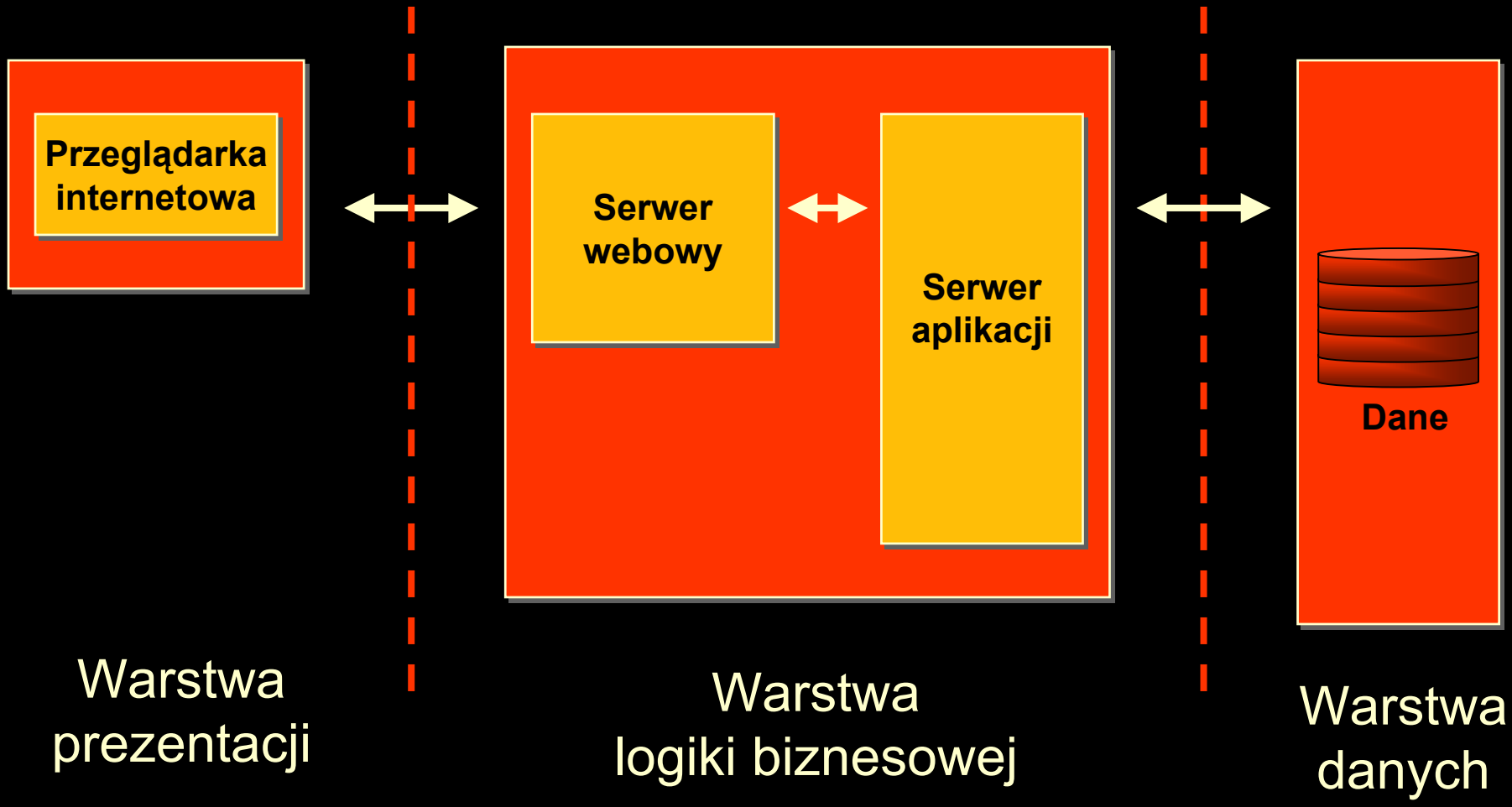
Client/Server



Client/Server

- Uproszczenie tworzenia i administracji złożonych aplikacji
- Podział aplikacji na część klienta i serwer
- Zapis danych w bazie z możliwością ich użycia przez inną aplikację
- Rozproszenie zasad biznesowych (klient i procedury składowane DBMS)
- Dostęp do danych przez wielu klientów

N-Tier


Przeglądarka internetowa
Serwer webowy
Serwer aplikacji
Dane
Warstwa prezentacji
Warstwa logiki biznesowej
Warstwa danych

N-Tier

- Technologia obiektów rozproszonych:
  - rozbicie aplikacji na samozarządzalne komponenty
  - współdziałanie komponentów pomiędzy różnymi sieciami i systemami operacyjnymi
- Integracja ze starszymi technologiami - technologia konektorów

N-Tier

- Skalowalność, wydajność
- Niezawodność i spójność danych
- Scentralizowanie zarządzania i administracja
- Zmniejszenie kosztów utrzymania systemów klienckich (klient w postaci przeglądarki WWW)
- Wielokrotne wykorzystanie kodu komponentów
- Integracja systemów monitorujących pracę serwera aplikacji

N-Tier

- Bezpieczeństwo - stosowanie firewall'i pomiędzy poszczególnymi warstwami aplikacji
- Implementacja puli zasobów (resource pooling) - mechanizmu zwiększającego wydajność aplikacji
- Łatwiejsze wykrywanie punktów spowalniających działanie aplikacji.
- Łatwiejsza lokalizacja błędów poszczególnych warstw

N-Tier

- Zwiększenie obciażenia poprzez dodatkową komunikację pomiędzy warstwami

- Skupienie się na realizacji rozwiązań stosowanych w aplikacji
- Łatwość przenoszenia aplikacji do na inne serwery aplikacji

N-Tier - zadania

- Organizacja i rozdzielanie żądań
- Autoryzacja użytkowników
- Obsługa wielozadaniowości
- Kierowanie zadań do innych maszyn w przypadku awarii
- Monitorowanie działania aplikacji

N-Tier - aplikacja

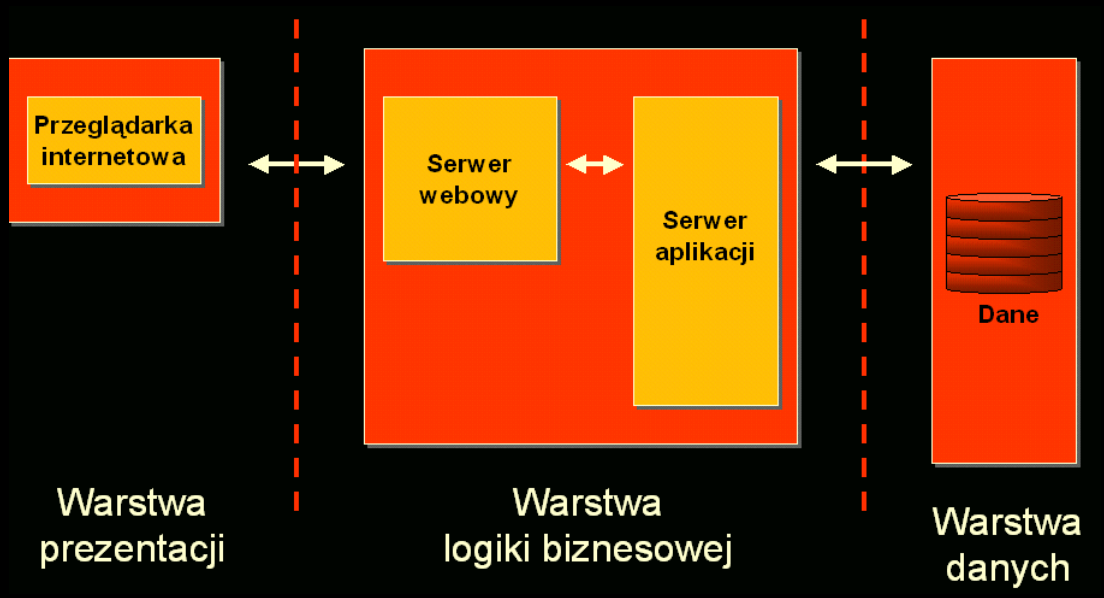
Jak stworzyć aplikację?

- niezawodną
- bezpieczną
- wydajną

Należy dokonać logicznego podziału aplikacji na warstwy tak, aby każda z nich realizowała własne zadanie.

Podział na warstwy

- warstwa prezentacji
- warstwa logiki biznesowej
- warstwa danych



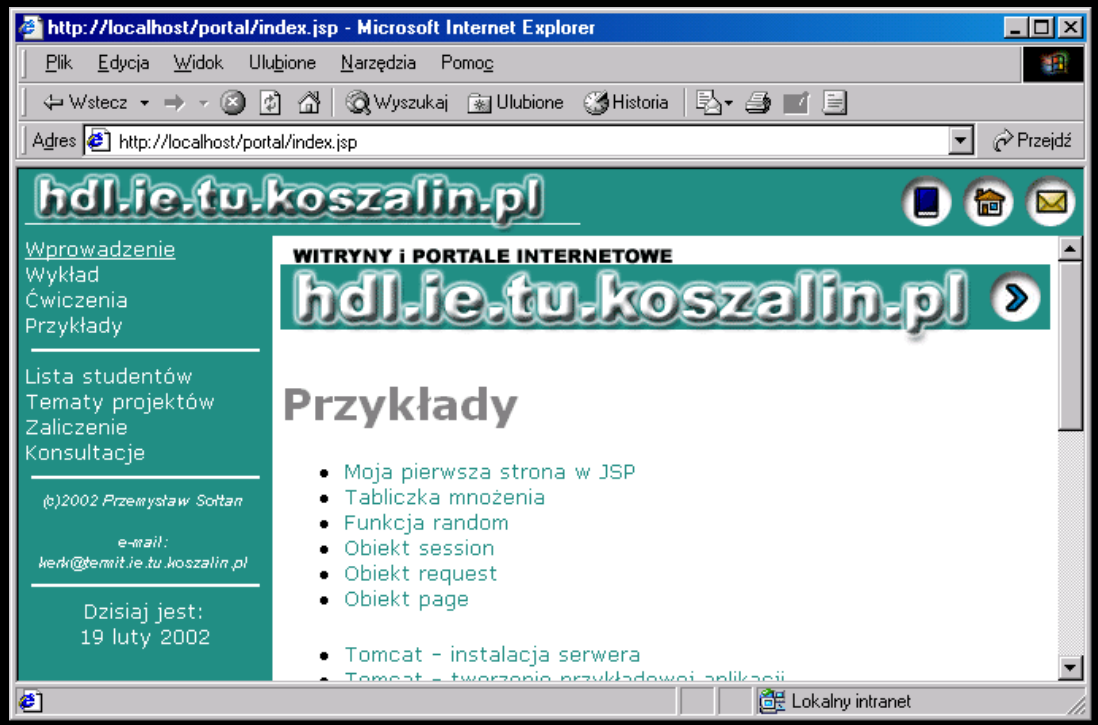
Podział na warstwy

Izolacja:

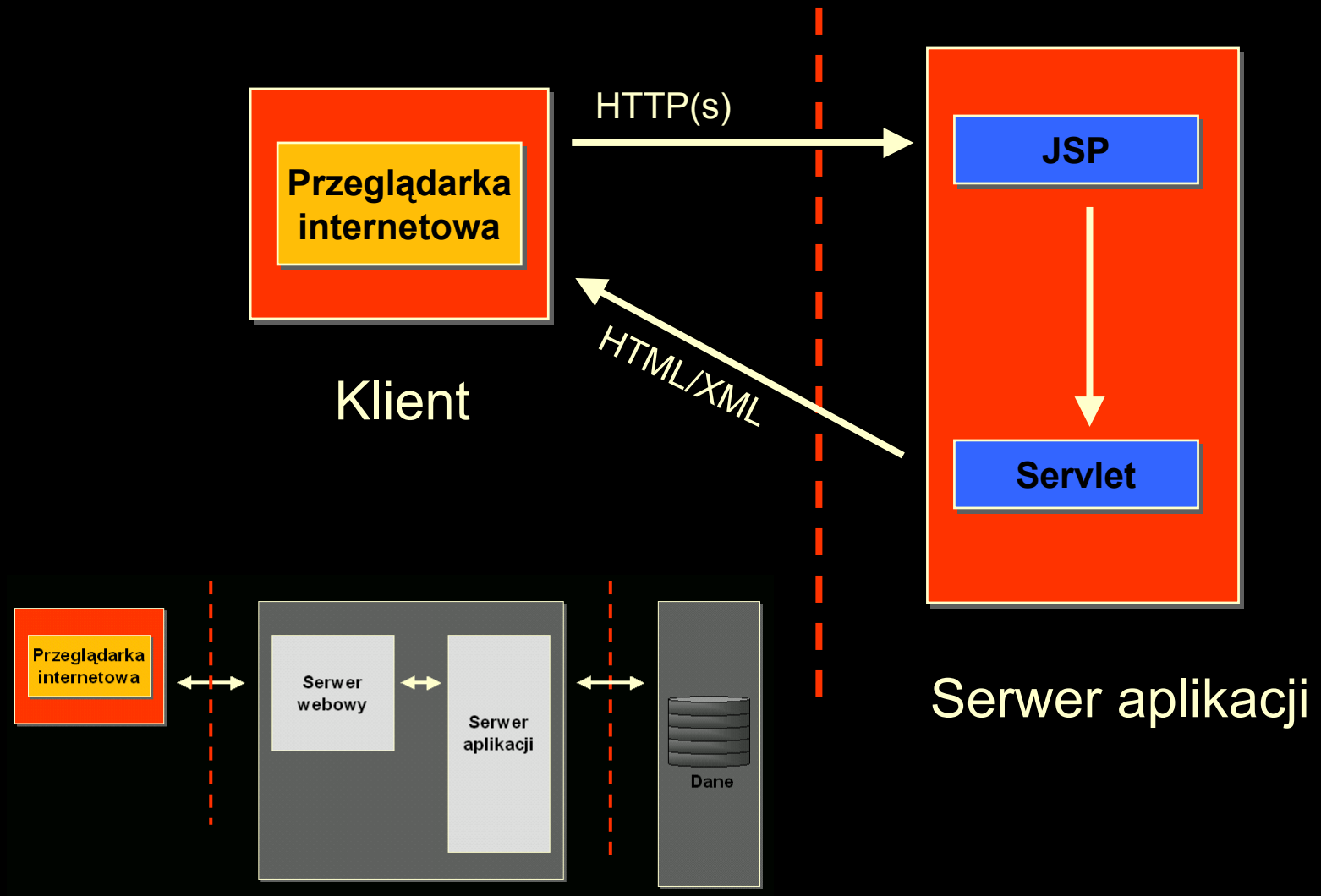
- podział aplikacji na warstwy w ten sposób, aby zmiany w jednej z nich miały jak najmniejszy wpływ na drugą
- podniesienie efektywności modyfikacji aplikacji (np. zastosowanie innego mechanizmu baz danych, czy też zmiana sposobu prezentacji danych u klienta)

Warstwa prezentacji

- Komponenty odpowiedzialne za wygląd aplikacji po stronie klienta
  - strony JSP
  - serwlety
  - aplety Java
  - aplikacje



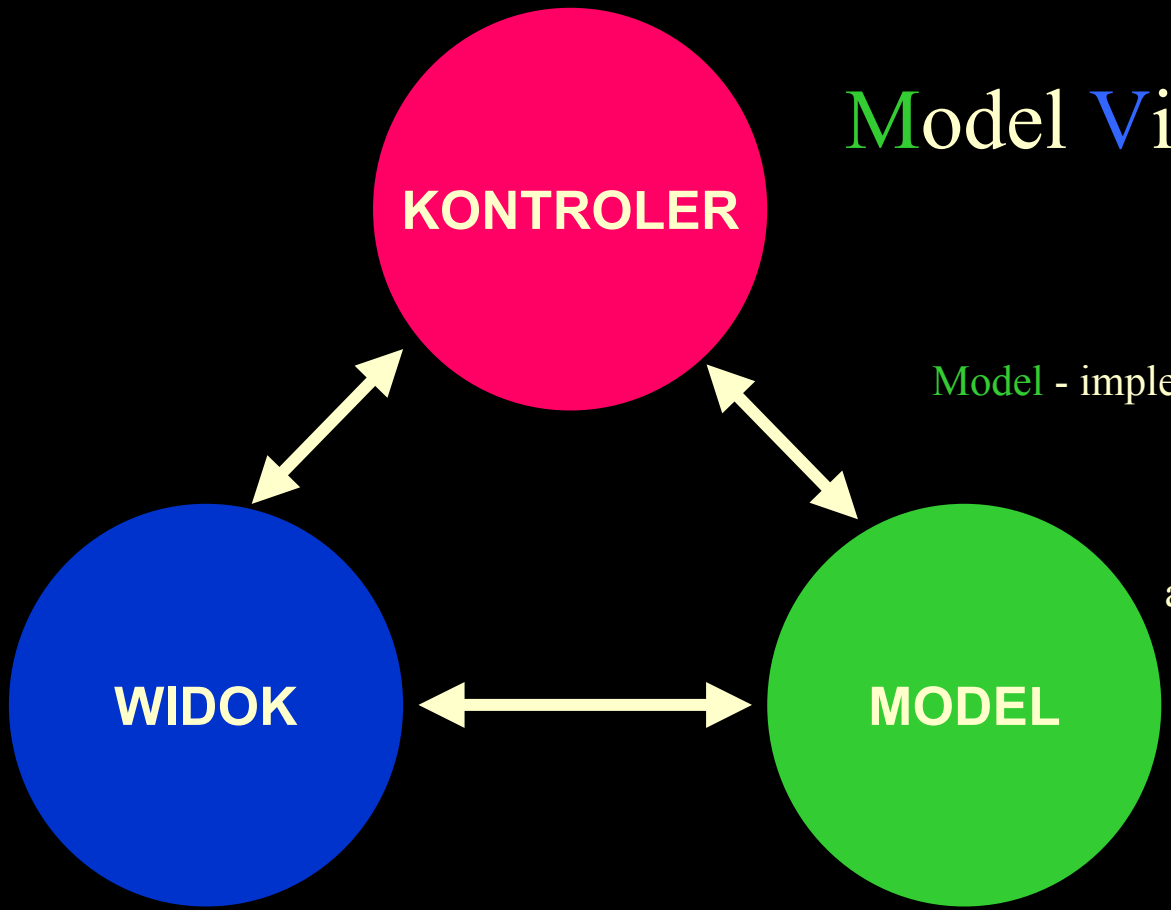
Warstwa prezentacji



Oddzielenie logiki biznesowej od interfejsu użytkownika

- Uproszczenie zarządzania projektem (MVC)
- Możliwość tworzenia wielu widoków dla określonej logiki biznesowej
- Ułatwienie procesu ponownego wykorzystania kodu (komponentowość)

MVC



Model View Controller

Model - implemenacja logiki aplikacji i zarządzanie dostępem do danych

View - prezentacja wyników aplikacji i interakcja z użytkownikiem

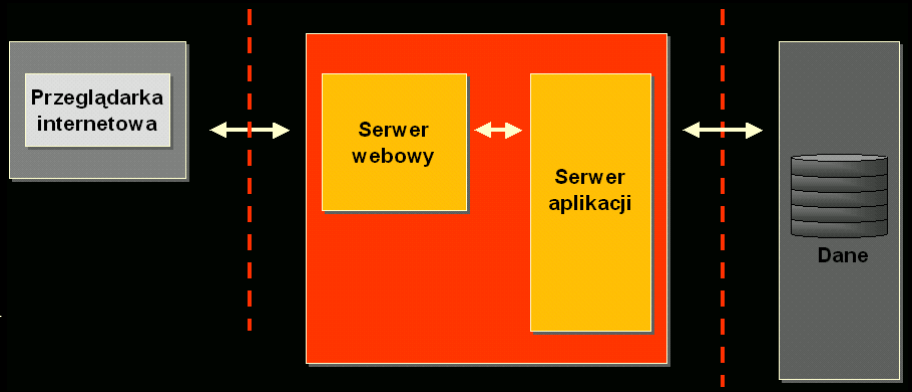
Controller - rozpoznawanie żądań i kontrola nawigacji

DESIGN PATTERN - WZORCE PROJEKTOWE

Warstwa logiki biznesowej

Logika biznesowa jest odpowiedzialna za wykonywanie głównych zadań aplikacji:

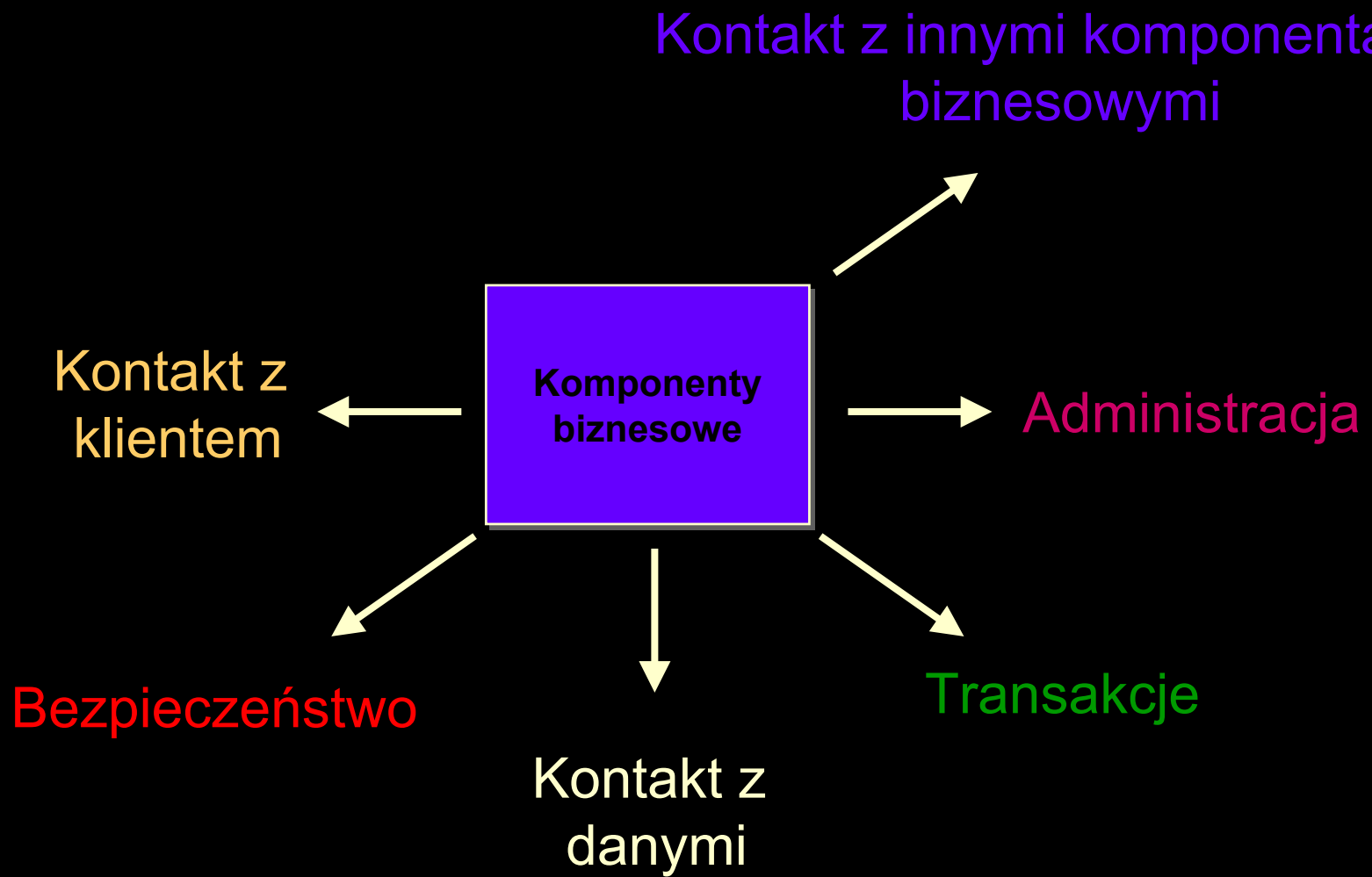
- dostępu do źródeł danych
- integracji systemu z nowymi funkcjami



Odwzorowanie metod za pomocą komponentów biznesowych: session bean i wielu entity beans.

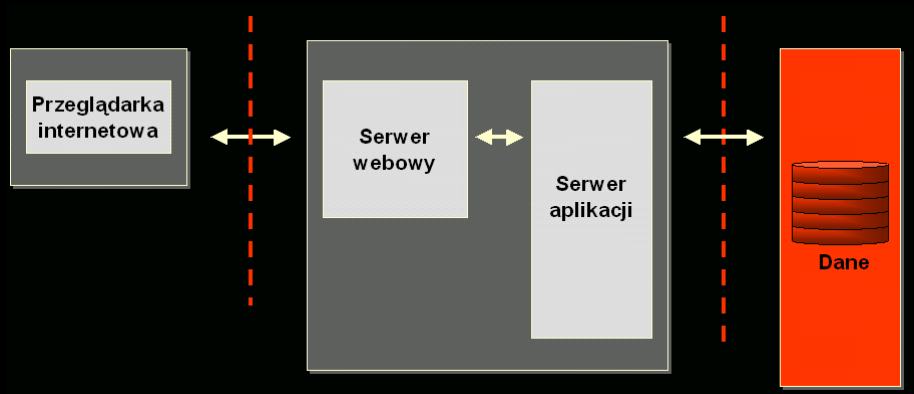
Warstwa obsługiwana jest przez serwer aplikacji zgodny z architekturą J2EE

Komponenty biznesowe



Warstwa danych

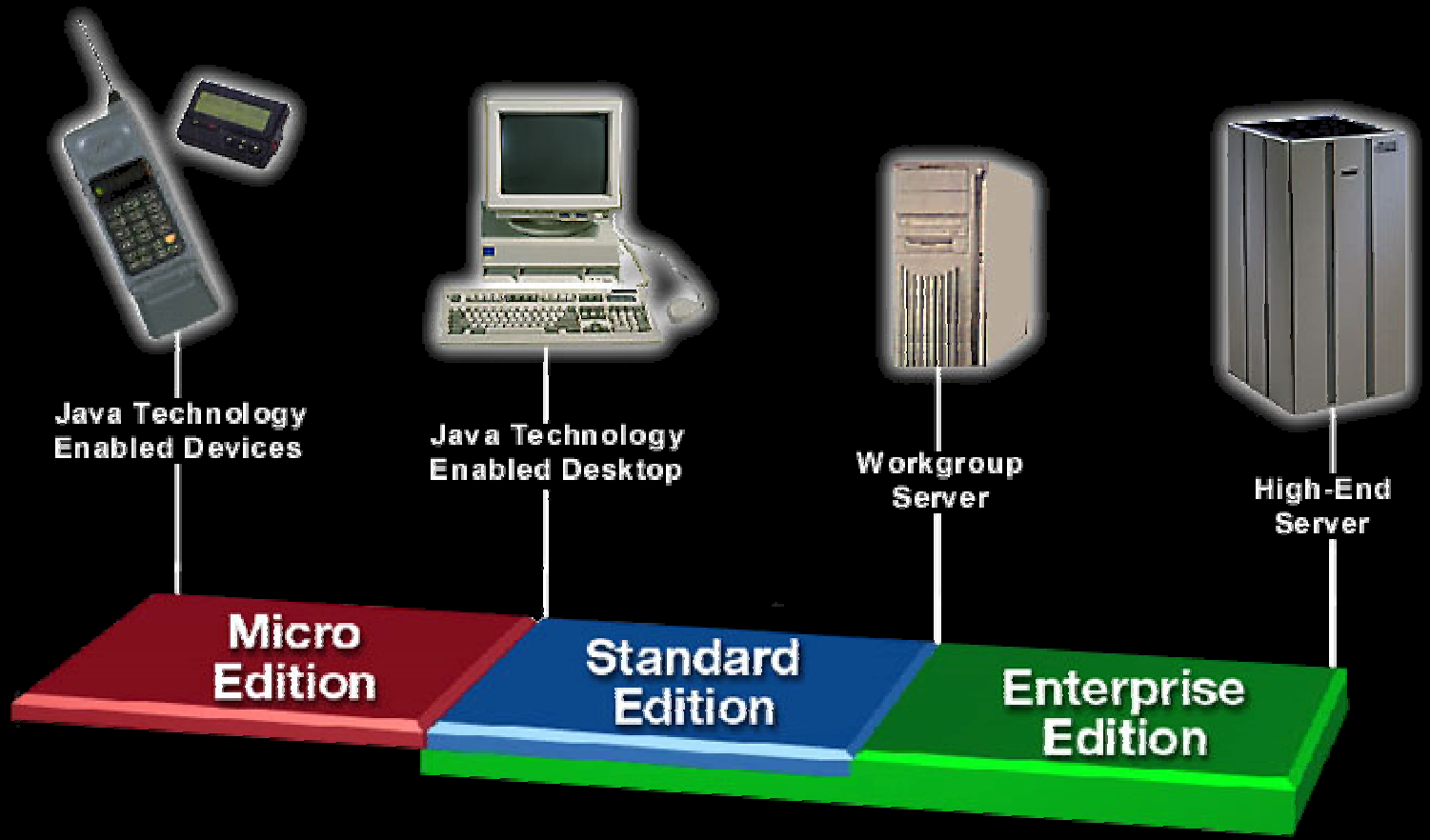
Warstwa danych jest źródłem danych dla aplikacji:



- relacyjna baza danych
- system plików
- system katalogów
- obiektowa baza danych.

Platforma JAVA

Platforma JAVA



Platforma JAVA

- Java 2, Standard Edition (J2SE) - definiuje platformę dla aplikacji ogólnych zastosowań. Platforma taka musi implementować podstawowy zakres API. Przeglądarki internetowe to przykład platformy J2SE.

Platforma JAVA

- Personal Java - podzbiór J2SE, przeznaczona do urządzeń przenośnych z ograniczonymi zasobami.
- Java Card - minimalny podzbiór klas Java do tworzenia aplikacji z wykorzystaniem kart inteligentnych (smart card).

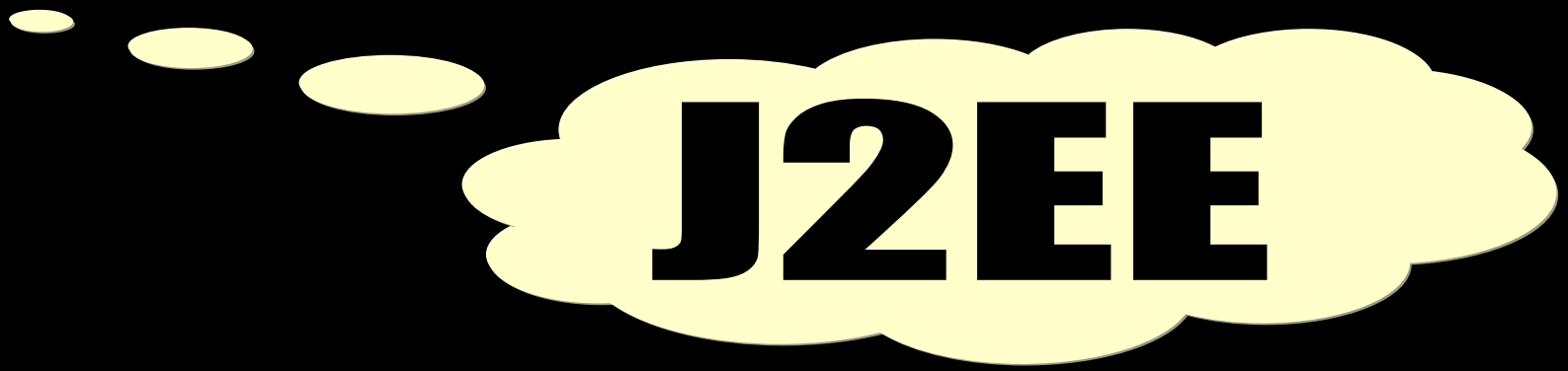
Platforma JAVA

- Java 2, Enterprise Edition - rozszerza J2SE o dodatkowe API pozwalające na tworzenie aplikacji wielowarstwowych klasy enterprise.

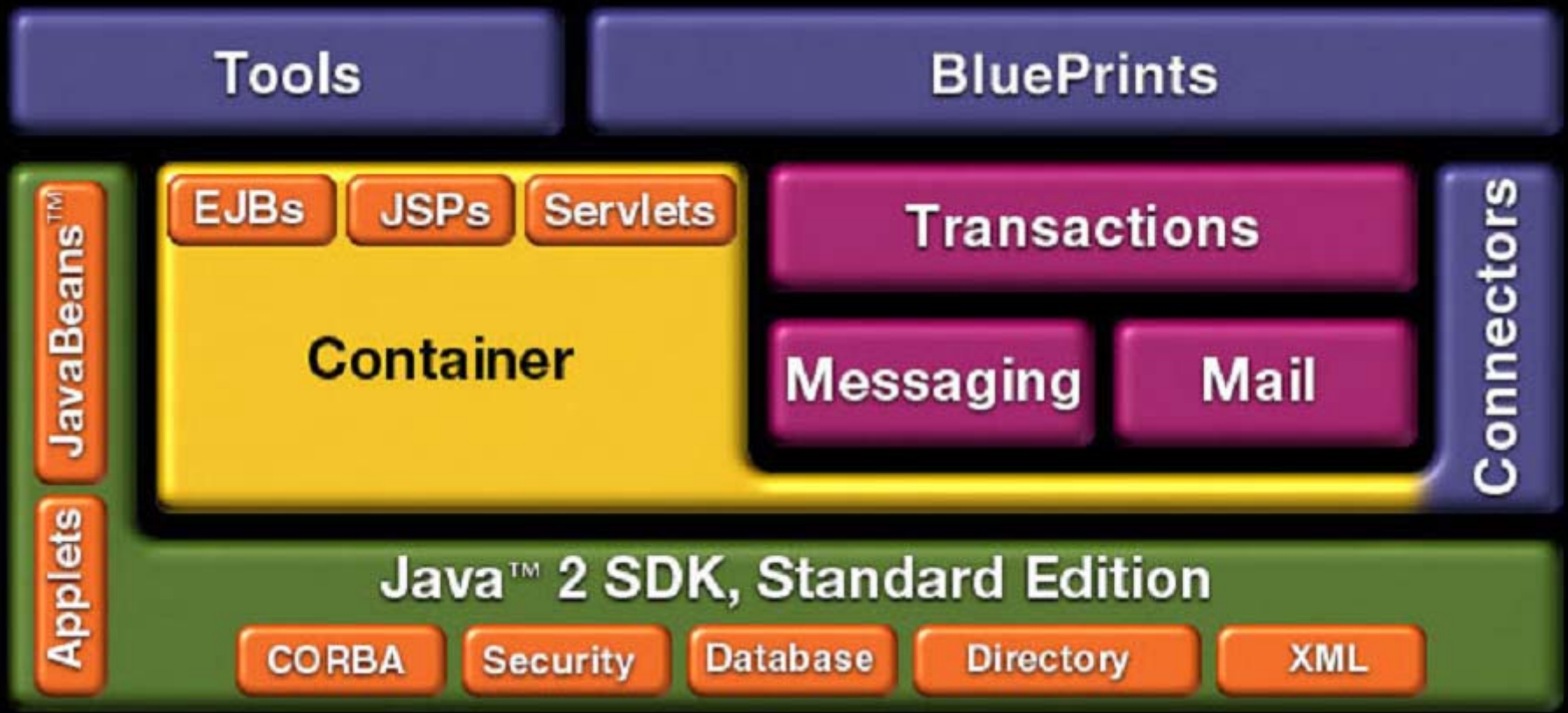
Technologie J2EE

Technologie J2EE

- Kontenery i technologie komponentowe
- Technologie komunikacyjne
- Usługi



Architektura J2EE



J2EE BluePrints - zbiór wskazówek, preferowanych rozwiązań, wzorców projektowych

J2EE jako rozszerzenie J2SE

J2SE
- Collections framework
- Internationalization
- I/O
- Java Archive Files (JAR files)
- JavaBeans
- Java DataBase Connectivity (JDBC)
- Java Foundation Classes (JFC)
- Java Interface Defn. Lang. (JavaIDL)
- Java Naming & Dir. Interface (JNDI)
- Java Native Interface (JNI)
- Language and utility packages
- Math
- Networking
- Object serialization
- Package version identification
- Reflection
- Reference objects
- Remote Method Invocation (RMI & RMI-IIOP)
- Resources
- Security and signed applets
- Sound

J2EE
- Enterprise Java Beans (EJB)
- JavaMail API
- Java Message Service (JMS)
- JavaServer Pages (JSP)
- Java Servlets
- Java Transaction API (JTA)
- Java Transaction Service (JTS)
- JavaBeans Activation Framework (JAF)
- J2EE Connector
- Java DataBase Connectivity (JDBC) extension

- Java Interface Definition Lang. (Java IDL)
- Java Naming & Directory Interface (JNDI)
- Remote Method Invocation (RMI-IIOP)

Enterprise API

J2SE i J2EE

Java2 Standard Edition (J2SE)

- JavaBean
- Java DataBase Connectivity (JDBC)
- Java Naming and Directory Interface (JNDI)
- Remote Method Invocation (RMI)

Java2 Enterprise Edition (J2EE)

- Enterprise Java Bean (EJB)
- Java Server Pages (JSP)
- Java Servlets (Servlets)

Enterprise API

Servlety

- tworzenie klas javy generujących kod HTML (java -> class)

Java Server Pages (JSP)

- dynamiczne tworzenie stron HTML z możliwością bezpośredniego umieszczania kodu java w plikach html (jsp -> java -> class)

Enterprise API

Enterprise Java Beans (EJB)

- komponenty działające w warstwie logiki biznesowej, zarządzane i wykonywane na serwerze w kontenerze EJB

Java Naming and Directory Interface (JNDI)

- usługa katalogowania pełniąca rolę zapamiętywania informacji o położeniu zasobów, scalania i zarządzania aplikacji J2EE

Enterprise API

Java Remote Method Invocation (RMI)

- metody obsługi serwisów sieciowych oraz obsługa ze zdalnych maszyn

JavaIDL

- obsługa połączeń i współpracy heterogenicznych obiektów (implementacja CORBY w języku java)

Enterprise API

Java Data Base Connectivity (JDBC)

- dostęp do relacyjnych systemów przechowywania danych (SQL)

Java Messaging Service (JMS)

- usługa asynchronicznej wymiany komunikatów (mechanizm kolejkowanie komunikatów)

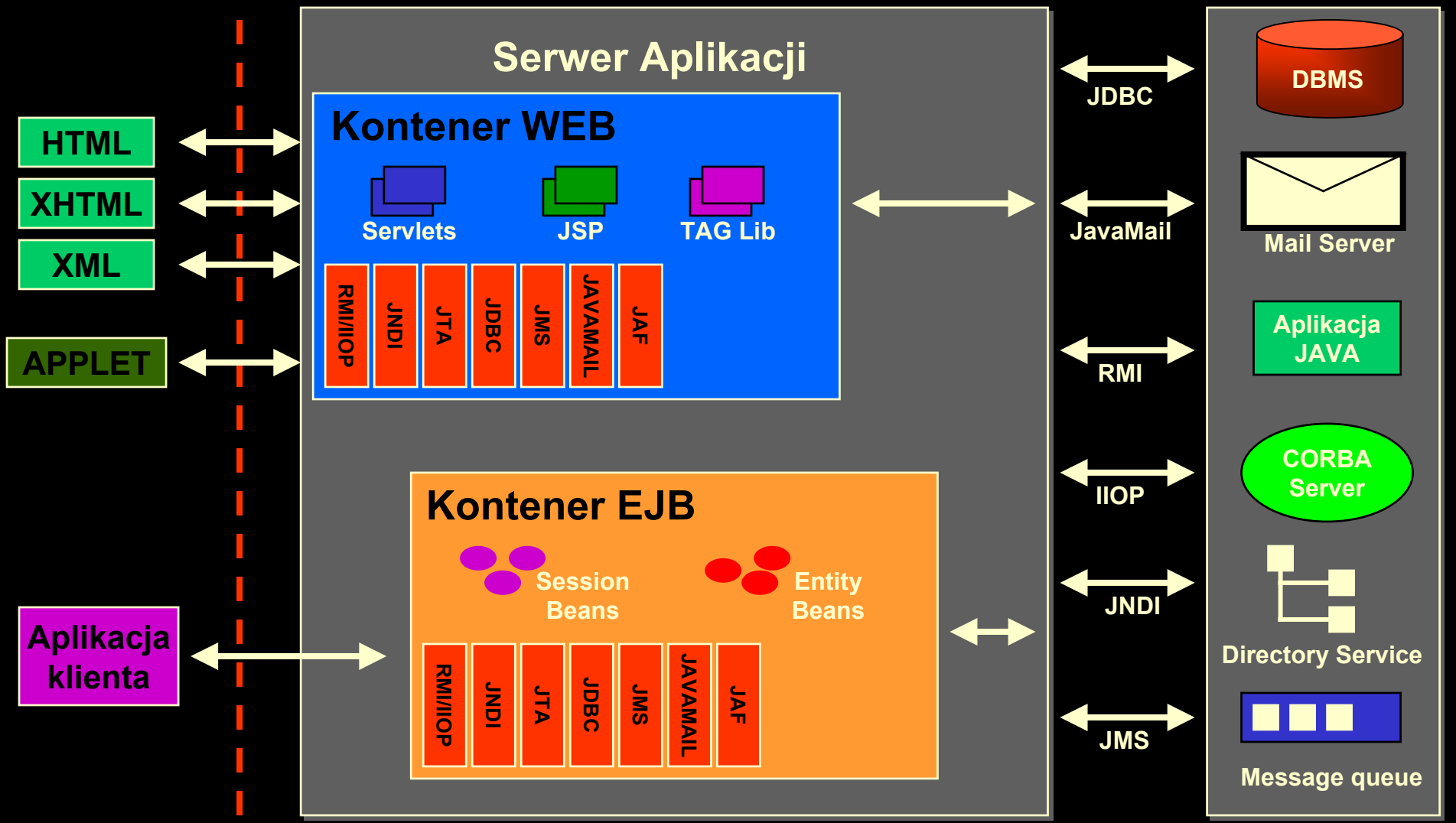
Enterprise API

Java Transaction API (JTA) i Java Transaction Service (JTS)

- interfejsy wspierające realizację transakcji
  - menadżer zasobów (system informacyjny)
  - menadżer transakcji (serwer)

Transakcja - jednostka logiczna pracy, gwarantująca poprawne wykonanie danej czynności

Platforma J2EE



Kontenery

Kontenery

Kontener - środowisko pracy dla zarządzanych przez siebie komponentów

Oddzielenie komponentów od pozostałych elementów systemu:

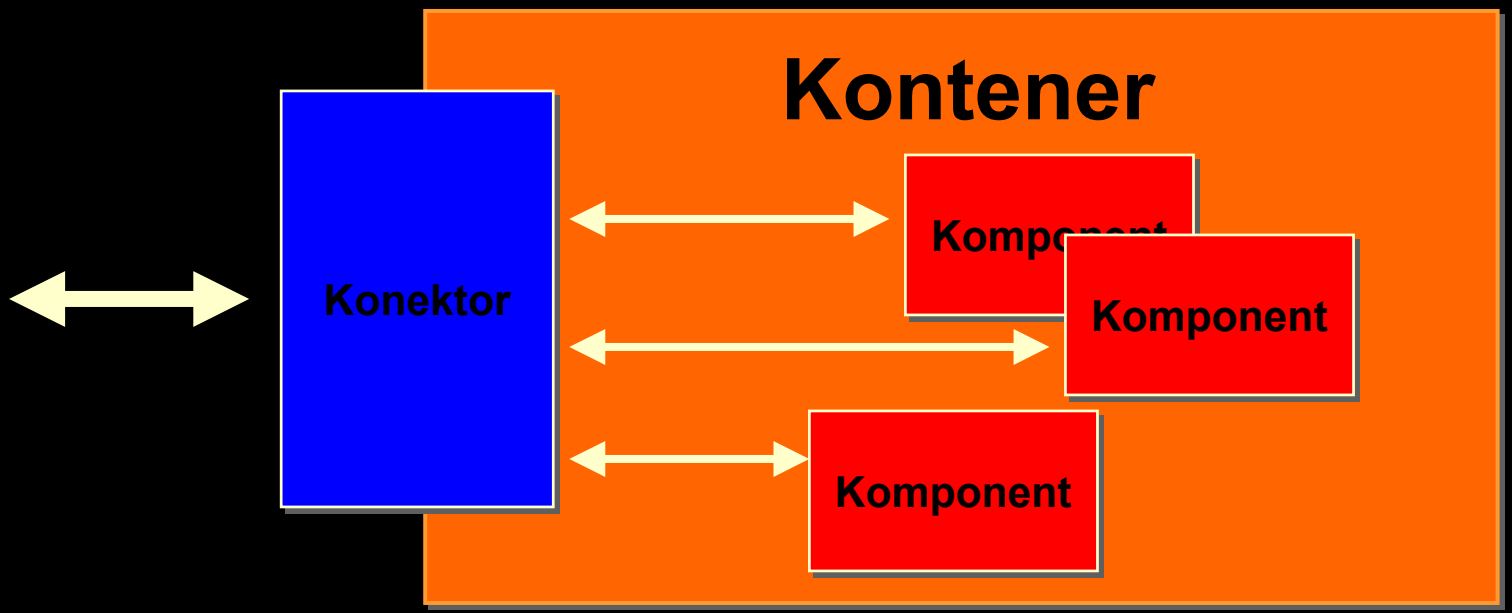
- separacja klienta od wpływu na komponent (hermetyzacja)
- separacja od zewnętrznych zasobów

Serwer aplikacji (kontener)

Serwer aplikacji:

- udostępnienie środowiska pracy dla komponentów
- implementacja szeregu interfejsów programowych
- wykonywanie określonych zadań (zarządzanie dostępem, wykorzystaniem zasobów, zarządzanie transakcjami i bezpieczeństwem)

Komponenty, kontenery i konektory

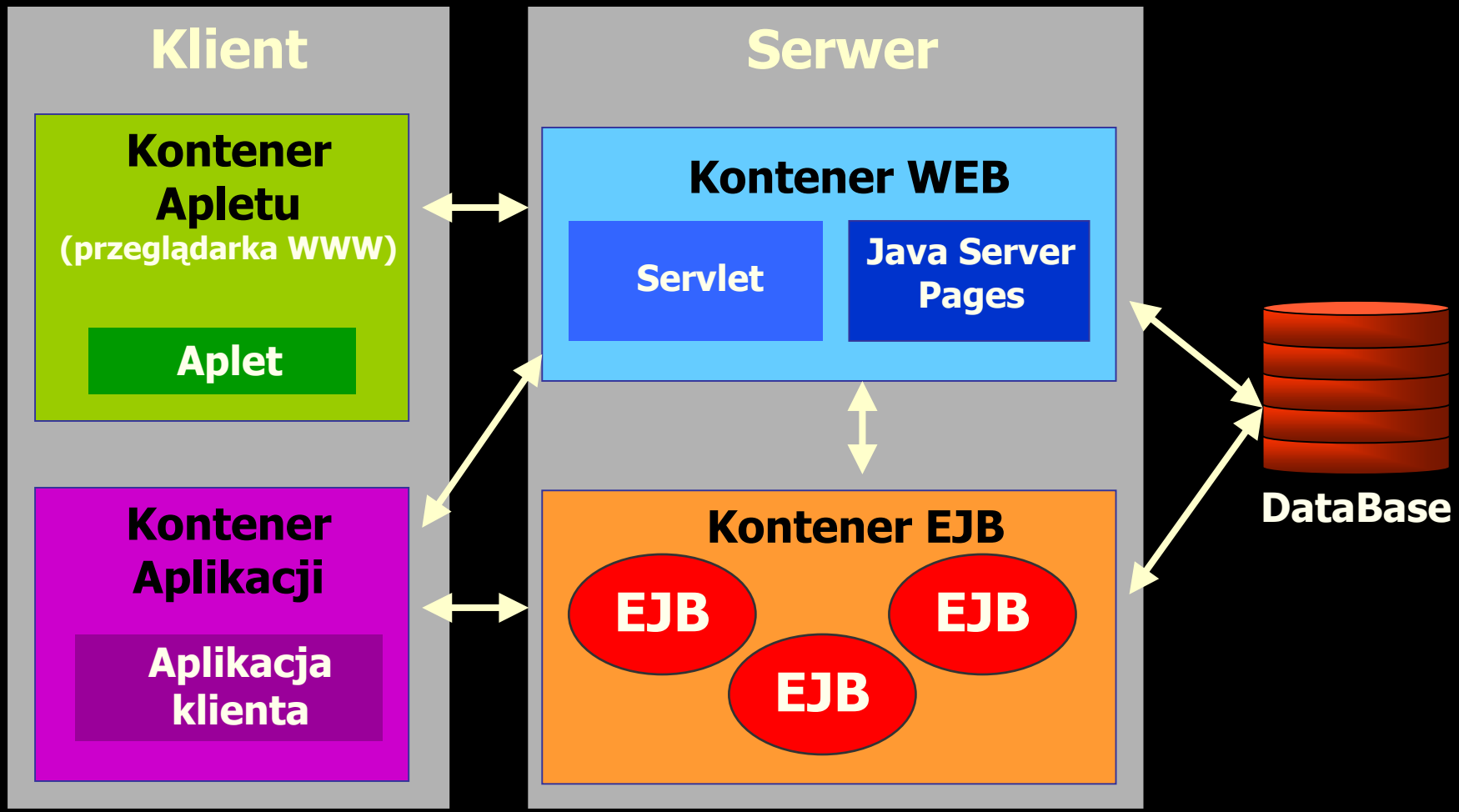


Konektor - interfejs pośredniczący umożliwiający komponentom kontakt ze środowiskiem zewnętrznym

Kontenery (Container)

- kontener apletu (przeglądarka WWW)
- kontener aplikacji klienta
- kontener WEB
- kontener Enterprise Java Bean (EJB)

Kontenery


Klient
Kontener Apletu
(przeglądarka WWW)
Aplet
Kontener Aplikacji
Aplikacja klienta
Serwer
Kontener WEB
Servlet
Java Server Pages
Kontener EJB
EJB
EJB
EJB
DataBase

J2EE API

Technology	Acronym	Application Client Container	Web Container	EJB Container
Enterprise Java Beans	EJB	✍*	✍*	✍
JavaMail API			✍	✍
Java Message Service	JMS	✍	✍	✍
JavaServer Pages	JSP		✍	
Java Servlets	Servlet		✍	
Java Transaction API	JTA		✍	✍
JavaBeans Activation Framework	JAF		✍	✍
J2EE Connector		✍	✍	✍
Java DataBase Connectivity	JDBC	✍	✍	✍
Java Interface Defn. Lang.	JavaIDL	✍	✍	✍
Java Naming & Dir. Interface	JNDI	✍	✍	✍
Remote Method Invocation	RMI-IIOP	✍	✍	✍

* Client APIs only

HTTP

HTTP
Hypertext Transfer Protocol

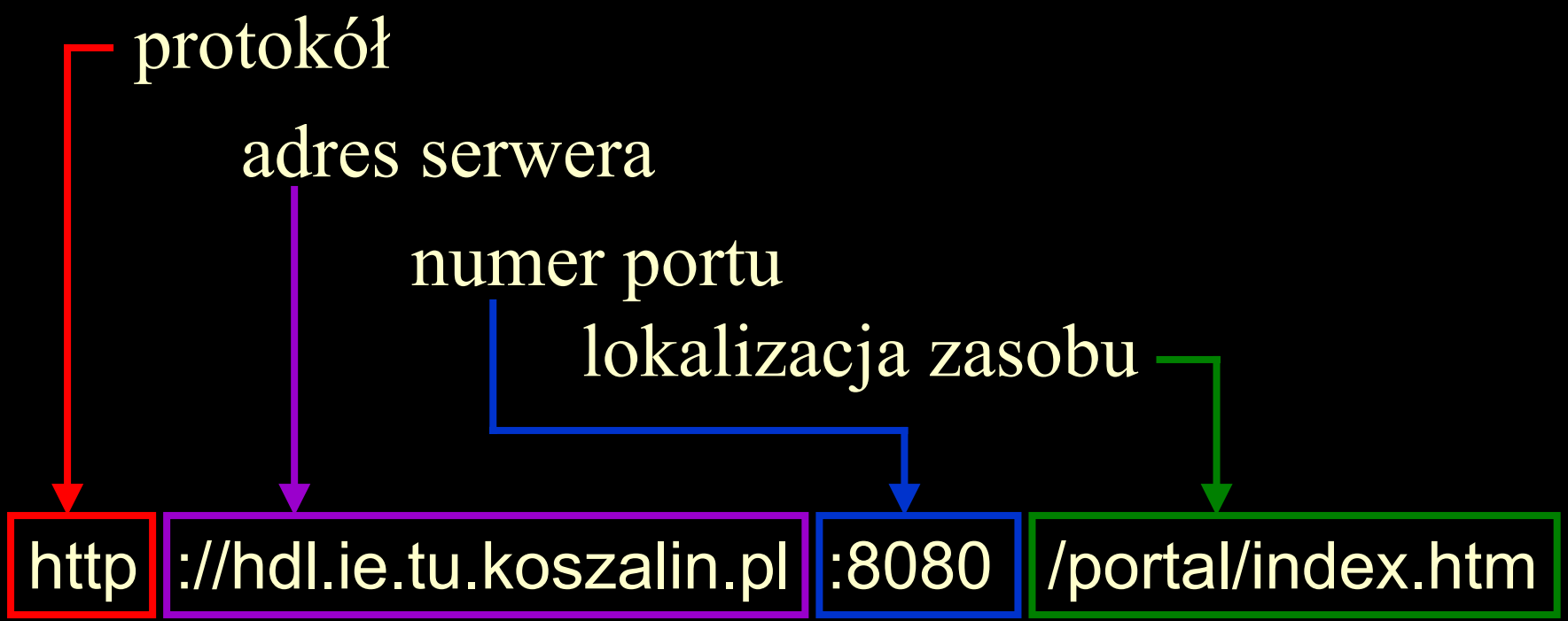
- GET
- POST
- HEAD
- PUT
- DELETE
- TRACE
- CONNECT

(dokumentacja RFC 2068)

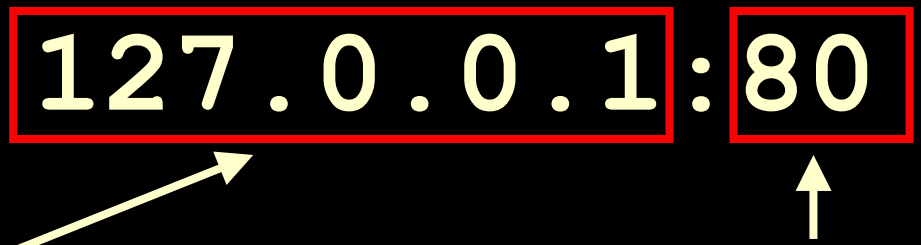
(dokumentacja RFC 2616) http://www.rfc-editor.org

URL - Uniform Resource Locator

Elementy adresu URL:



TCP/IP
Adres IP i numer portu



Adres IP (4 bajty)

0-127.x.x.x - klasa A

128-191.x.x.x - klasa B

192-223.x.x.x - klasa C

127.0.0.1 - localhost

(dokumentacja RFC 922)

Numer portu (2 bajty) -
(określenie rodzaju usługi)

119 - NNTP

23 - TELNET

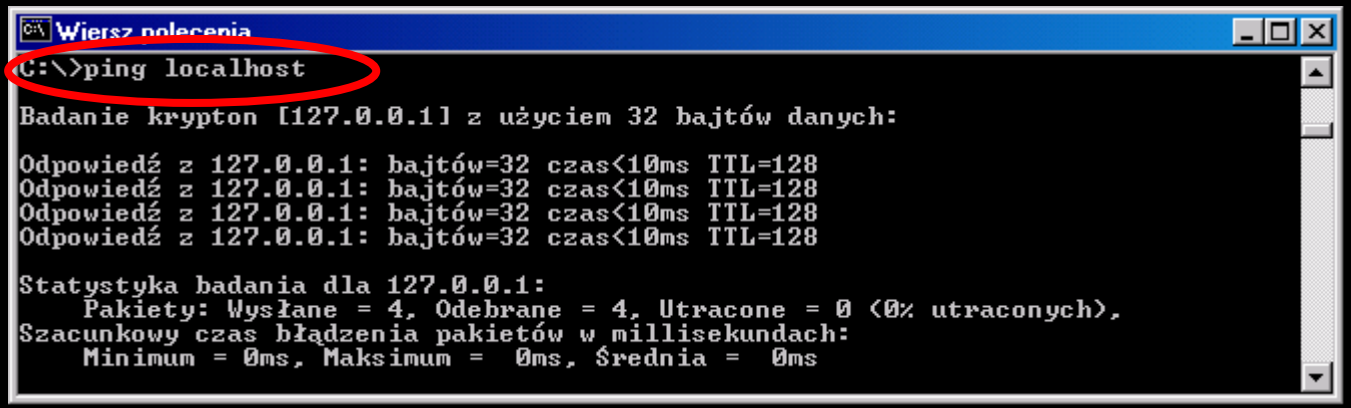
20 - FTP

25 - SMTP

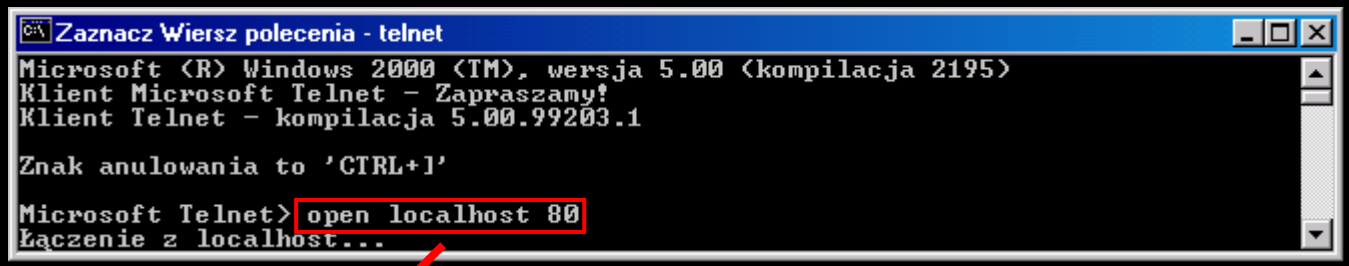
53 - DNS

80 - HTTP

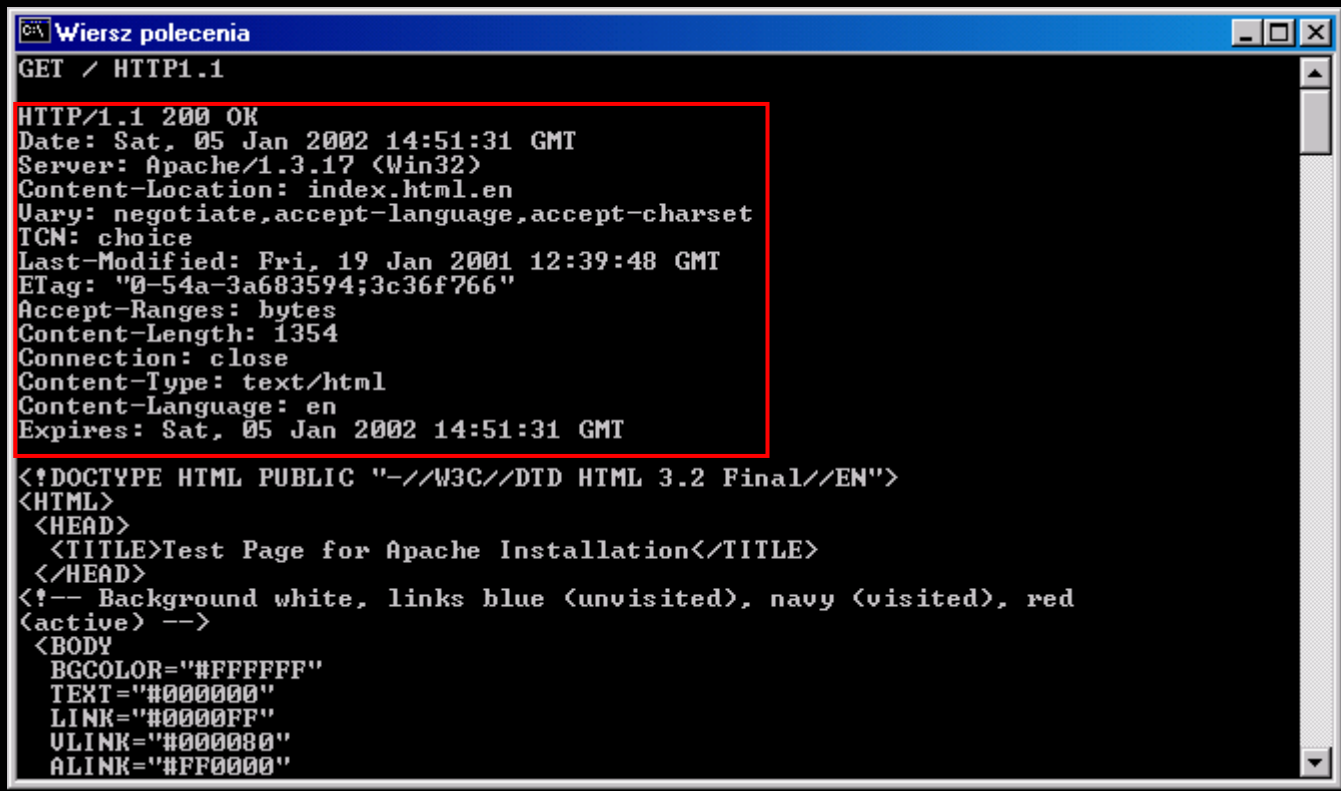
Klient - ping


Wiersz polecenia
C:\>ping localhost
Badanie krypton [127.0.0.1] z użyciem 32 bajtów danych:
Odpowiedź z 127.0.0.1: bajtów=32 czas<10ms TTL=128
Odpowiedź z 127.0.0.1: bajtów=32 czas<10ms TTL=128
Odpowiedź z 127.0.0.1: bajtów=32 czas<10ms TTL=128
Odpowiedź z 127.0.0.1: bajtów=32 czas<10ms TTL=128
Statystyka badania dla 127.0.0.1:
Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0 (0% utraconych),
Szacunkowy czas błądzenia pakietów w millisekundach:
Minimum = 0ms, Maksimum = 0ms, Średnia = 0ms

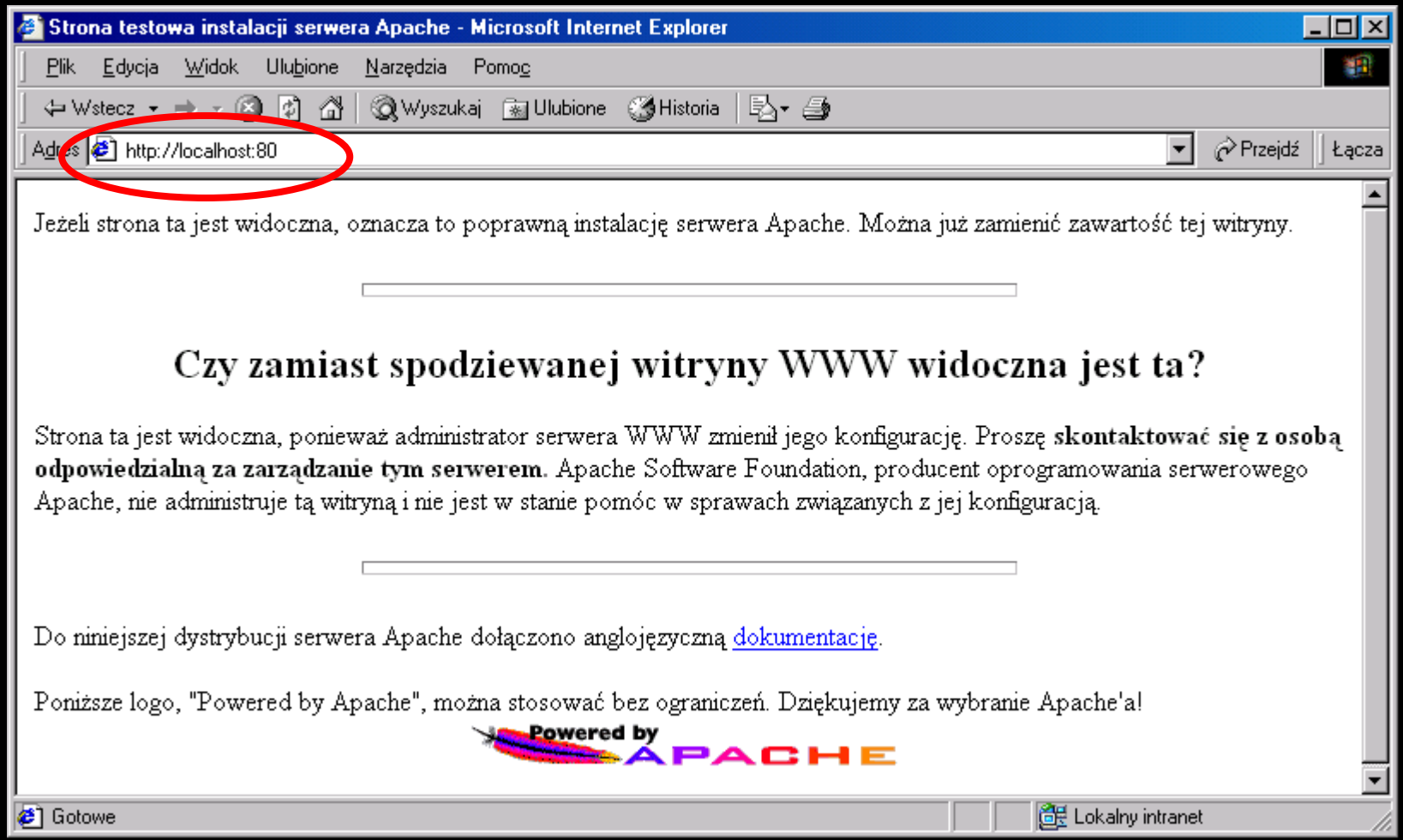
Klient - telnet



GET / HTTP1.1



Klient - przeglądarka WWW



HTTPS

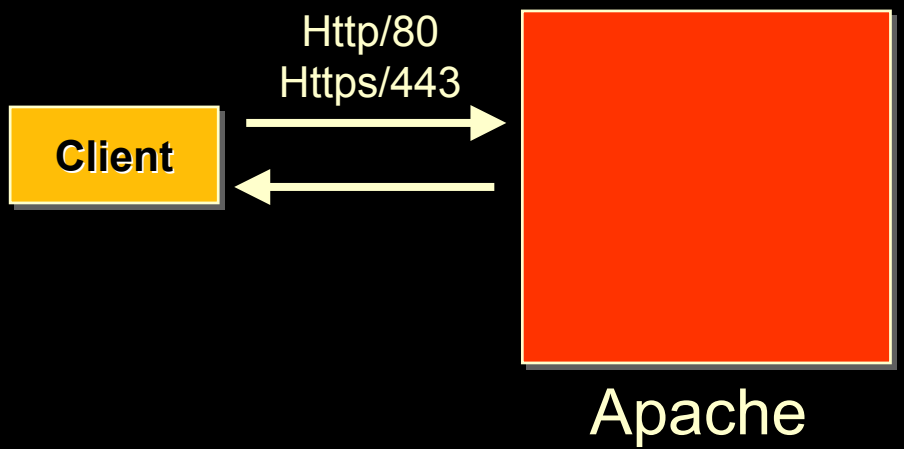
Protokół HTTPS wykorzystuje się do bezpiecznej komunikacji HTTP

- protokół HTTP za pomocą Secure Sockets Layer (SSL)
- komunikacja z wykorzystaniem kryptografii (szyfrowania danych)
- komunikacja poprzez domyślny port 443

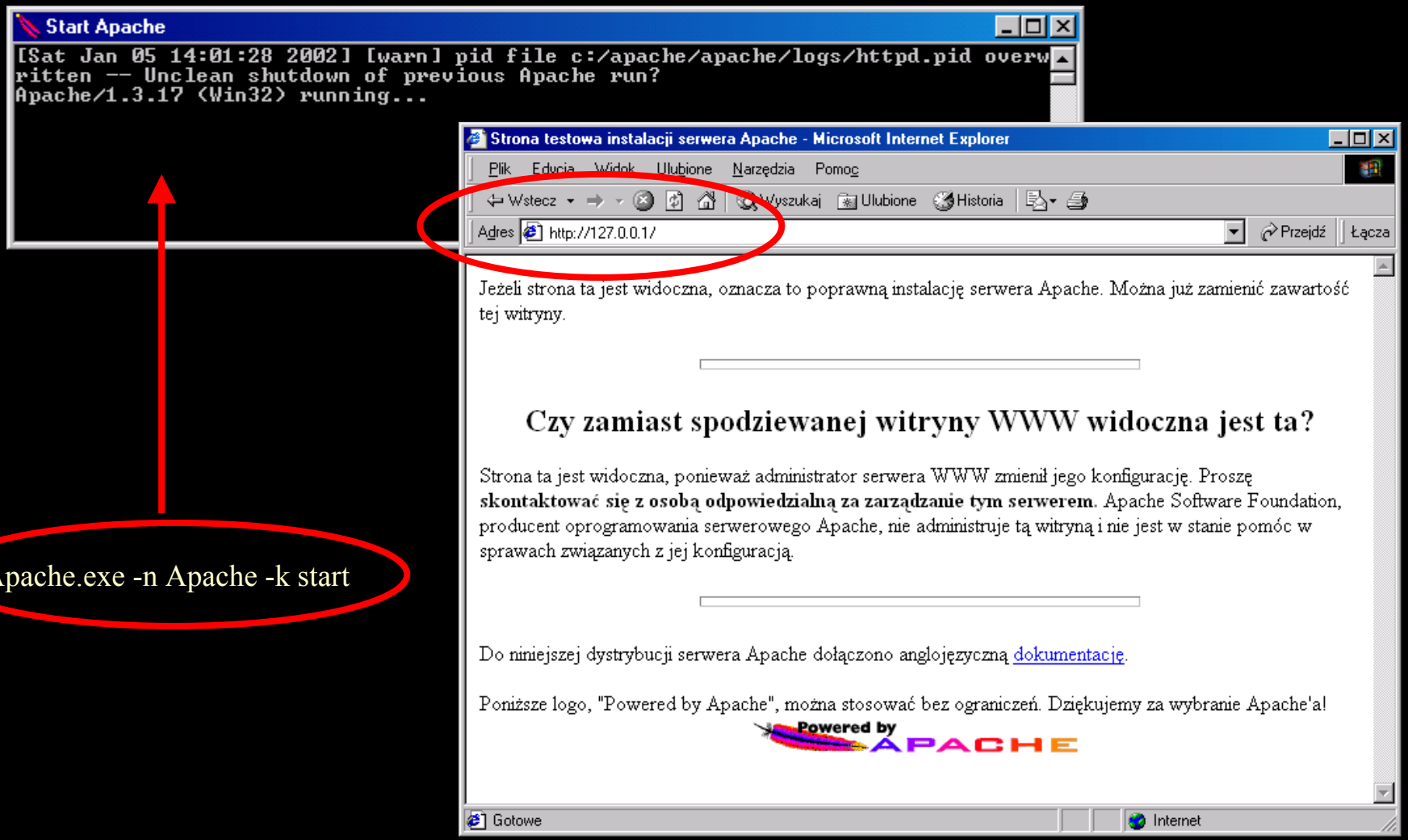
Serwery www

Serwer Apache

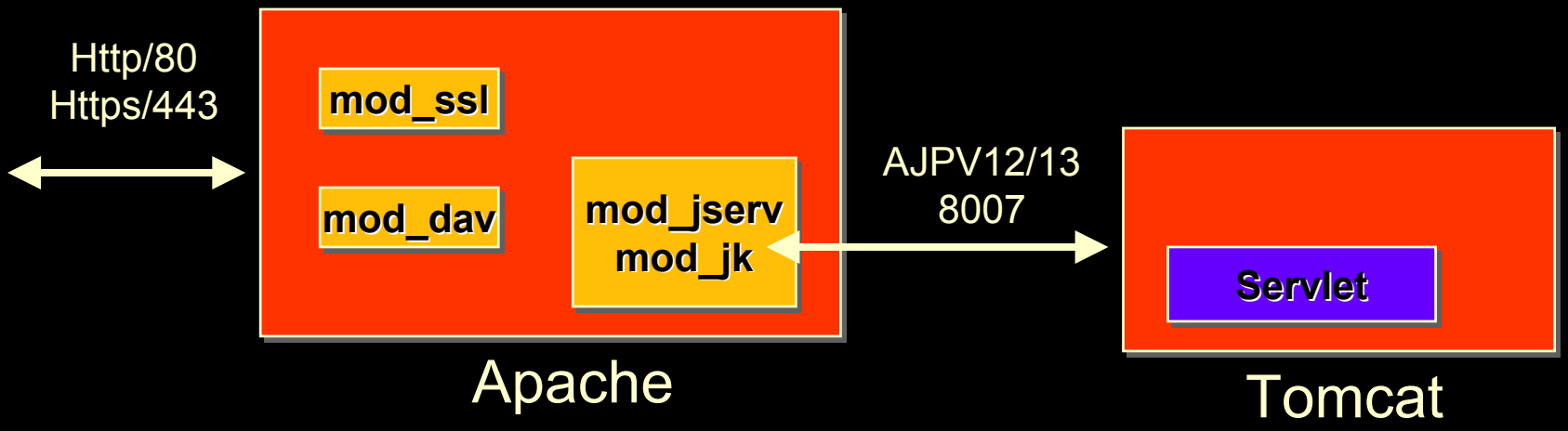
- Serwer Apache



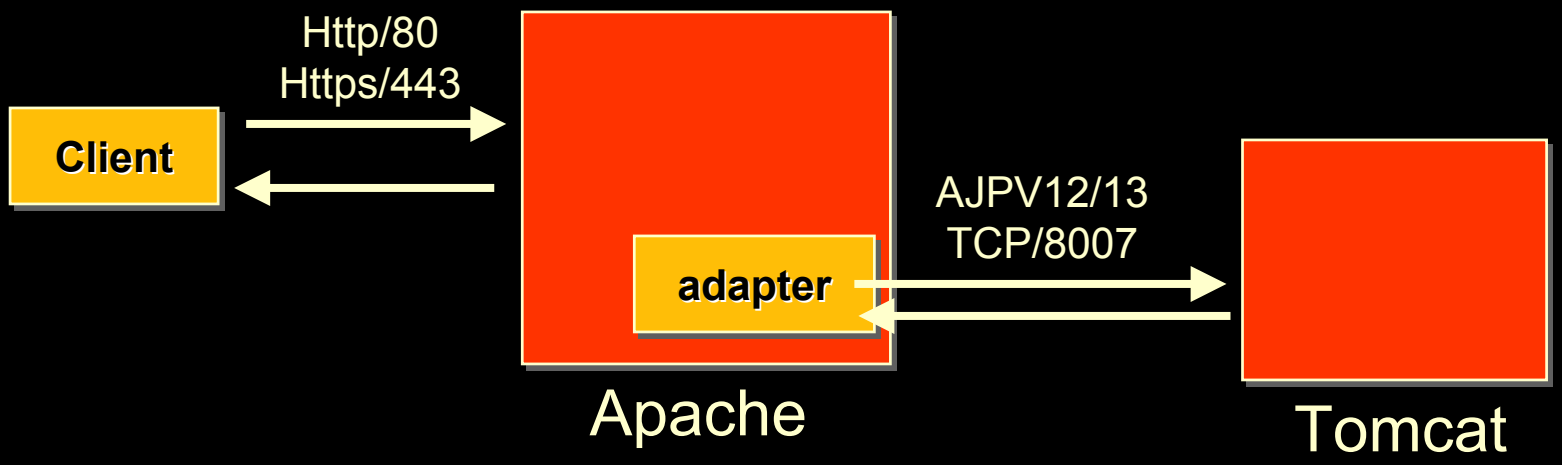
Serwer Apache



Apache+Tomcat



Apache+Tomcat



Serwery aplikacji

Java

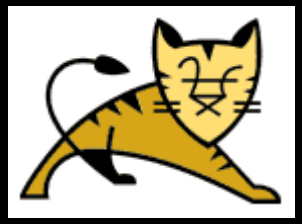
- po stronie klienta
  - słaba wydajność (aplikacje java i aplety)
  - różne wersje JVM klientów
- po stronie serwera
  - kontrola środowiska JVM
  - szybkość języka nie odgrywa znaczącej roli (straty czasu na poziomie baz danych i połączeń sieciowych)

Serwery Aplikacji

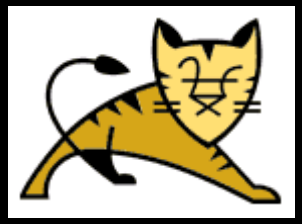
- SUN (J2EE server)
- Apache Jakarta (Tomcat)
- Caucho (Resin)
- JBOSS
- Lutris (Enhydra)

Komercyjne serwery aplikacji

- IBM (Websphere Application Server)
- BEA (Weblogic Application Server)
- Borland (Borland Application Server)
- Oracle (Oracle 9i Application Server)
- Macromedia (JRun Application Server)
- IONA (iPlanet Application Serwer)
- Pramati (Pramati Server)
- HP (HP Bluestone Total-e-Server)

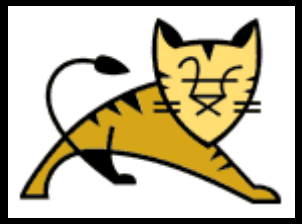


Serwer TOMCAT

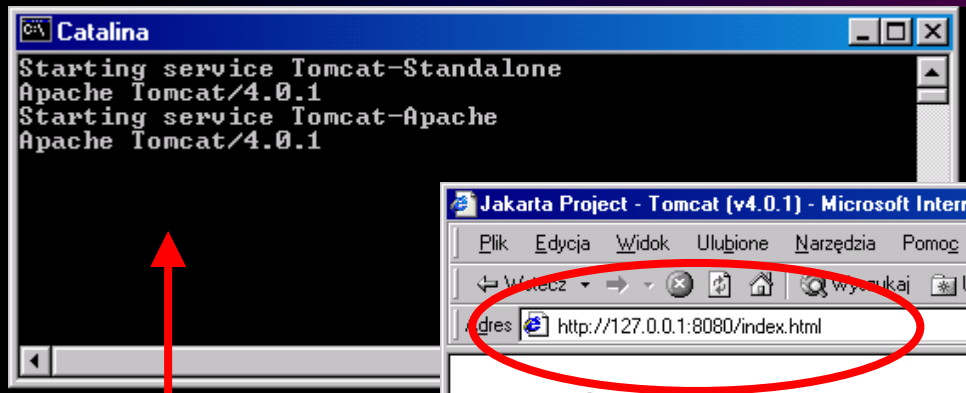


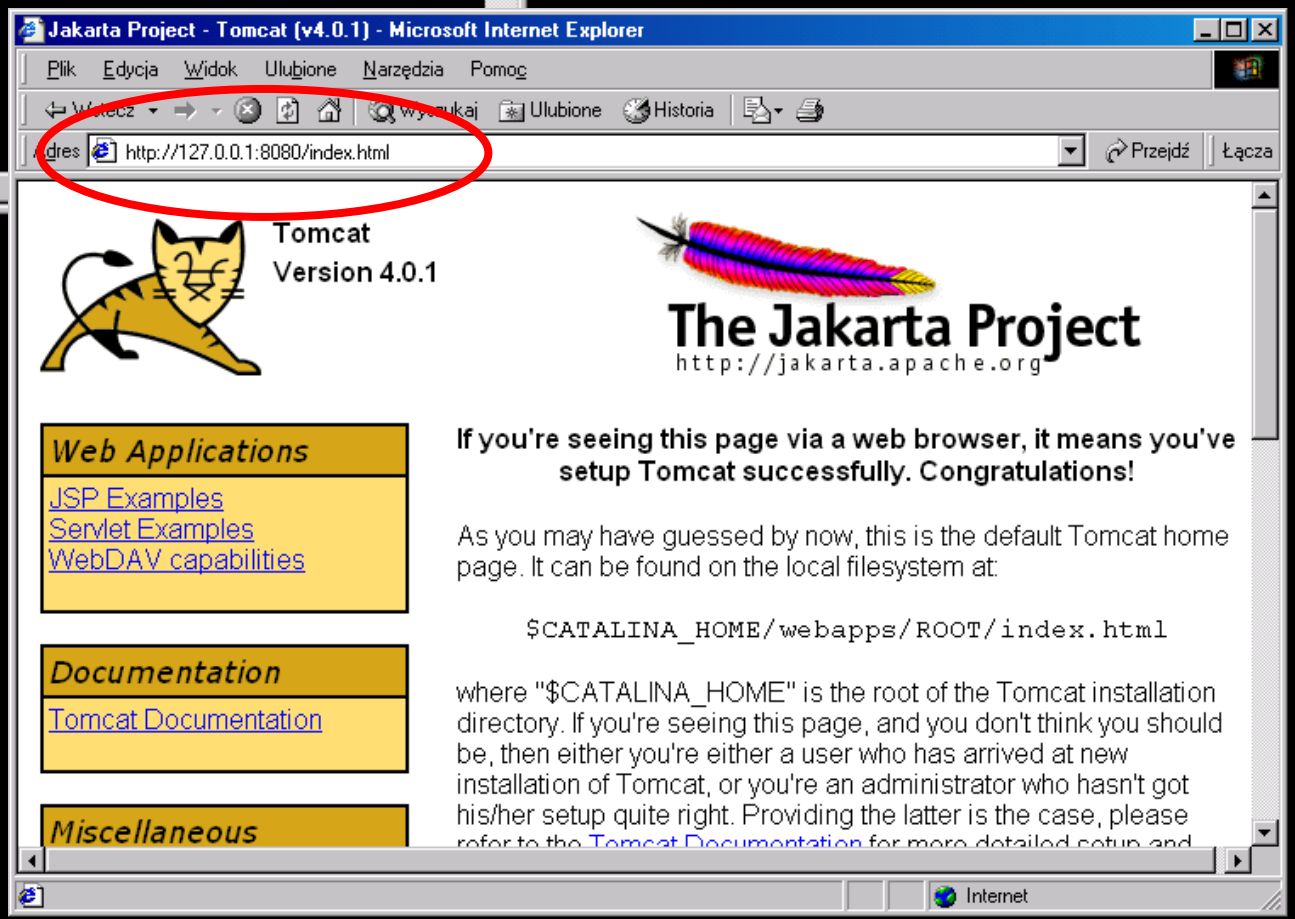
Instalacja i uruchomienie

- instalacja serwera np. do katalogu c:\java\tomcat
- ustawienie zmiennych środowiskowych JAVA_HOME=c:\java\jdk TOMCAT_HOME=c:\java\tomcat
- start serwera c:\java\tomcat\bin\startup.bat
- test serwera http://localhost:8080
- zatrzymanie serwera c:\java\tomcat\bin\shutdown.bat



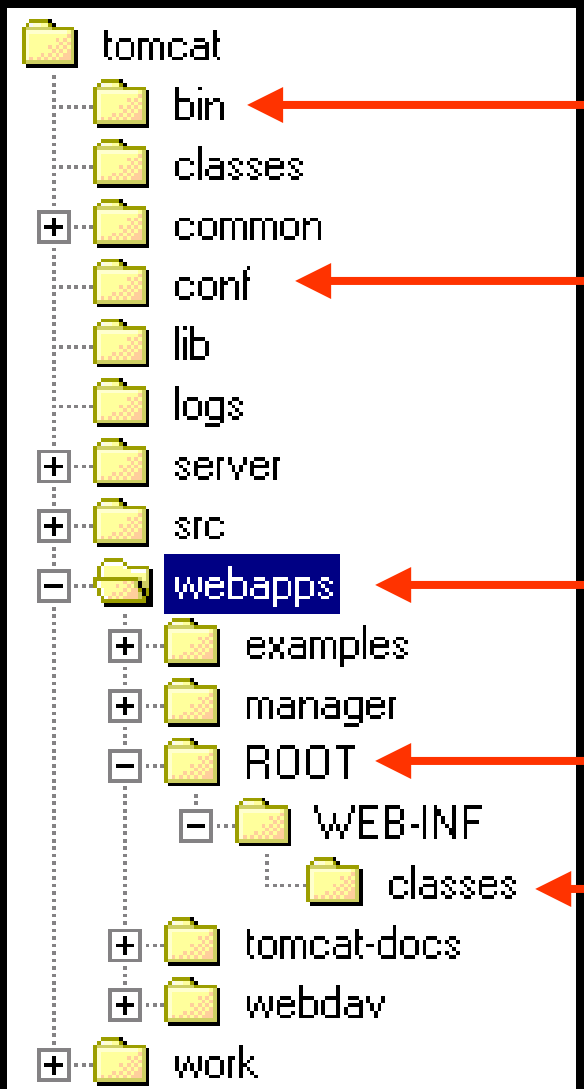
http://localhost:8080





Startup.bat

Struktura katalogów serwera TOMCAT



Katalog bin: uruchomienie (startup.bat) i zatrzymanie (shutdown.bat) serwera

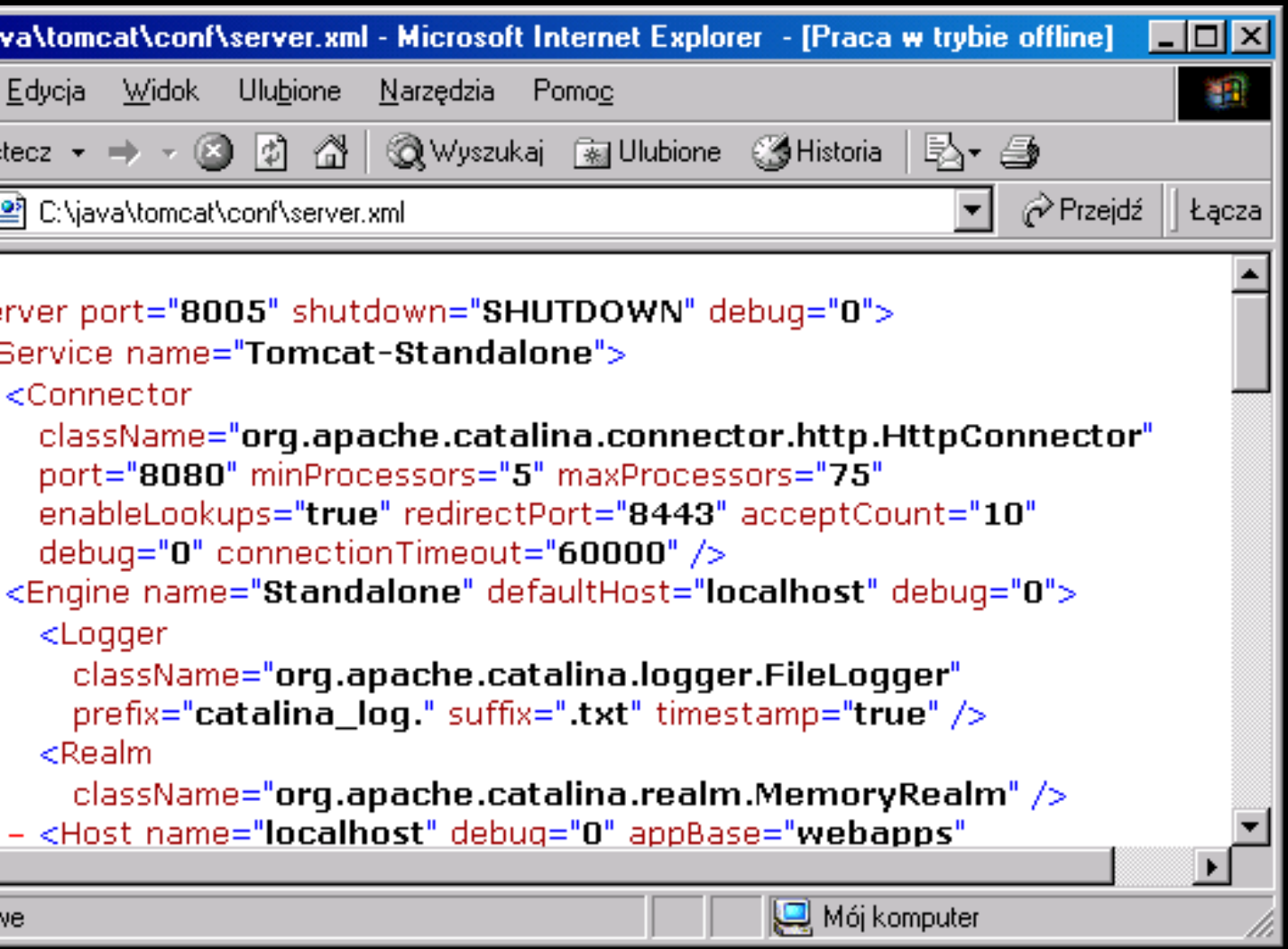
Katalog conf: pliki konfiguracyjne serwera (server.xml i web.xml)

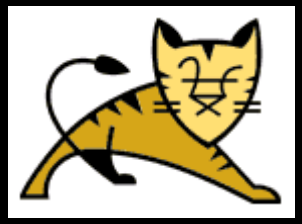
Katalog webapps: katalog zawierający aplikacje webowe.

Katalog do testowania plików jsp

Katalog do testowania serwletów

Plik konfiguracyjny server.xml





Zmiana numeru portu

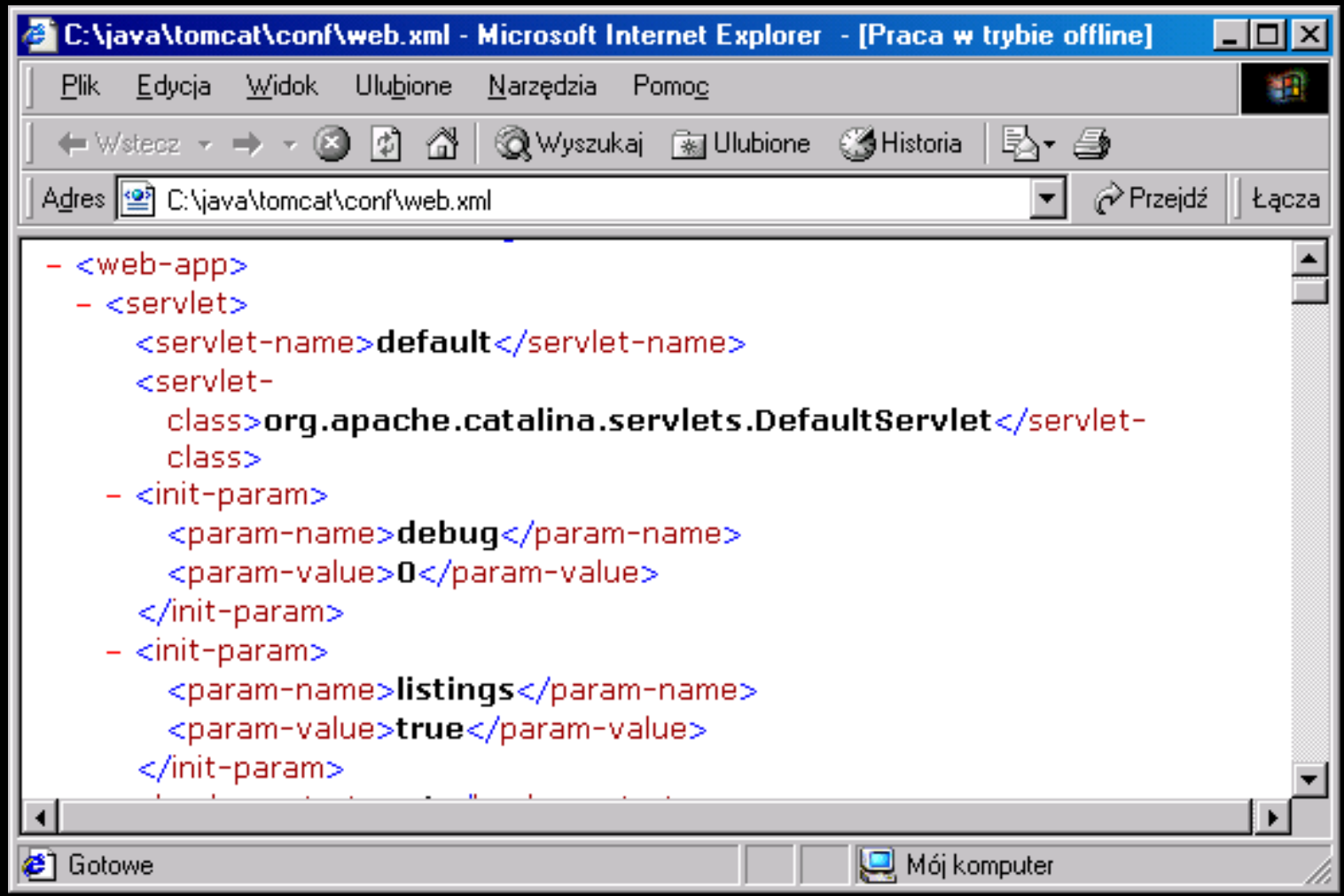
- Modyfikacja pliku konfiguracyjnego serwera c:\java\tomcat\conf\server.xml

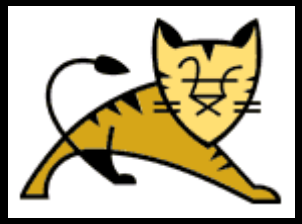
```
<Connector
        className="org.apache.catalina.connector.http.HttpConnector"
        port="8080"
        minProcessors="5"
        maxProcessors="75"
        enableLookups="true"
        redirectPort="8443" acceptCount="10"
        debug="0" connectionTimeout="60000"
/>
```

Zamiana numeru portu z 8080 na 80

- Restart serwera
- Test serwera http://localhost:80 lub http://localhost

Plik konfiguracyjny web.xml





http://localhost/examples/jsp
http://localhost/examples/servlets

Servlet Examples - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Historia

Adres http://localhost/examples/Start ts/index.html Przejdź Łącza

Servlet Examples with Code

This is a collection of examples which der
parts of the Servlet API. Familiarity with t
assumed.

These examples will only work when viev
you are viewing these pages via a "file://...
provide with this Tomcat release regardin
web server.

Wherever you see a form, enter some dat

Gotowe

JSP Examples - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Historia

Adres http://localhost/examples/jsp/index.html Przejdź Łącza

JSP Samples

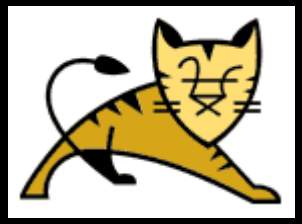
This is a collection of samples demonstrates the usage of different parts of the Java Server Pages (JSP) specification.

These examples will only work when these pages are being served by the ServletRunner. They will not work if you are viewing these pages via a "file://..." URL.

To navigate your way through the examples, the following icons will help:

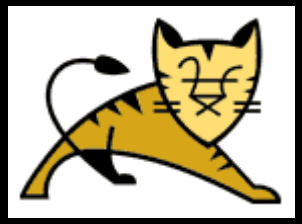
Execute the example

Gotowe Lokalny intranet

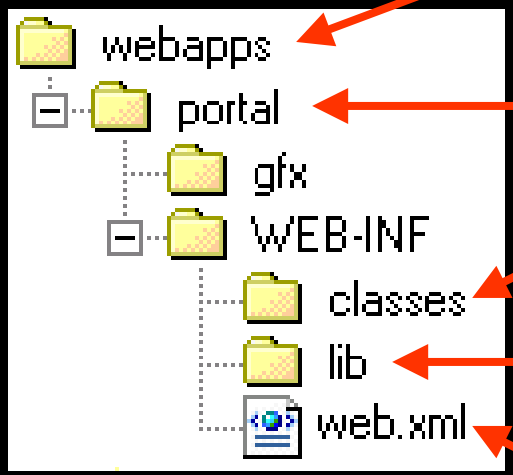


Aplikacja WEB

- JRE (serwer)
- JSP
- Servlet
- Server-side JavaBeans
- statyczne strony HTML, XHTML, XML...
- Klasy po stronie klienta (applety, JavaBeans, klasy JAVA
- JRE (client)



Struktura aplikacji WEB



Katalog webapps: katalog zawierający aplikacje webowe.

aplikacja WEB (pliki html, graficzne, jsp,...)

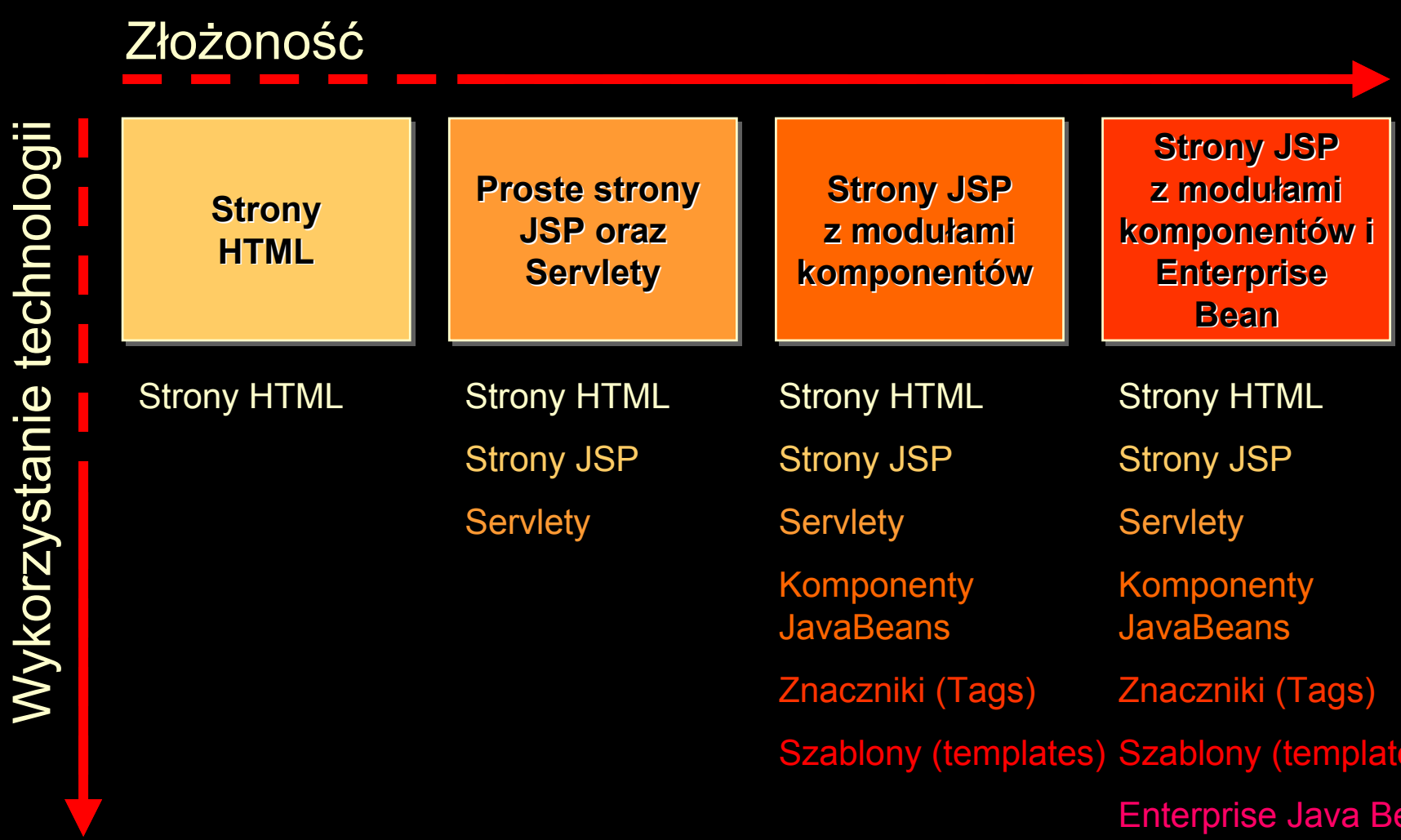
pliki class (serwlety)

pliki jar

plik konfiguracyjny web.xml aplikacji WEB

JSP

Złożoność projektu strony WWW



JSP

- Uproszczenie tworzenia i zarządzania dynamicznym tworzeniem stron WWW (wywoływanie programów po stronie serwera);
- Łączenie kodu Java z HTML
- Oddzielenie graficznego wyglądu strony od jej zawartości

JSP, ASP, PHP

JSP

- technologia firmy SUN z wsparciem innych firm (IBM, Borland, Oracle, Iplanet,...)
- java - lepsze wsparcie przy ponownym wykorzystaniu kodu komponentów
- bogactwo rozwiązań dla przedsiębiorstw (Enterprise)
- wieloplatformowość
- bezpieczeństwo (mechanizmy ochrony wbudowane w język java)

ASP

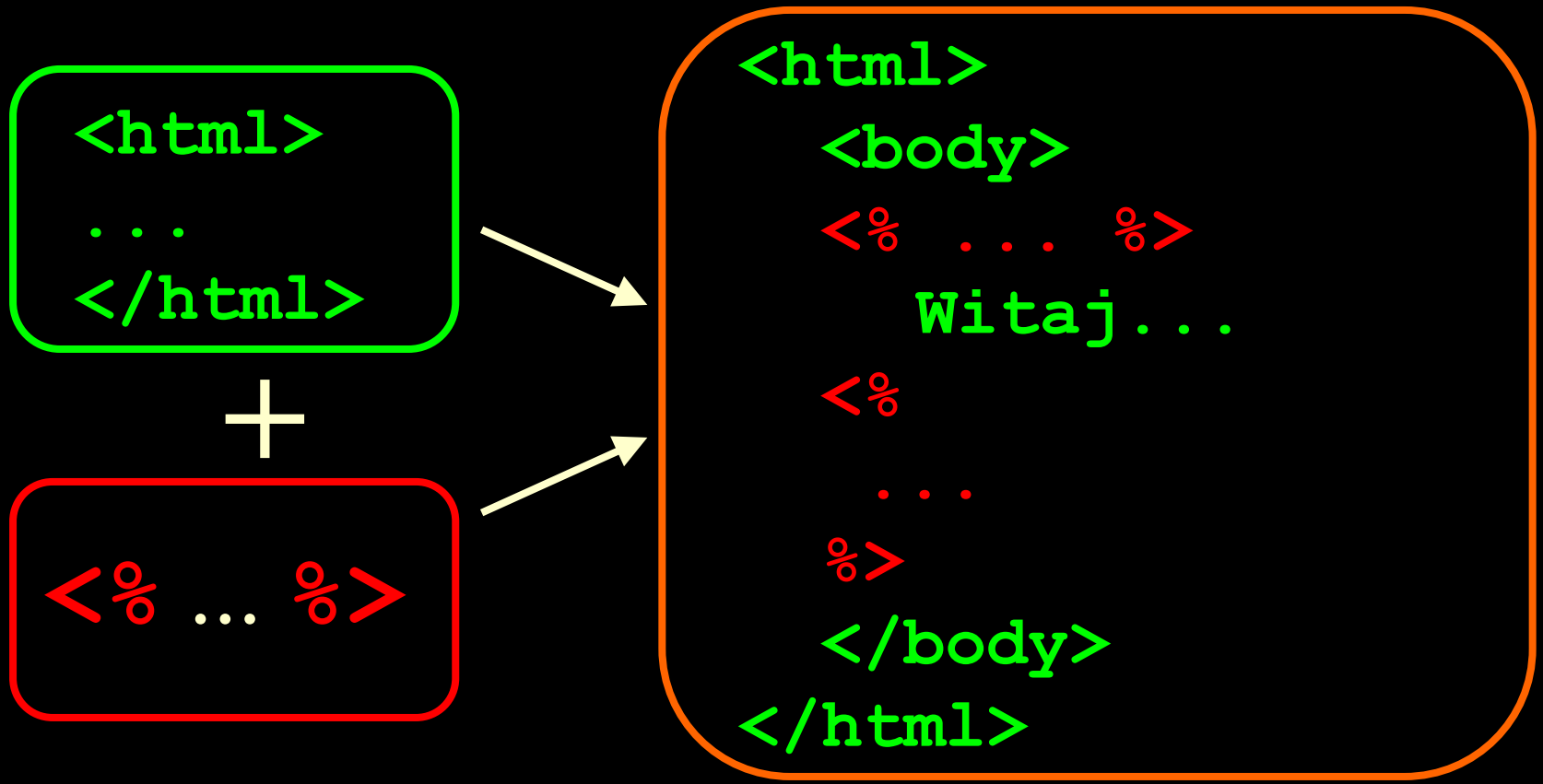
- technologia firmy Micro$oft
- skrypty w VBScript i innych językach specyfikacji ASP (#C)
- ograniczenie do systemu Windows i serwera IIS

PHP

- open-source
- nowy język skryptowy
- wieloplatformowość

JSP

- JSP - powiązanie HTML i kodu Javy poprzez wykorzystanie znaczników JSP

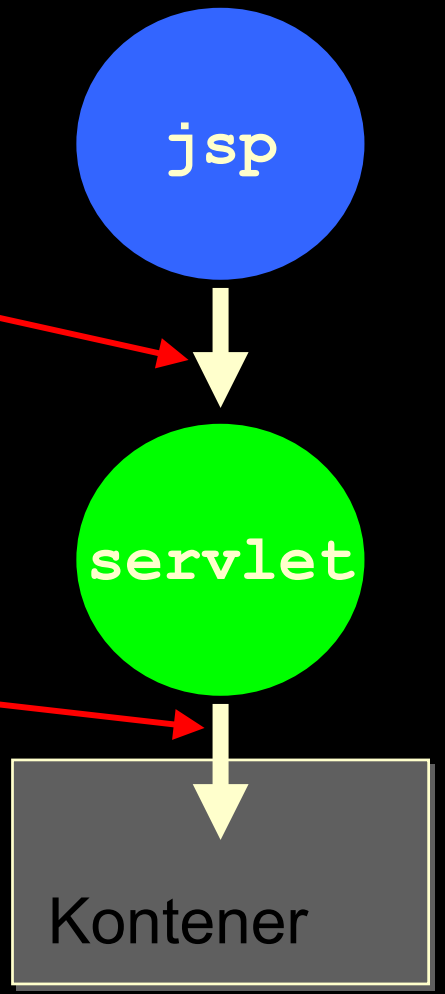


Ładowanie strony JSP przez kontener

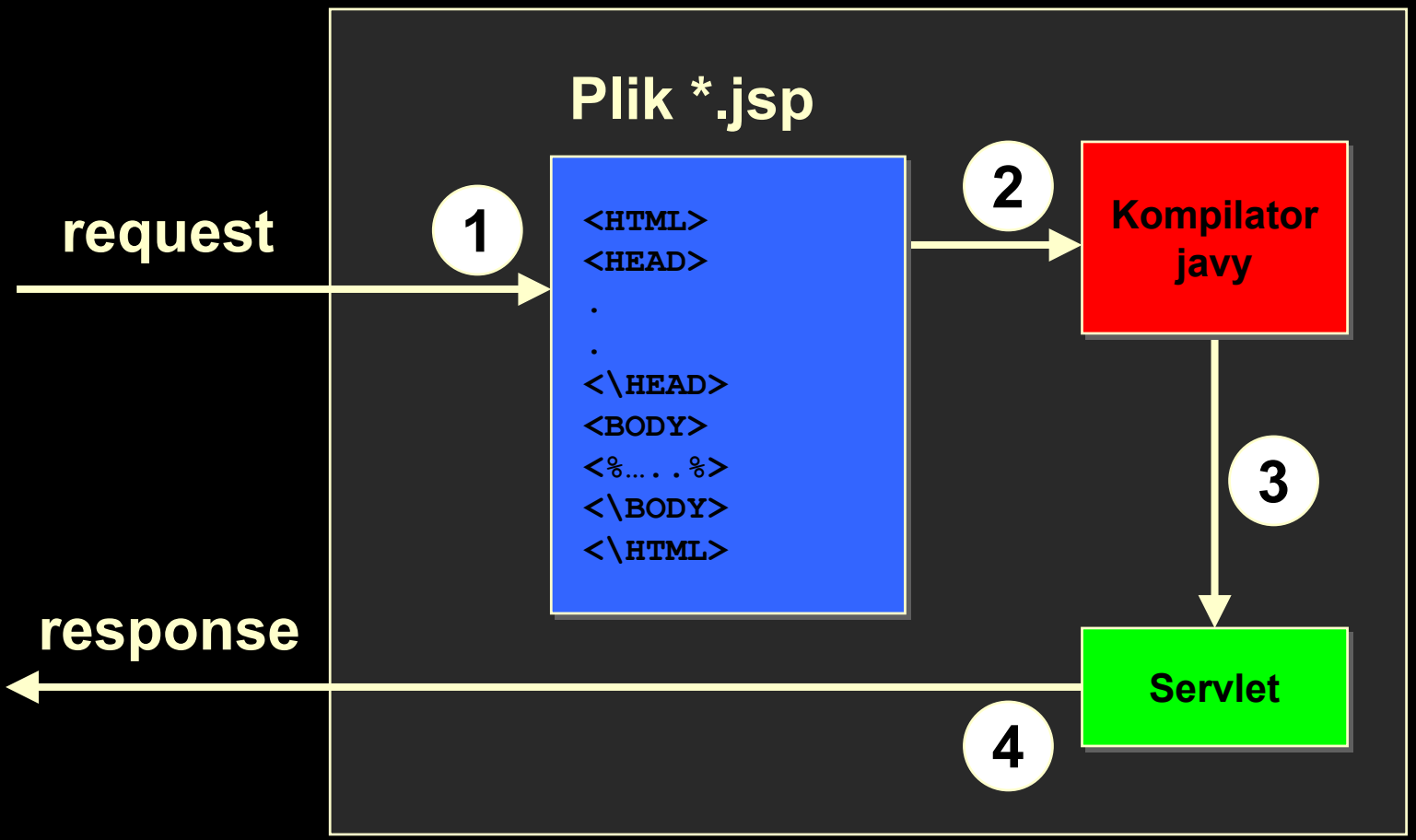
Pierwsze ładowanie strony JSP:

- generacja kodu servletu z JSP
- kompilowanie kodu servletu

- ładowanie servletu do kontenera



Ładowanie strony JSP



JSP => Servlet => class

witaj$jsp.java

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class witaj$jsp extends HttpJspBase {

    static {
    }
    public witaj$jsp( ) {
    }

    private static boolean _jspx_inited = false;

    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {
    }
...
...
```

witaj.jsp

```
<html>
  <head>
    <title>Witaj</title>
  </head>
<body>
            <% out.println("Witaj"); %>
</body>
</html>
```

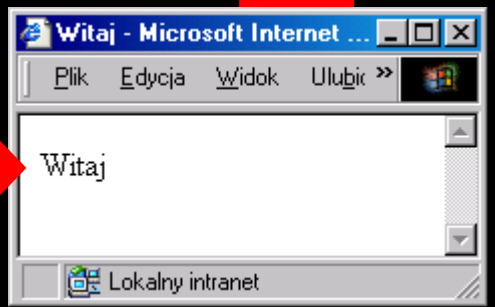
2

1

3

Witaj$jsp.class

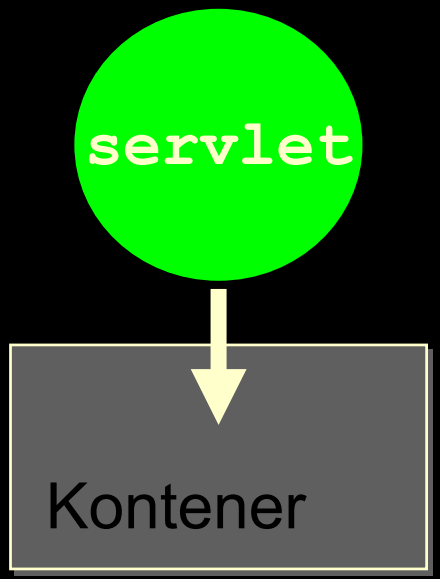
4



Ponowne odwołanie klienta do strony JSP

W przypadku braku modyfikacji strony JSP:

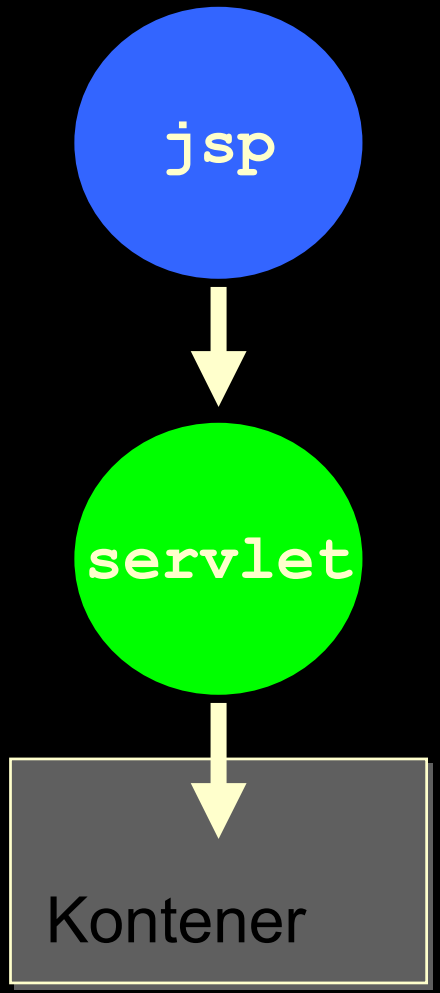
- strona JSP zachowuje się jak statyczna strona HTML z powiązanymi servletami
- generacja kodu wysyłanego do klienta przez skompilowany servlet



Ponowne odwołanie klienta do strony JSP

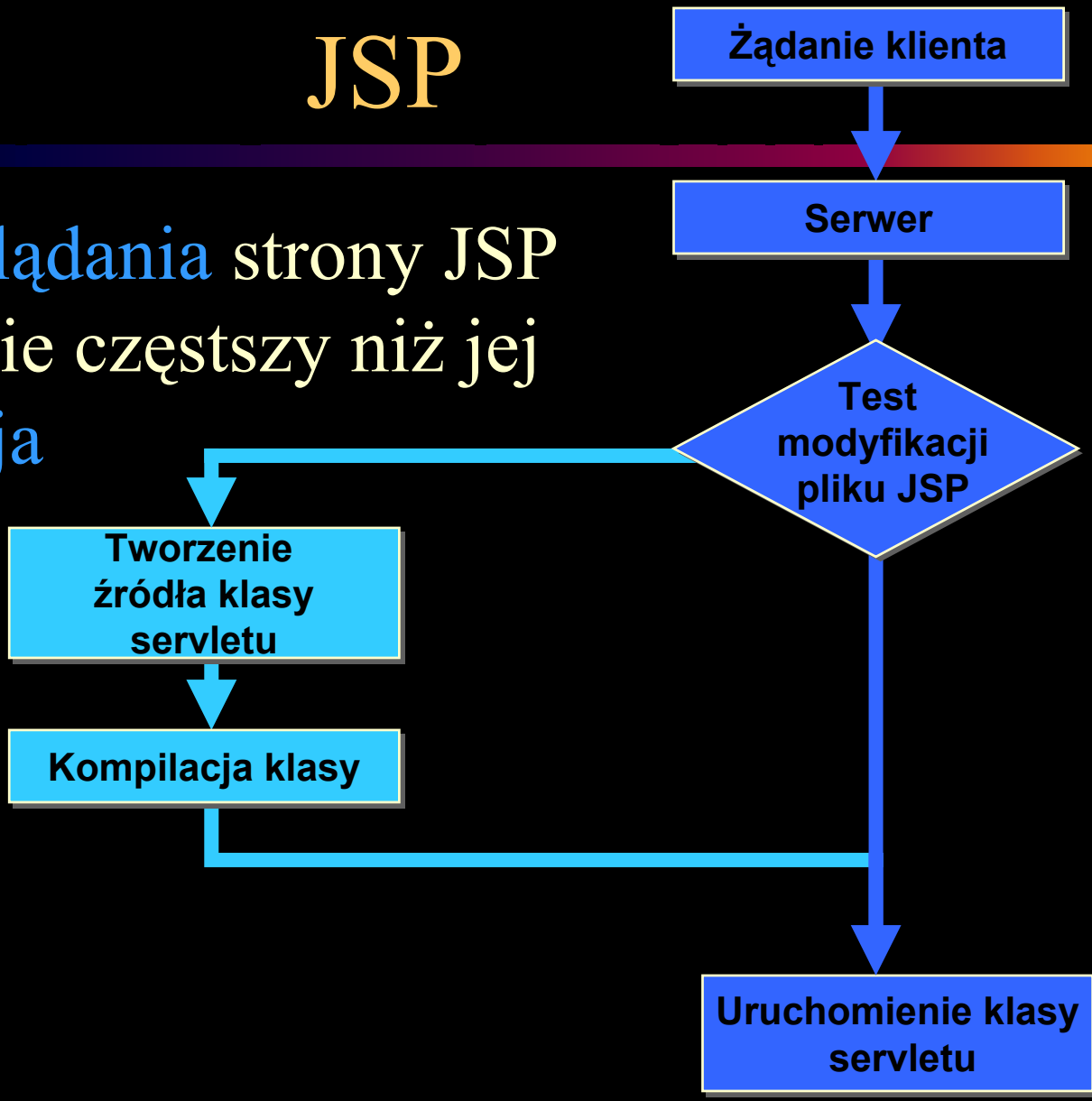
W przypadku modyfikacji strony JSP:

- ponowna kompilacja i przeładowanie strony przy następnym żądaniu strony
- dłuższy czas odpowiedzi przy pierwszym żądaniu

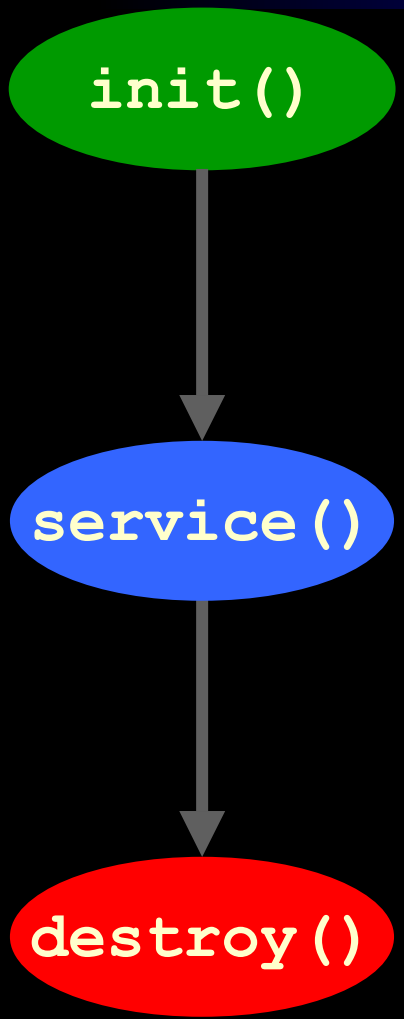


JSP

Proces przeglądania strony JSP jest znacznie częstszy niż jej modyfikacja



Cykl życia



`init()` - rozpoczęcie „życia” strony JSP (servletu)

- jednokrotne wywołanie
- tworzenie i inicjalizacja zasobów i danych

`service()` - obsługa żądań (zleceń) zgłaszanych przez klienta i obsługa odpowiedzi

`destroy()` - zakończenie działania strony JSP (servletu)

- zwalnianie zasobów utworzonych w metodzie `init()`
- zamykanie połączeń z bazami danych

JSP (servlet) - cykl życia

JSP i Servlety

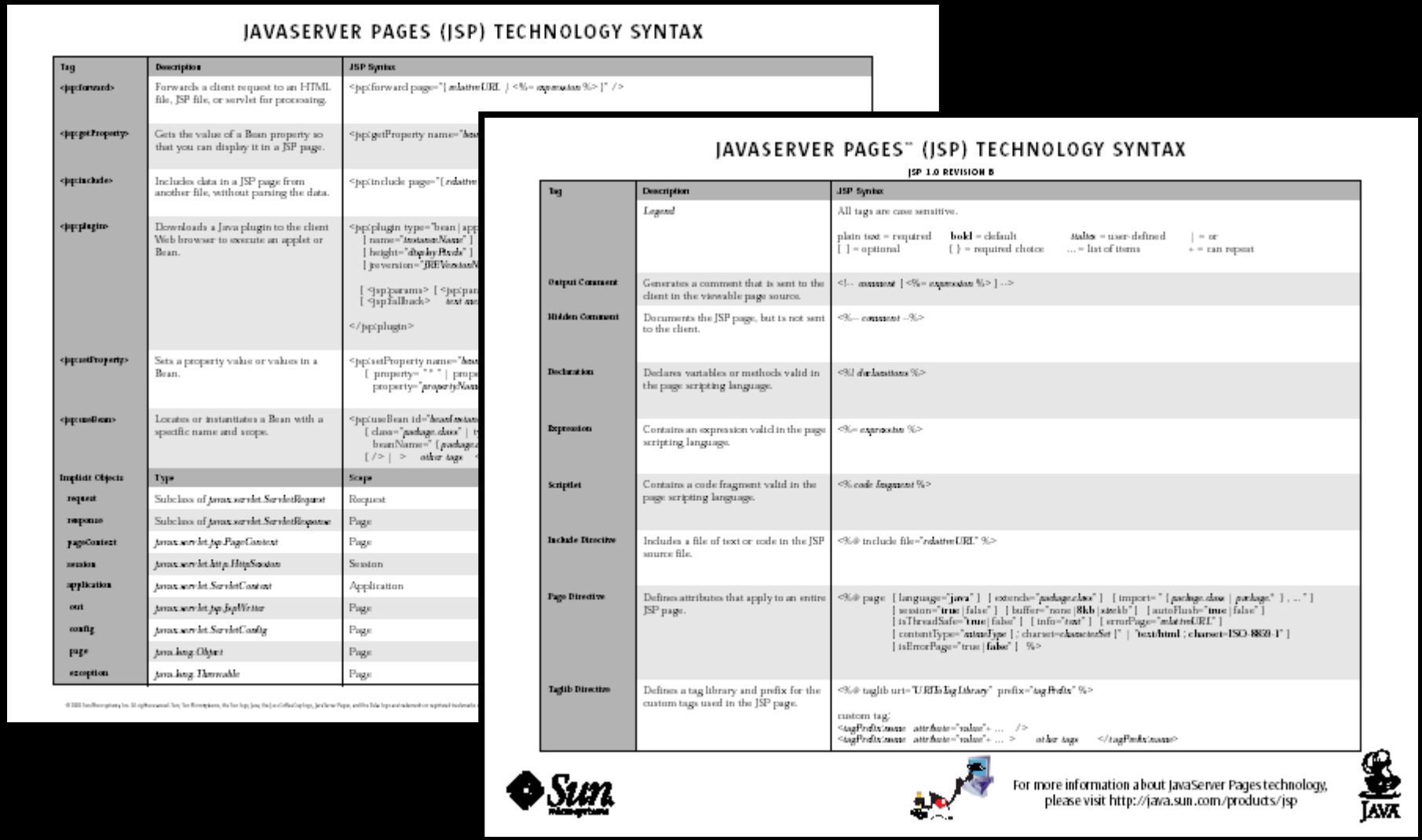
- Programowanie w JSP nie wymusza znajomości serwletów
- Łatwa integracja z kodem HTML
- Brak potrzeby ustawiania zmiennej środowiskowej CLASSPATH oraz kompilacji kodu JSP (dba o to serwer aplikacji)
- Brak wymogu umieszczania kodu JSP w specjalnym katalogu (jak to ma miejsce z servlet'ami)

Elementy skryptowe

- tworzenie i modyfikacja zawartości stron
- manipulowanie obiektami

- Skryptlety `<% ... %>`
- Dyrektywy `<%@ ... %>`
- Wyrażenia `<%= ... %>`
- Deklaracje `<%! ... %>`
- Akcje `<jsp: ... >`

Składnia JSP



JAVASERVER PAGES (JSP) TECHNOLOGY SYNTAX

Tag	Description	JSP Syntax
<jsp:forward>	Forwards a client request to an HTML file, JSP file, or servlet for processing.	<jsp:forward page="{ relativeURL | <%= expression %> }" />
<jsp:getProperty>	Gets the value of a Bean property so that you can display it in a JSP page.	<jsp:getProperty name="bean
<jsp:include>	Includes data in a JSP page from another file, without parsing the data.	<jsp:include page="{ relative
<jsp:plugin>	Downloads a Java plugin to the client Web browser to execute an applet or Bean.	<jsp:plugin type="bean | app { name="beanInstanceName" } { height="displayPixels" } { jreversion="JRE Version Number { <jsp:params> { <jsp:par { <jsp:fallback> text mes </jsp:plugin>
<jsp:setProperty>	Sets a property value or values in a Bean.	<jsp:setProperty name="bean { property="*" | prop property="propertyName
<jsp:useBean>	Locates or instantiates a Bean with a specific name and scope.	<jsp:useBean id="beanInstanc { class="package.class" | t beanName="{ package.c { /> | > other tags

Implicit Objects	Type	Scope
request	Subclass of javax.servlet.ServletRequest	Request
response	Subclass of javax.servlet.ServletResponse	Page
pageContext	javax.servlet.jsp.PageContext	Page
session	javax.servlet.http.HttpSession	Session
application	javax.servlet.ServletContext	Application
out	javax.servlet.jsp.JspWriter	Page
config	javax.servlet.ServletConfig	Page
page	java.lang.Object	Page
exception	java.lang.Throwable	Page

JAVASERVER PAGES™ (JSP) TECHNOLOGY SYNTAX

JSP 1.0 REVISION B

Tag	Description	JSP Syntax
	Legend	All tags are case sensitive. plain text = required **bold** = default *italics* = user-defined | = or [ ] = optional { } = required choice ... = list of items + = can repeat
Output Comment	Generates a comment that is sent to the client in the viewable page source.	<!-- *comment* { <%= *expression* %> } -->
Hidden Comment	Documents the JSP page, but is not sent to the client.	<%-- *comment* --%>
Declaration	Declares variables or methods valid in the page scripting language.	<%! *declarations* %>
Expression	Contains an expression valid in the page scripting language.	<%= *expression* %>
Scriptlet	Contains a code fragment valid in the page scripting language.	<% *code fragment* %>
Include Directive	Includes a file of text or code in the JSP source file.	<%@ include file="*relativeURL*" %>
Page Directive	Defines attributes that apply to an entire JSP page.	<%@ page { language="**java**" } { extends="*package.class*" } { import=" { *package.class* | *package.** } , ... " } { session="**true** | false" } { buffer="none | **8kb** | *size*kb" } { autoFlush="**true** | false" } { isThreadSafe="**true** | false" } { info="*text*" } { errorPage="*relativeURL*" } { contentType="*mimeType* { ; charset=*characterSet* }" | "**text/html ; charset=ISO-8859-1**" } { isErrorPage="true | **false**" } %>
Taglib Directive	Defines a tag library and prefix for the custom tags used in the JSP page.	<%@ taglib uri="*URIToTagLibrary*" prefix="*tagPrefix*" %> custom tag: <*tagPrefix:name* *attribute*="*value*"+ ... /> <*tagPrefix:name* *attribute*="*value*"+ ... > *other tags* </*tagPrefix:name*>

Sun Microsystems

For more information about JavaServer Pages technology, please visit http://java.sun.com/products/jsp

JAVA

Komentarze

- HTML - przesyłane do klienta wraz z kodem strony (*dostępne w przeglądarce klienta*)

```
<!--komentarz HTML -->

<!--komentarz <%= wyrażenie %> -->
```

- JSP - opisujące kod strony (*dostępne dla twórcy strony*)

```
<%-- komentarz JSP --%>

<% /** komentarz JAVA **/ %>
```

<!--komentarz HTML -->

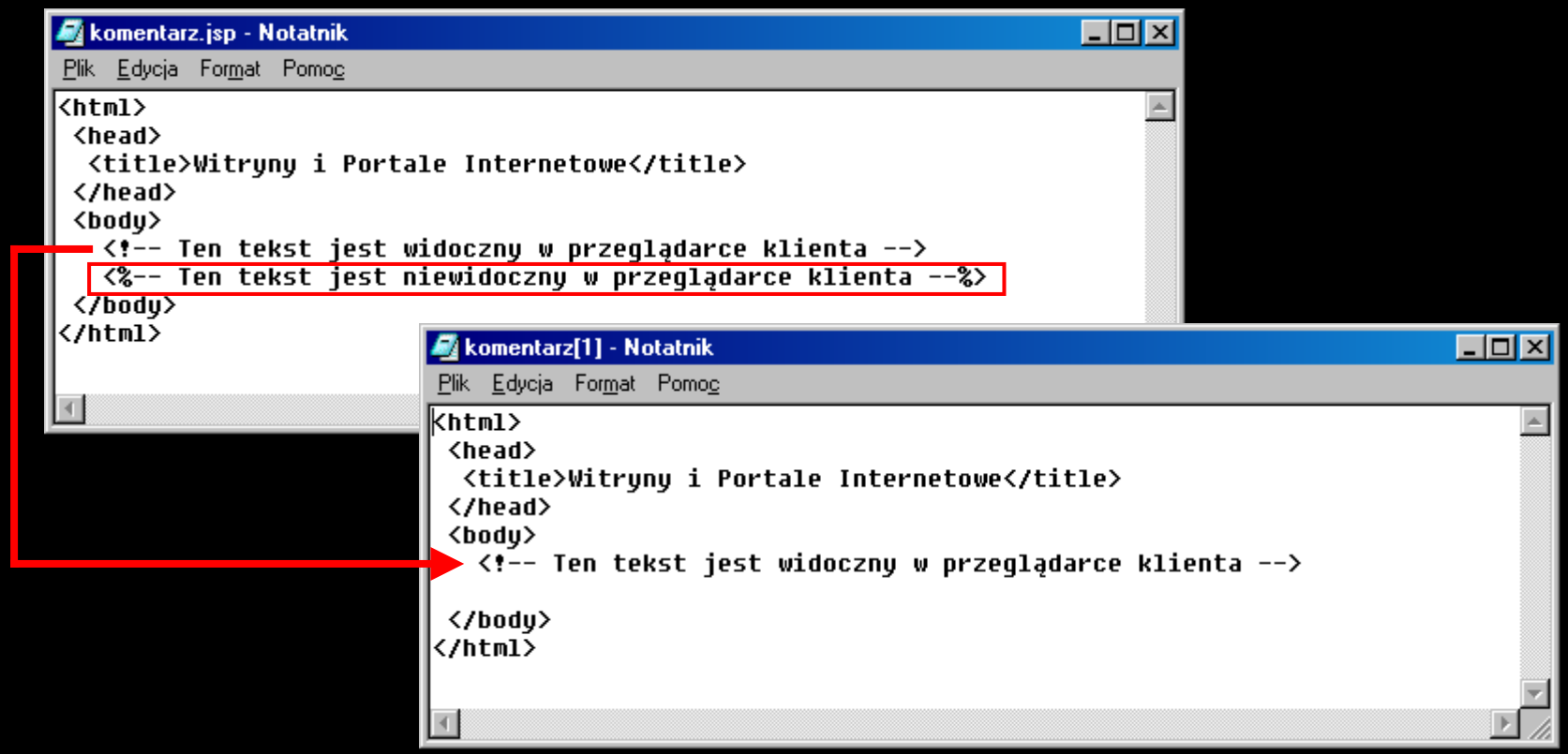
```
<html>
     <body>
     <h1>Tytuł</h1>
     <!--
          Ten tekst jest widoczny w
           przeglądarce klienta
     -->
     </body>
</html>
```

<%-- komentarz JSP --%>

```
<html>
     <body>
     <h1>Tytuł</h1>
     <%--
          Ten tekst jest niewidoczny w
          przeglądarce klienta
     --%>
     </body>
</html>
```

Komentarze HTML i JSP

Strona JSP (serwer)

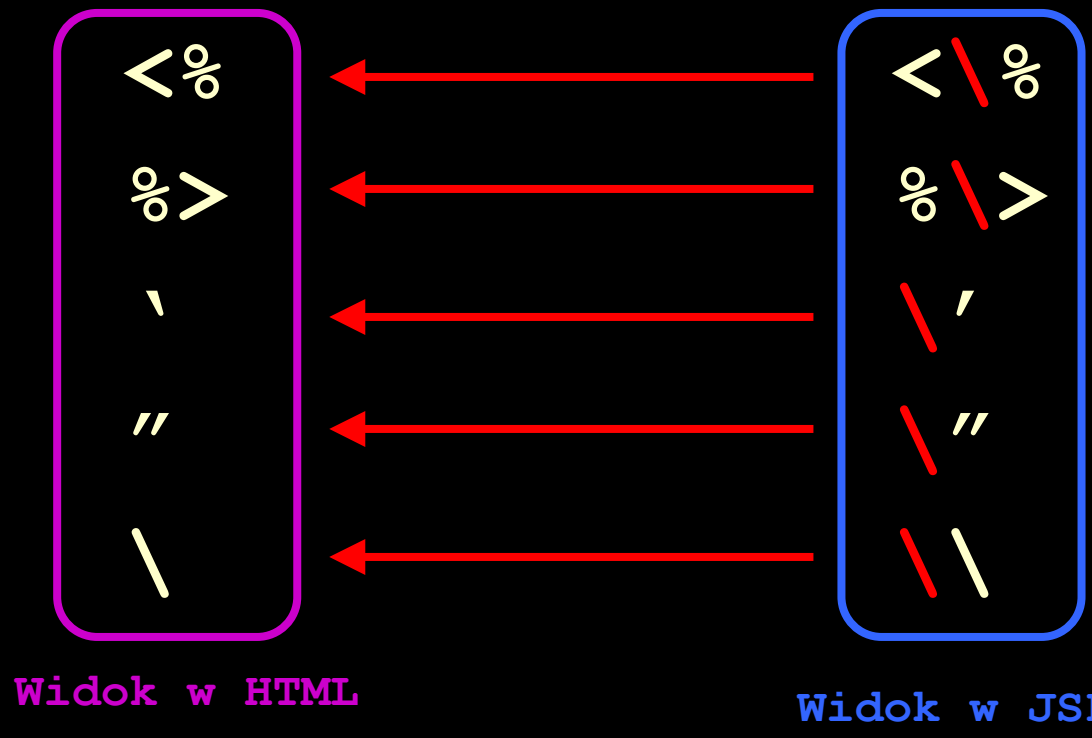


Przeglądarka (klient)

W HTML nie istnieje możliwość tworzenia zagnieżdżonych komentarzy

Znaki specjalne

W JSP aby wyświetlić znak specjalny należy (podobnie jak w C++) wykorzystać dodatkowy znak \



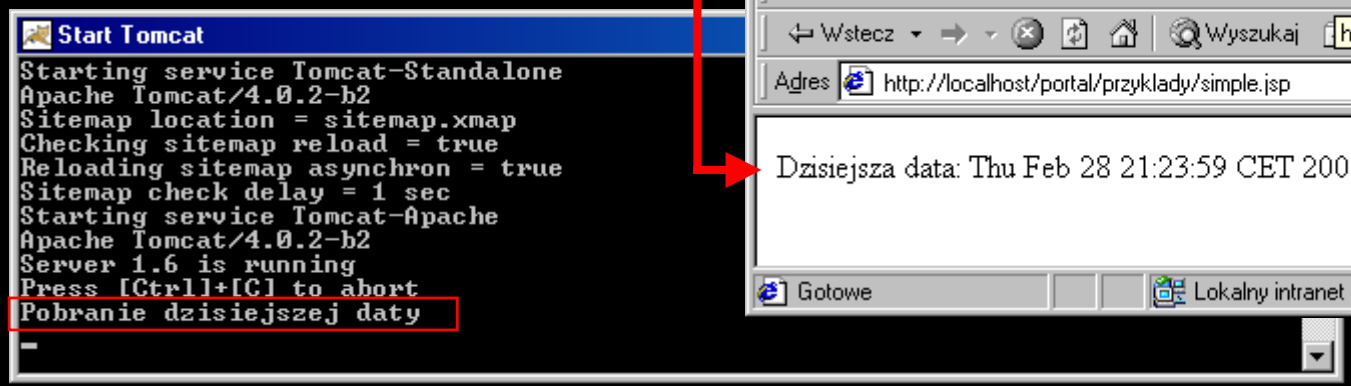
Skryptlety

```
<html>
  <body>
    <%
      System.out.println("Pobranie dzisiejszej daty");
      java.util.Date data = new java.util.Date();
    %>
      Data: <%= data %>
  </body>
</html>
```

Elementy skryptowe ... (skryptlety) ... :-)

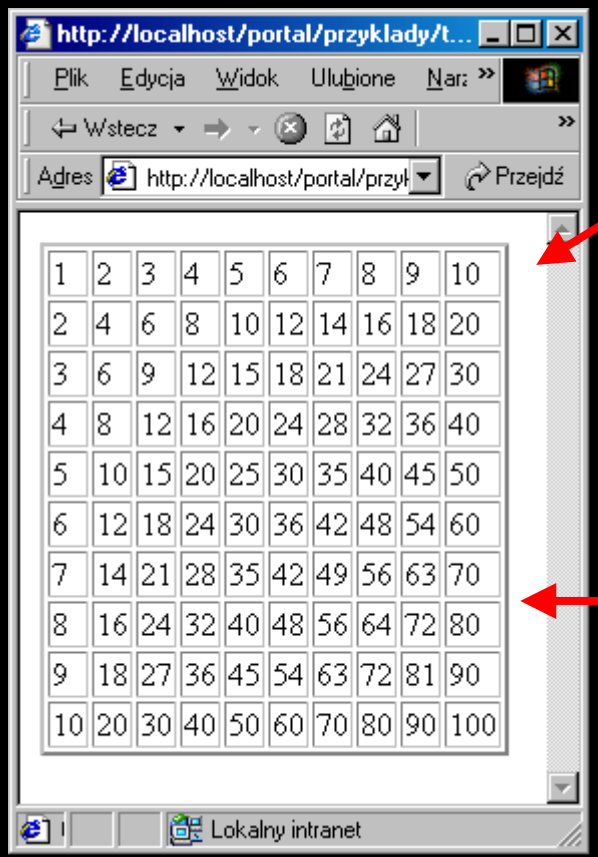
Skryptlety

```
<html>
  <body>
    <%
      System.out.println("Pobranie dzisiejszej daty");
      java.util.Date data = new java.util.Date();
    %>
      Data: <%= data %>
  </body>
</html>
```



Tabliczka mnożenia

Przeglądarka(klient)



Strony JSP (serwer)

tabliczka1.jsp - Notatnik

Plik Edycja Format Pomoc

```
<html>
 <head>
 </head>
<body>

<table border="2">
<% for(int i=1; i<=10; i++){ %>
  <tr>
  <% for(int j=1; j<=10; j++){ %>
    <td><%= i*j %></td>
  <%}%>
  </tr>
<%}%>
</table>
</body>
</html>
```

tabliczka2.jsp - Notatnik

Plik Edycja Format Pomoc

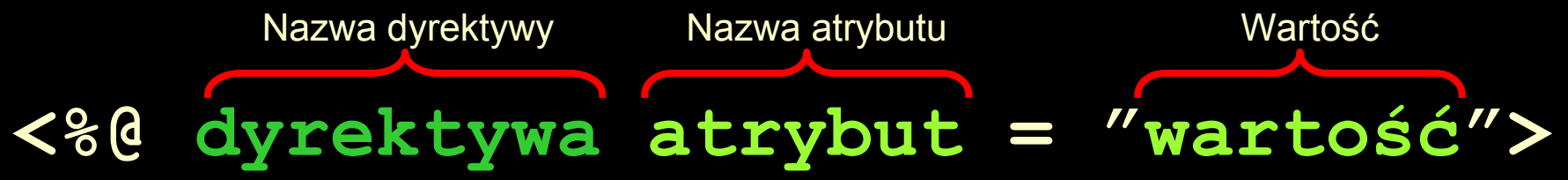
```
<html>
 <head>
 </head>
<body>

<table border="2">
<%
    for( int i=1; i<=10; i++ ) {
      out.write("<tr>");
      for( int j=1; j<=10; j++ ) {
        out.print("<td>"+i*j+"</td>");
      }
      out.println("</tr>");
    }
%>
</table>
</body>
</html>
```

Dyrektywy

Dyrektywy są parametrami służącymi do ustawiania parametrów działania kontenera JSP

- ustawianie parametrów kompilacji/translacji
- definiowanie języka, ...

i nie wyświetlają żadnej zawartości



Rodzaje dyrektyw

`<%@ dyrektywa ... %>`

W JSP zdefiniowano trzy dyrektywy:

- page - definiowanie właściwości strony
- include - dołączenie plików do strony JSP
- taglib - deklarowanie biblioteki znaczników

<%@ ... %>

Dyrektywa page

Dyrektywa page pozwala na definiowanie parametrów dla całej strony JSP (bezpośredni wpływ na proces tłumaczenia kodu javy)

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

<%@ ... %>

Dyrektywa page

language	`<%@ page language="java" %>`
extends	`<%@ page extends="com.taglib..." %>`
import	`<%@ page import="java.util.*" %>`
session	
buffer	`<%@ page buffer="none" %>`
autoFlush	`<%@ page autoFlush="true" %>`
isThreadSafe	`<%@ page isThreadSafe="true" %>`
info	`<%@ page info="PORTAL" %>`
errorPage	`<%@ page error="error.jsp" %>`
IsErrorPage	
contentType	

<%@ ... %>

Parametr language dyrektywy page

- language - określenie języka używanego do dynamicznej generacji zawartości.
  - wartość domyślan: java

```
<%@ page language="java" %>
```

Specyfikacja JSP opisuje tylko użycie języka JAVA, ale dostępna jest również obsługa Javascriptu do generacji JSP

<%@ ... %>

Parametr extends dyrektywy page

- extends - określenie klasy nadrzędnej (bazowej) z której następuje dziedziczenie strony (servletu) JSP.
  - wartość domyślan: org.apache.jasper.runtime.HTTPJspBean

```
<%@ page extends="com.taglib..." %>
```

<%@ ... %>

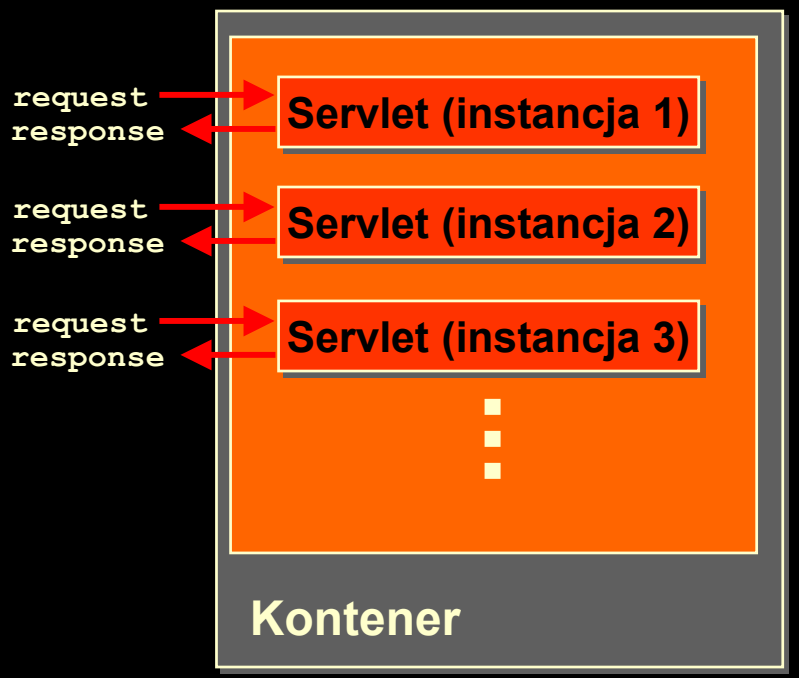
Parametr isThreadSafe dyrektywy page

- isThreadSafe - określenie poziomu bezpieczeństwa wątku

```
<%@ page isThreadSafe="false" %>
```

false - kontener przesyła żądania klienta jedno po drugim w kolejności ich nadchodzenia

true - kontener przesyła jednocześnie wszystkie żądania klienta do strony

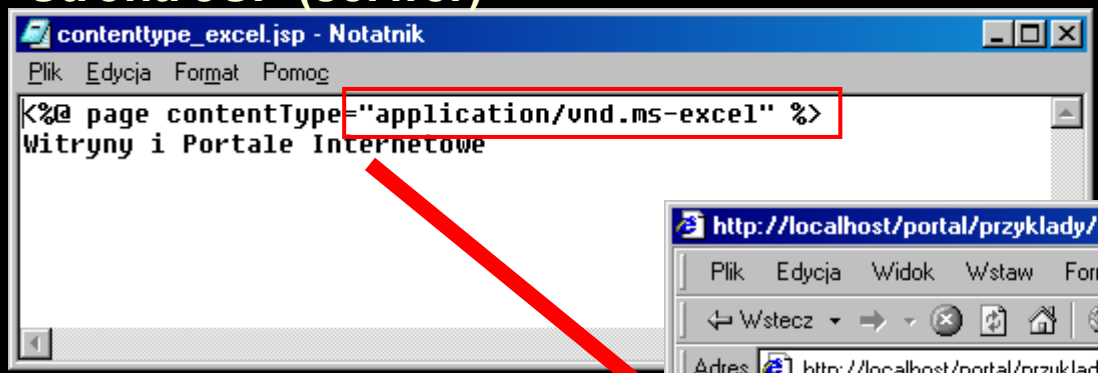


<%@ ... %>

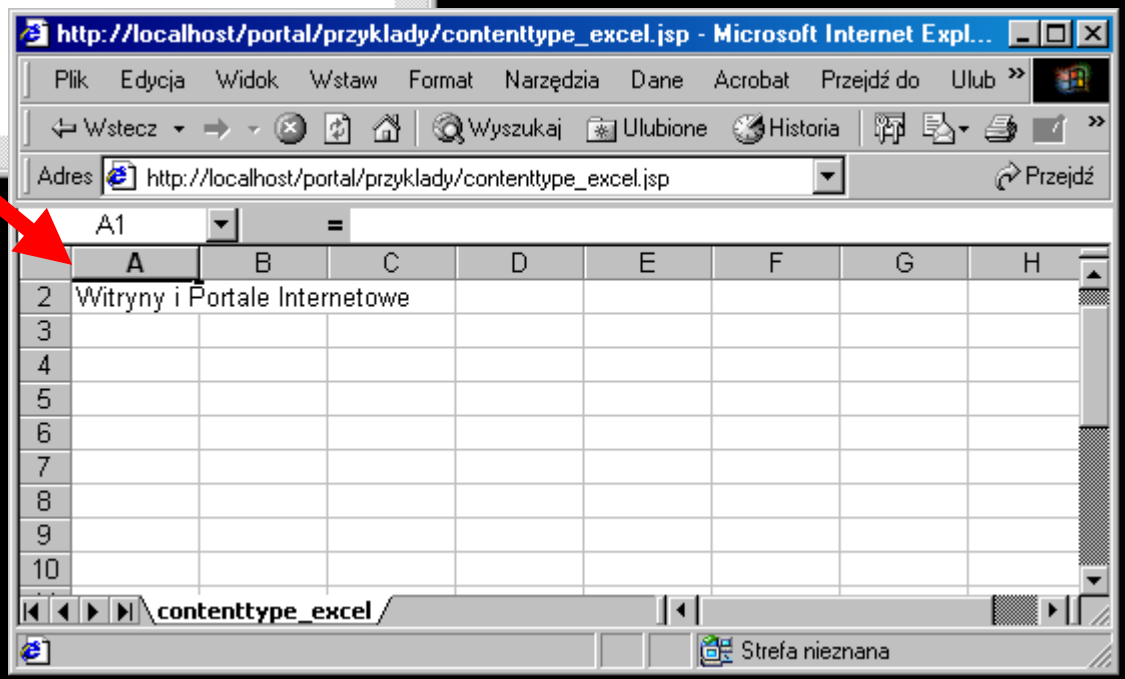
Parametr contentType dyrektywy page

```
<%@ page contentType = "application/vnd.ms-excel" %>
```

Strona JSP (serwer)



Przeglądarka+Excel (klient)



MIME { **M**ultipurpose **I**nternet **M**ail **E**xtension

<%@ ... %>

Parametr isErrorPage dyrektywy page

isErrorPage - deklarowanie strony jako strony diagnostycznej...

```
<%@ page isErrorPage = "true" %>
```

Strona JSP (serwer)

errorpage.jsp - Notatnik

Plik Edycja Format Pomoc

```
<html>
  <head>
    <title>Errorpage</title>
  </head>
<body>
<%@ page isErrorPage="true" %>

  <h1>Strona diagnostyczna...</h1>

</body>
</html>
```

Przeglądarka (klient)

Errorpage - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Historia

Adres http://localhost/portal/przyklady/errorpage.jsp Przejdź

Strona diagnostyczna...

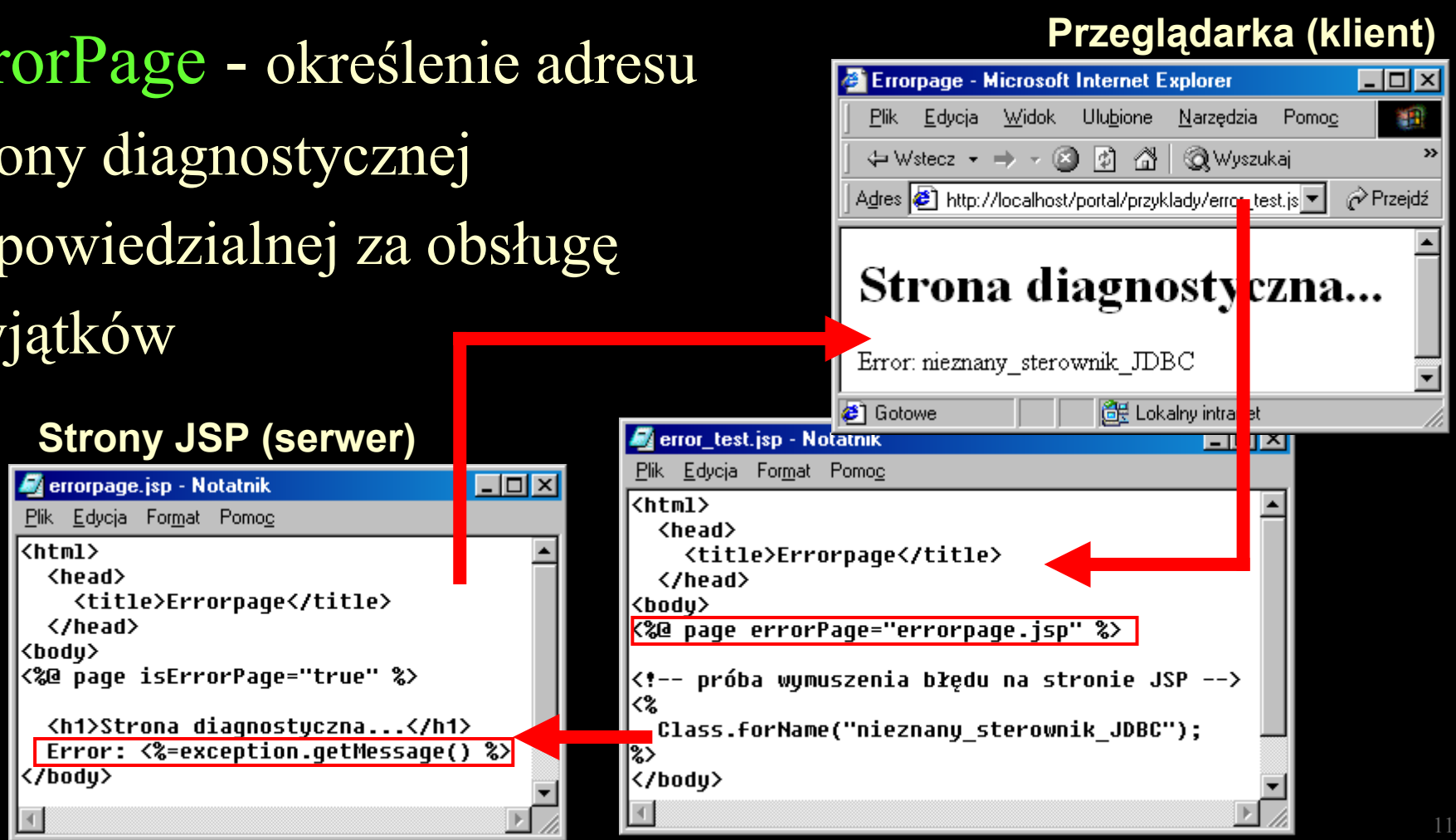
Gotowe Lokalny intranet

<%@ ... %>

Parametr erorrPage dyrektywy page

```
<%@ page errorPage = "errorpage.jsp" %>
```

errorPage - określenie adresu strony diagnostycznej odpowiedzialnej za obsługę wyjątków



<%@ ... %>

Parametr session dyrektywy page

```
<%@ page session = "true" %>
```

Strona JSP (serwer)

session_true.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page session="true" %>
<html>
 <head>
 </head>
 <body>
  <h1>Session</h1>
 </body>
</html>
```

Konsola standard

```
GET /przyklady/session_true.jsp HTTP1.1

HTTP/1.1 200 OK
Content-Type: text/html;charset=ISO-8859-1
Date: Mon, 11 Mar 2002 22:20:10 GMT
Server: Apache Tomcat/4.0.3 (HTTP/1.1 Connector)
Connection: close
Set-Cookie: JSESSIONID=2F2FDC5BDEDB99DEDCFCCE9CE5E0811F;Path=/


<html>
 <head>
 </head>
```

Telnet (klient)

Strona JSP (serwer)

session_false.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page session="false" %>
<html>
 <head>
 </head>
 <body>
  <h1>Session</h1>
 </body>
</html>
```

Konsola standard

```
GET /przyklady/session_false.jsp HTTP1.1

HTTP/1.1 200 OK
Content-Type: text/html;charset=ISO-8859-1
Date: Mon, 11 Mar 2002 22:18:25 GMT
Server: Apache Tomcat/4.0.3 (HTTP/1.1 Connector)
Connection: close


<html>
 <head>
 </head>
 <body>
  <h1>Session</h1>
 </body>
</html>
```

Telnet (klient)

<%@ ... %>

Dyrektywa include

Dyrektywa include służy do wstawiania tekstu i kodu do dokumentu JSP w procesie jego translacji

```
<html>
  <body>
      <%@ include file="baner.jsp" %>
</body>
</html>
```

Realizacja procesu translacji pliku baner.jsp i jego wstawienie w miejsce dyrektywy include

<%@ ... %>

Dyrektywa include

include_main.jsp - Notatnik

Plik Edycja Format Pomoc

```
<html>
  <head>
    <title><%@include file='include_template.htm'%></title>
  </head>
<body>
<h1><%@include file='include_template.htm'%></h1>
<table border="1">
   <tr><td><%@include file='include_template.htm'%></td>
      <td><table>
         <tr><td><%@include file='include_template.htm'%></td></tr>
         <tr><td><%@include file='include_template.htm'%></td></tr>
         </table>
      </td>
   </tr>
</table>
<a href=""><%@include file='include_template.htm'%></a>
</body></html>
```

Strona JSP (serwer)

include_template.htm - Notatnik

Plik Edycja Format Pomoc

```
Witryny i Portale Internetowe
```

Przeglądarka (klient)

Witryny i Portale Internetowe - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Historia

Adres http://localhost/portal/przyklady/include_main.jsp Przejdź

Witryny i Portale Internetowe

Witryny i Portale Internetowe	Witryny i Portale Internetowe Witryny i Portale Internetowe

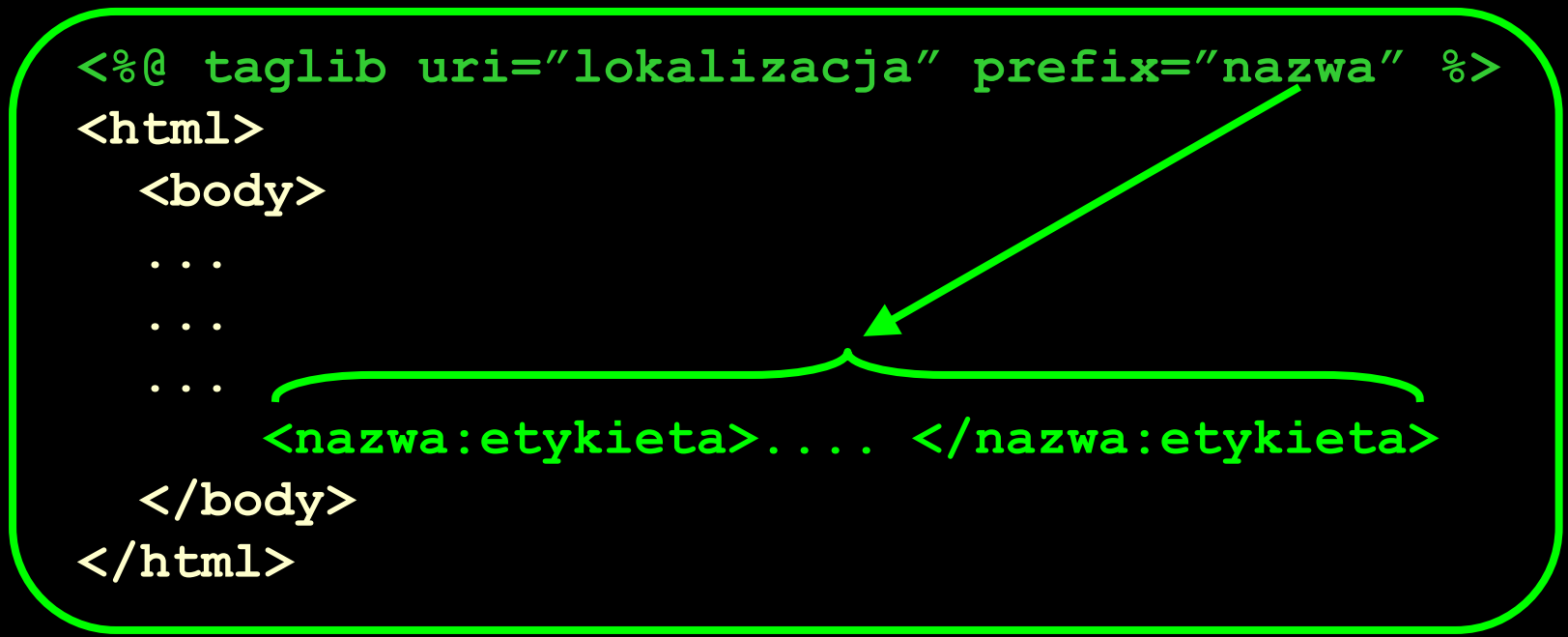
Witryny i Portale Internetowe

Lokalny intranet

<%@ ... %>

Dyrektywa taglib

Dyrektywa taglib deklaruje wykorzystanie biblioteki znaczników zdefiniowanych przez użytkownika



Upodobnienie technologii JSP do HTML

<%@ ... %>

Dyrektywa taglib

Biblioteka TagLib (serwer)

portal.tld - Notatnik

Plik Edycja Format Pomoc

```
<?xml version="1.0" encoding="ISO-8859-1"
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.
        "http://java.sun.com/j2ee/dtds/we

<taglib>

  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>portal</shortname>
  <uri>hdl.ie.tu.koszalin.pl</uri>
  <info>Witryny i Portale Internetowe</in

  <tag>
    <name>data</name>
    <tagclass>portal.data</tagclass>
    <bodycontent>empty</bodycontent>
    <info>portal</info>
  </tag>
```

Strona JSP (serwer)

tag_data.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<%@ taglib uri="/WEB-inf/portal.tld" prefix="portal" %>
<html>
 <head>
  <title>Pobieranie bieżącej daty</title>
 </head>
 <body>
        Dzisiaj jest <portal:data/>
 </body>
</html>
```

Przeglądarka (klient)

Pobieranie bieżącej daty - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione

Adres http://localhost/portal/przyklady/tag_data.jsp Przejdź

Dzisiaj jest: 5 marzec 2002

Gotowe Lokalny intranet

Wyrażenia

Wyrażenie wyjściowe powoduje skierowanie wartości zawartej w znaczniku do strumienia wyjściowego

```
<%= wyrażenie %>
```

Wyrażenie wyświetla daną zawartości poprzez konwersję na typ java.lang.String

<%= ... %>

Wyrażenia

<%= wyrażenie %>

```
<html>
  <body>
   <%= 2*2 %>
  </body>
</html>
```

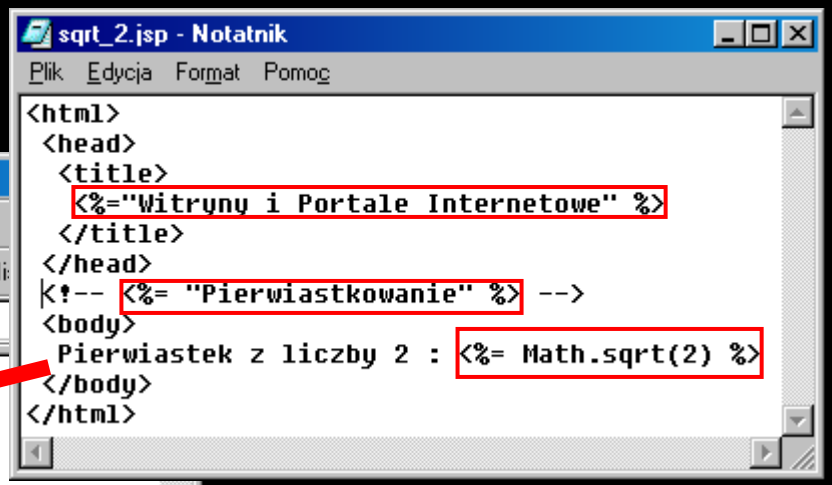
```
<html>
  <body>
   <%= new java.util.Date() %>
  </body>
</html>
```

```
<html>
  <body>
   <%= "Witryny i Portale Internetowe"%>
  </body>
</html>
```

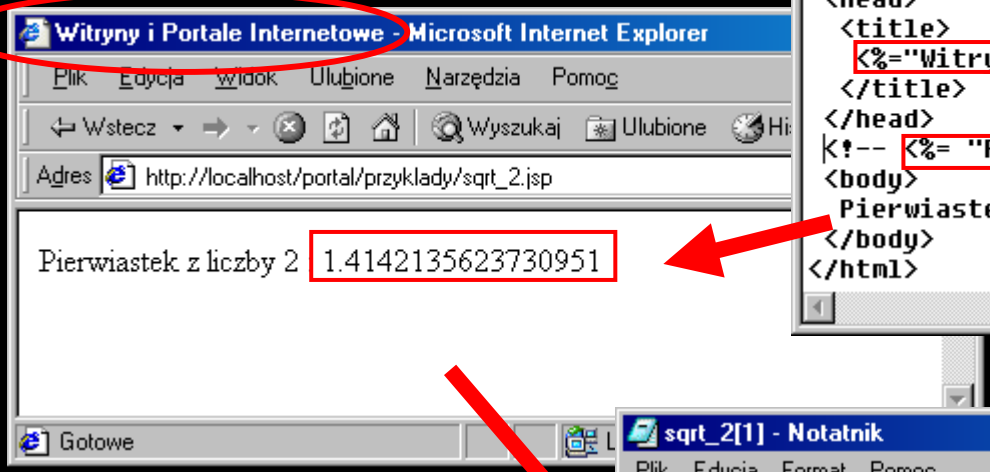
<%= . . . %>

Wyrażenia

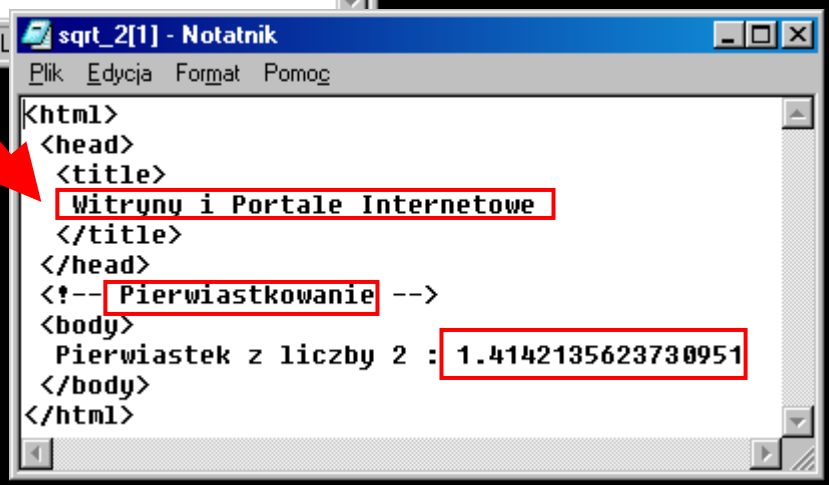
Strona JSP (serwer)



Przeglądarka (klient)



Źródło strony HTML (klient)



Deklaracje

Deklaracje zmiennych lub metod (poprawa czytelności strony JSP)

```
<%! deklaracja %>
```

Blok deklaracji nie może obsługiwać strumienia wyjściowego out.

<%! . . . %>

Deklaracje

<%! deklaracja %>

```
<%! Int a = 0; %>
<html>
  <body>
   <%! Int x,y,z; %>
    . . .
    . . .
  </body>
</html>
```

<%! ... %>

Deklaracje

<%! deklaracja %>

```
<html>
 <%!  Date data = new Date();
      Date getDate()
      {
       System.out.println("Wywołanie metody getDate()");
       return data;
      }
   %>
  <body>
  Data: <%= getDate() %>
  </body>
</html>
```

<%! . . . %>

Deklaracje

Deklaracja funkcji inicjalizacji `jspInit()` i niszczenia `jspDestroy()` strony JSP

init_and_destroy.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<html>

<head>
<title>Tworzenie i niszczenie JSP</title>
</head>

<body>
<%@ page language="java"%>

<%! public void jspInit() {
        System.out.println("Inicjalizacja JSP"); } %>


<%! public void jspDestroy() {
        System.out.println("Niszczenie JSP"); } %>

<p>Wynik działania należy zobaczyć w plikach log serwera Tomcat</p>

</body>
</html>
```

Strona JSP (serwer)

Konsola (serwer)

Start Tomcat

```
Starting service Tomcat-Standalone
Apache Tomcat/4.0.3
Starting service Tomcat-Apache
Apache Tomcat/4.0.3
Inicjalizacja JSP
Niszczenie JSP
```

Akcje

Akcje

Akcje standardowe to mechanizm ułatwiający wykonywanie najczęstszych operacji.

```
<jsp: ... >
```

Akcje umożliwiają:

- używanie komponentów JavaBean
- wysyłanie pluginów do klienta
- przekierowanie na inną stronę
- dołączanie statyczne i dynamiczne stron

<%jsp: ... %>

Akcje standardowe

<jsp: ... >

- <jsp:useBean>
- <jsp:setProperyt>
- <jsp:getProperty>
- <jsp:include>
- <jsp:forward>
- <jsp:param>
- <jsp:plugin>

Akcje mają składnie elementów XML (znacznik startowy, ciało i znacznik końcowy)

useBean

Akcja umożliwia dodanie do strony instancji komponentu JavaBean

Atrybuty:

id- nazwa instancji obiektu (za pomocą tej nazwy możliwe jest odwoływanie się do obiektu Java Beans

useBean

Atrybuty:

scope - czas życia obiektu JavaBean (zasięg)

class - nazwa klasyobiektu JavaBean

beanName - nazwa komponentu używana podczas jego tworzenia

type - typ zmiennej skryptowej

setProperty

Akcja umożliwia ustawienie właściwości komponentu JavaBean określonego akcją useBean

Atrybuty:

name - nazwa instancji obiektu

property - nazwa atrybutu, którego wartość ma zostać zmodyfikowana

setProperty

Użycie * jako nazwy atrybutu powoduje przeszukanie wszystkich parametrów żądania HTTP i dopasowanie parametrów odpowiadajcych nazwą i typem, parametrom komponentu JavaBean

param - nazwa parametru

value - przypisanie wartości dla wybranego atrybutu komponentu

getProperty

Akcja umożliwia pobranie właściwości komponentu JavaBean określonego akcją useBean

```
<jsp:getProperty ... >
```

Atrybuty

Atrybuty:

name- nazwa instancji obiektu, którego atrybut ma zostać odczytany

property - nazwa atrybutu, którego wartość ma zostać odczytana

param

Akcja pozwala na podawanie parametrów w postaci par nazwa i wartość parametru wewnątrz innych akcji.

Atrybuty:

name - nazwa parametru

value - wartość parametru

param

```
<jsp:params>
  <jsp:param name = "nazwa parametru"
             value = "wartość parametru" >
</jsp:params>
```

Akcję `<jsp:param>` stosuje się przy akcjach:

- `<jsp:include>`
- `<jsp:forward>`
- `<jsp:plugin>`

include

Akcja umożliwia dołączenie do bieżącej strony JSP zasobów statycznych i dynamicznych

Atrybuty:

page - względny adres zasobu

flush - wartość logiczna (true/false) decydujca o obróżnianiu bufora

include

W przypadku buforowania strumienia następuje jego wyczyszczenie (zawartość strony zostaje zignorowana)

Za pomocą akcji param umieszczonego wewnątrz znacznika `<jsp:include> ... </jsp:include>` można dokonać przekazania parametrów kierowanych w zapytaniu do wstawianego zasobu

forward

Akcja umożliwia przejście do innego zasobu w czasie wykonywania programu

```
<jsp:forward ... >
```

Atrybuty

Wystąpienie znacznika akcji

```
<jsp:forward page="adres_zasobu">
```

powoduje zakończenie wykonywania kodu bieżącej strony i przejście do podanego zasobu

forward

Atrybuty:

page - adres zasobu

(statyczny, serwlet lub strona JSP) do którego zostanie przekazanie sterowanie

Za pomocą akcji param umieszczonego wewnątrz znacznika `<jsp:forward> ... </jsp:forward>` można dokonać przekazania parametrów

plugin

Akcja umożliwiająca wysłanie appletu javy do przeglądarki klienta

Znacznik **<jsp:plugin>** zostaje zamieniony znacznikiem **<object>** lub **<embed>**, w zależności od typu przeglądarki

(następuje wysłanie odpowiednich ustawień do strumienia wyjściowego)

plugin

Atrybuty:

type - opreślenie typu pluginu (np. aplet)

code - nazwa klasy, która zostaje wykonana przez plugin

codebase - lokalizacja względnego adresu URL kodu klasy

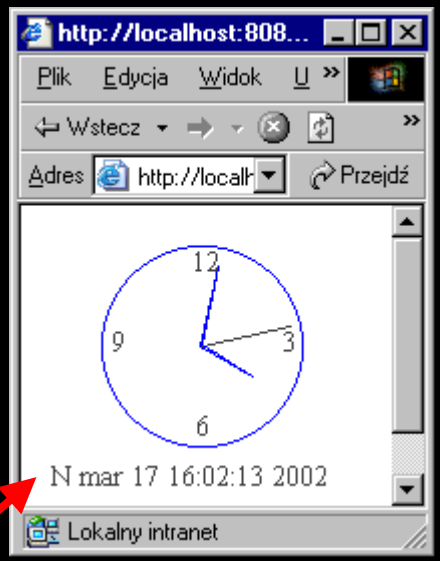
plugin

Strona JSP (serwer)

plugin.jsp - Notatnik

Plik Edycja Format Pomoc

```
<jsp:plugin type="applet"
     code="Clock2.class"
     codebase="/przyklady/applet"
     jreversion="1.2" width="160" height="150">

    <jsp:fallback>
        Znaczniki OBJECT i EMBED nie są obsługiwane przez przeglądarkę.
    </jsp:fallback>

</jsp:plugin>
```



Źródło strony (klient)

plugin[1] - Notatnik

Plik Edycja Format Pomoc

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" width="160"  height="150"
codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-1_2_2-win.cab#Version=1,2,2,0">
<PARAM name="java_code" value="Clock2.class">
<PARAM name="java_codebase" value="/przyklady/applet">
<PARAM name="type" value="application/x-java-applet;version=1.2">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.2"  width="160"  height="150"
pluginspage="http://java.sun.com/products/plugin/" java_code="Clock2.class"
java_codebase="/przyklady/applet" >
<NOEMBED></COMMENT>Znaczniki OBJECT i EMBED nie są obsługiwane przez przeglądarkę.
</NOEMBED></EMBED>
</OBJECT>
```

<jsp:root>

JSP jako znaczniki XML

- walidacja dokumentów JSP
- możliwość manipulacji dokumentami JSP za pomocą standardowych narzędzi do XML
- transformacja XML za pomocą XSLT (generacja dokumentów JSP)
- możliwość przechowywania stron JSP w bazach danych XML

JSP jako znaczniki XML

```
<% . . . %>
```

Wszelkie zasady składni znaczników JSP są zgodne z XML

```
<jsp:. . . >. . .</jsp:. . .>
```

- jsp:root
- jsp:text

- jsp:scriptlet
- jsp:directive
- jsp:expression
- jsp:declaration

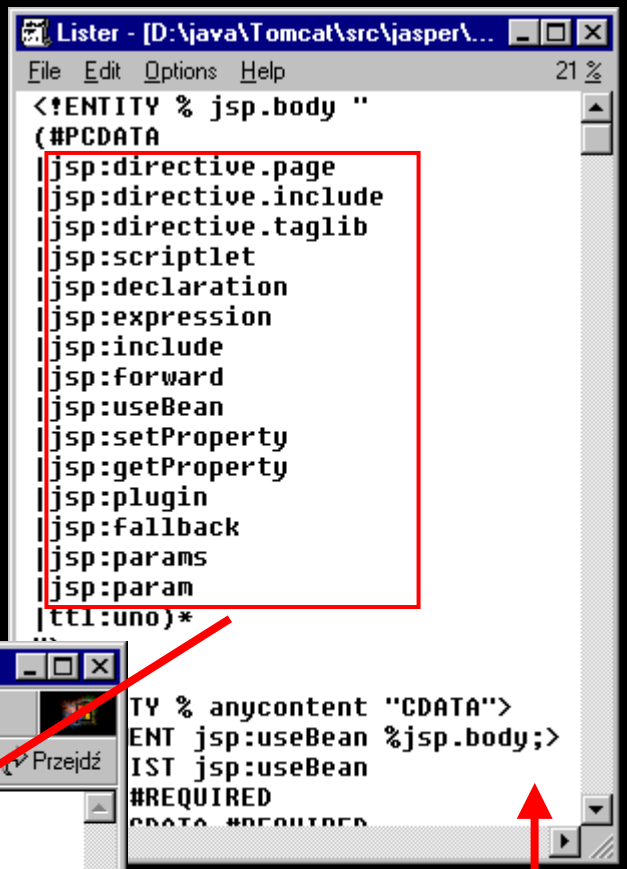
Akcje

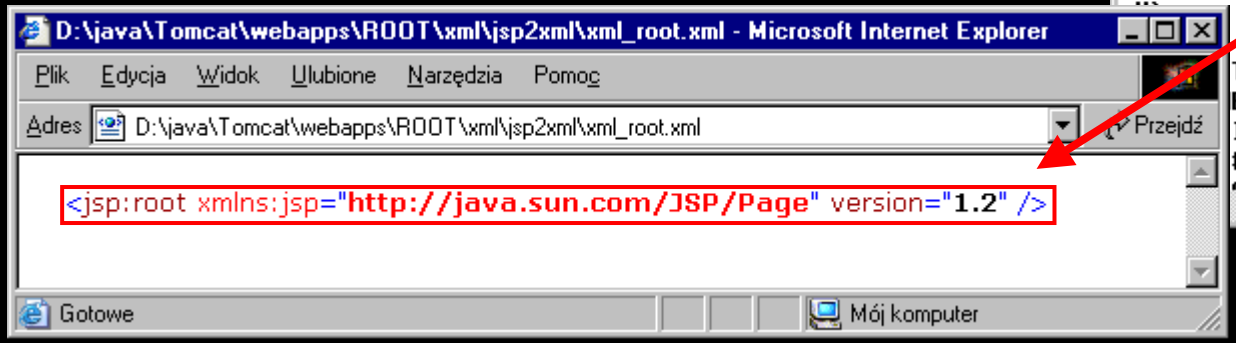
- jsp:useBean
- jsp:setProperty
- jsp:getProperty
- jsp:include
- jsp:forward
- jsp:param
- jsp:plugin

Element główny jsp:root

```
<jsp:root>
...
</jsp:root>
```

Strony XML zawsze zawierają tylko jeden element główny (root)

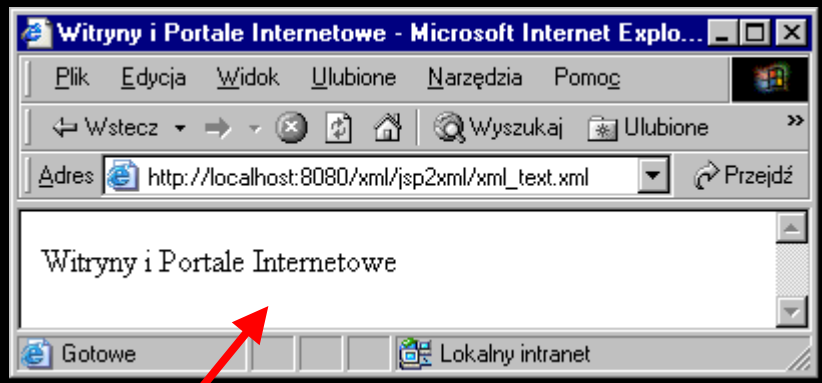


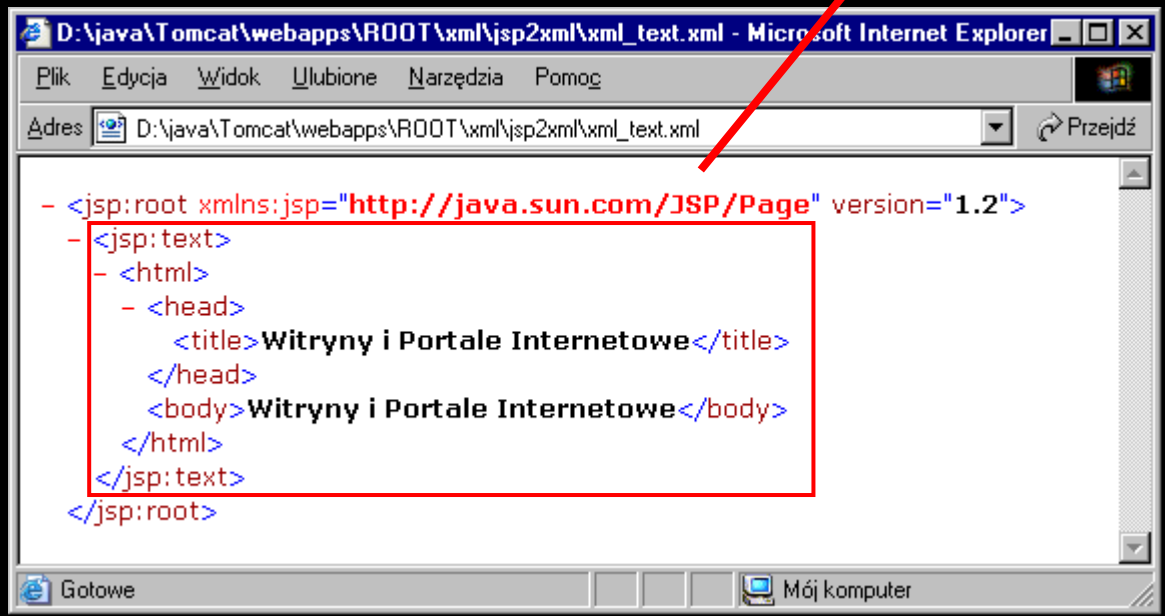


TOMCAT_HOME\src\jasper\src\share\org\apache\jasper\resource\jsp12.dtd

Elementy tekstowe jsp:text

```
<jsp:text>
...
</jsp:text>
```

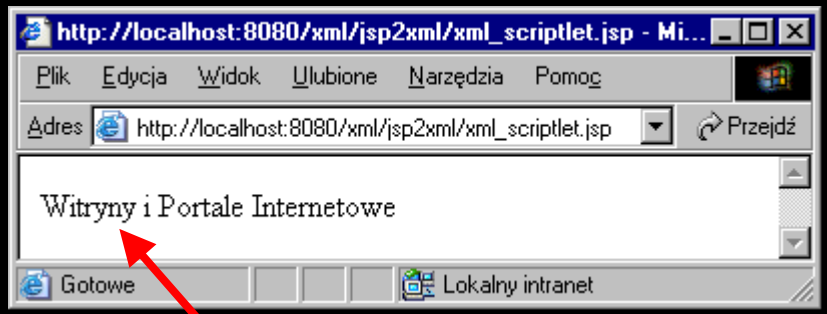


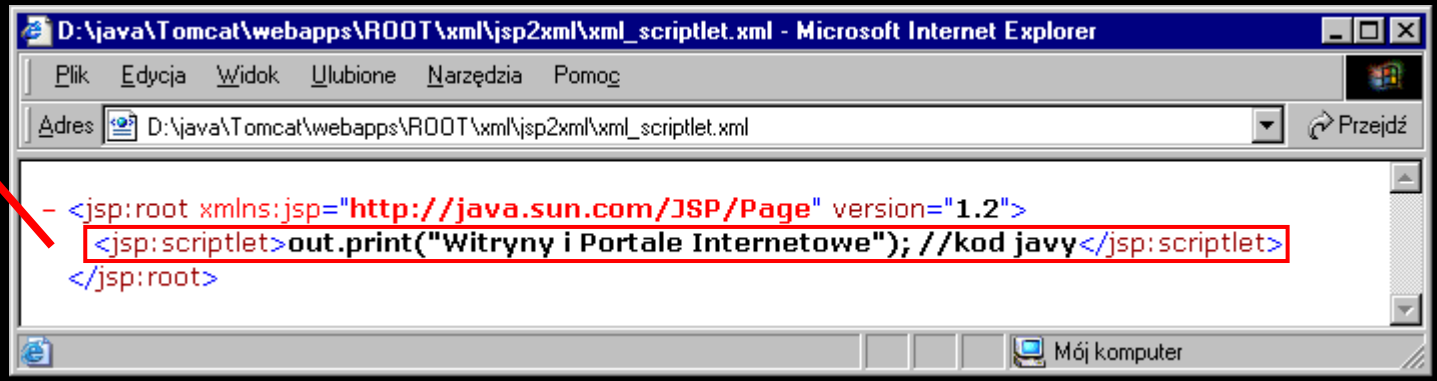


Scriptlety jsp:scriptlet

<% ... %>

<jsp:scriptlet>
...
</jsp:scriplet>



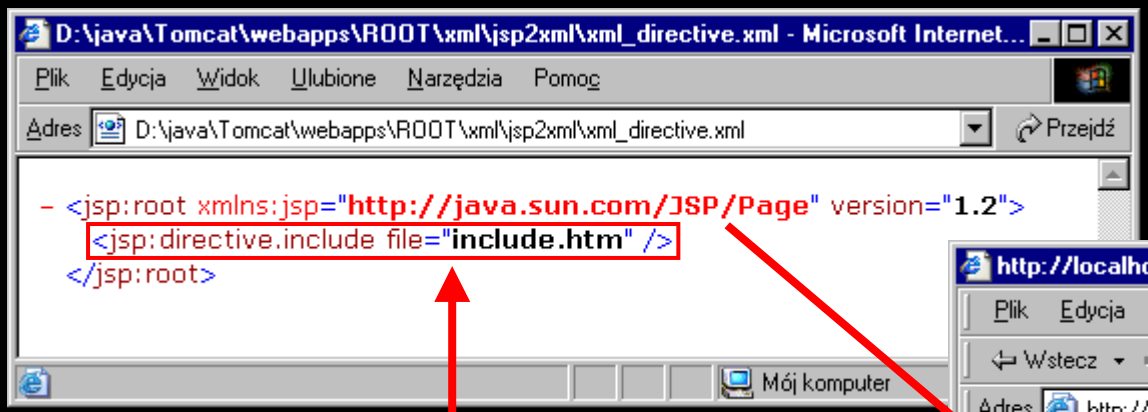


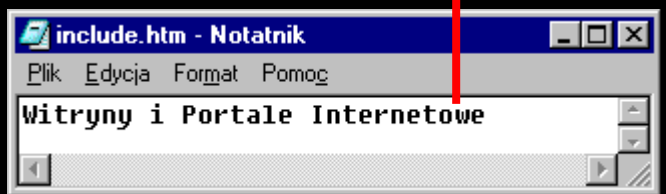
Dyrektywy jsp:directive

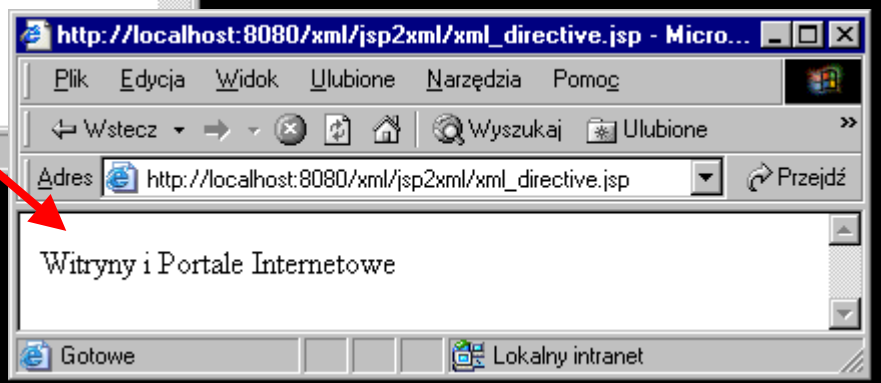
```
<%@ ... %>
```

```
<jsp:directive.nazwa atrybuty... />
```

Nazwa dyrektywy: *page, include lub taglibs*

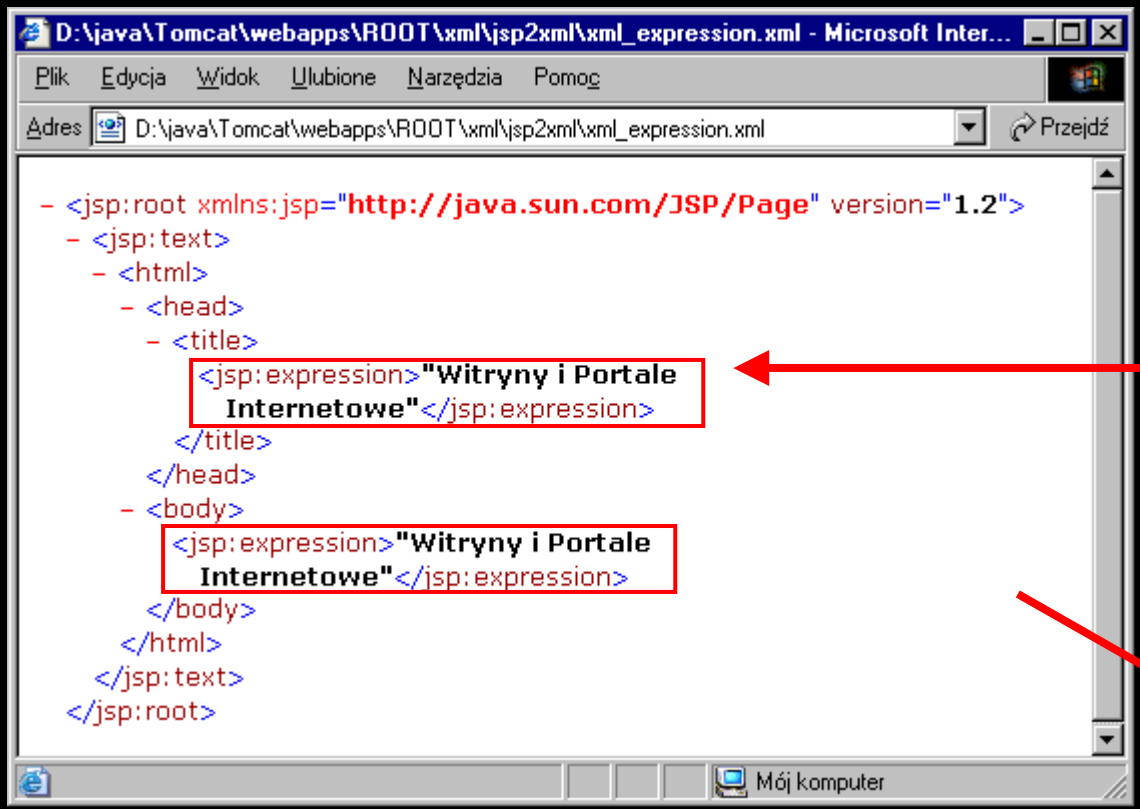






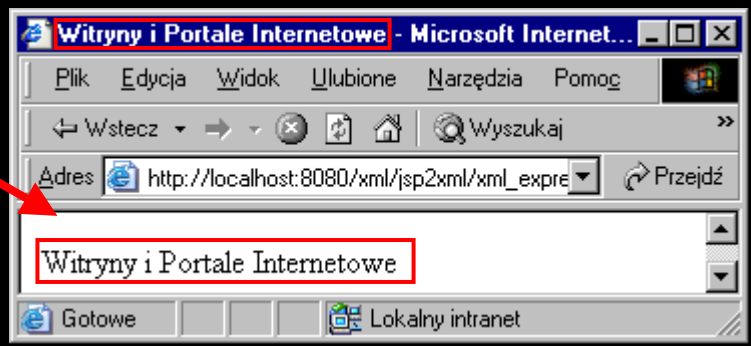
Wyrażenia jsp:expression

```
<jsp:expression>...</jsp:expression>
```



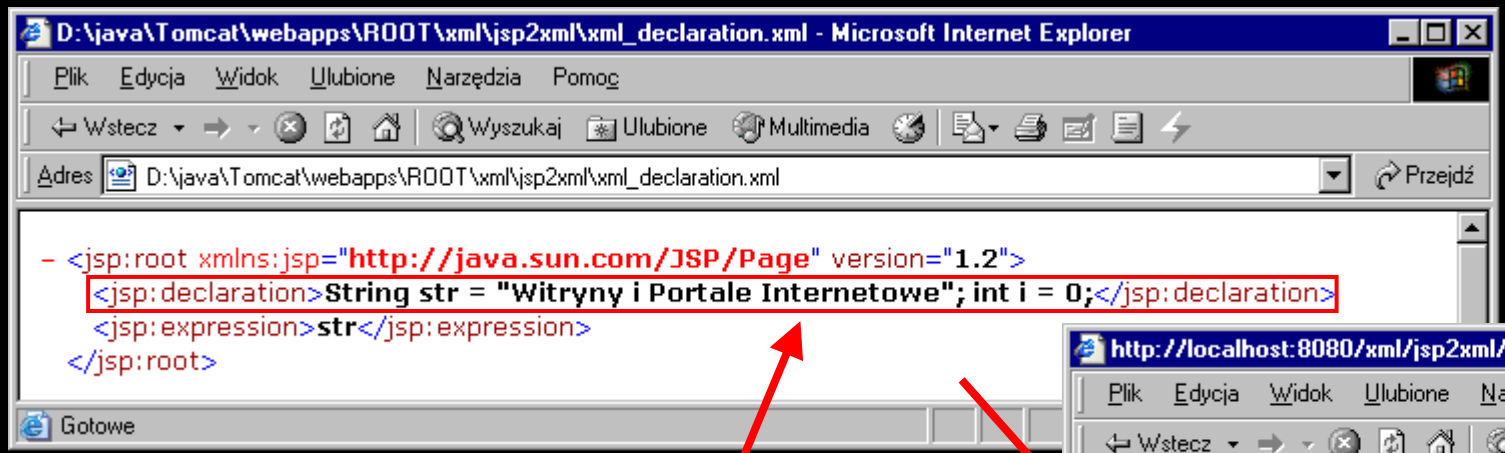

```
<%=  ...  %>
```

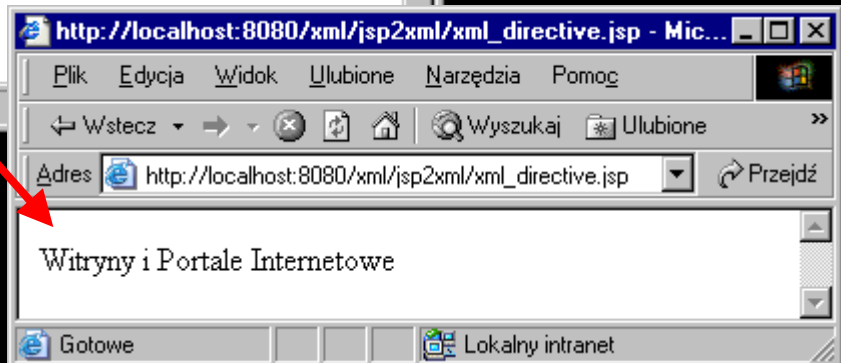
W przypadku łańcuchów należy stosować znaki " "



Deklaracje jsp:declaration

```
<jsp:declaration>
   ...
</jsp:declaration>
```





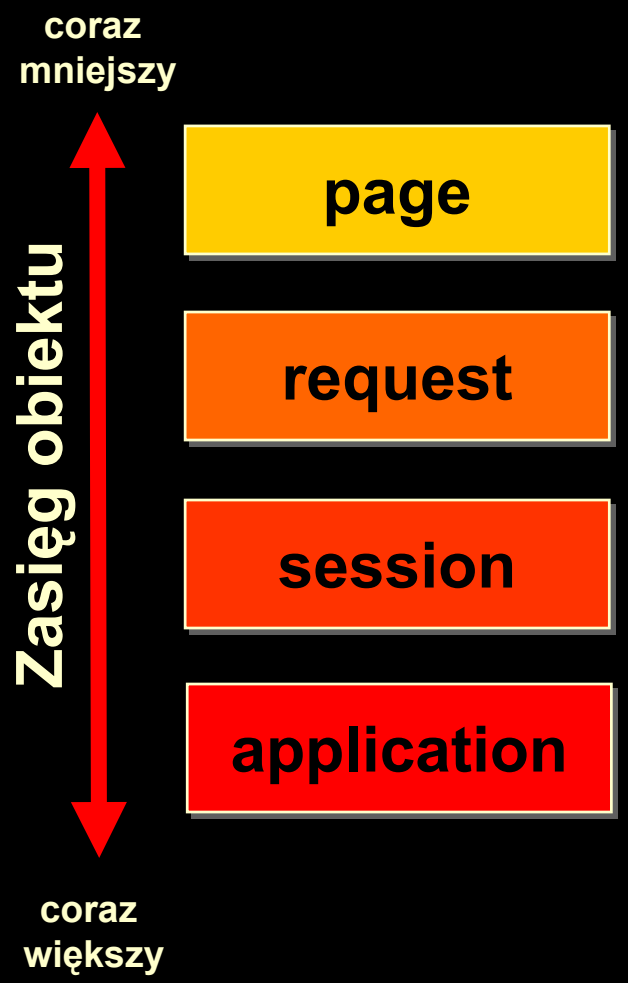
W znaczniku można umieszczać jednocześnie wiele deklaracji oddzielając je znakiem średnika

Obiekty

Tworzenie obiektów

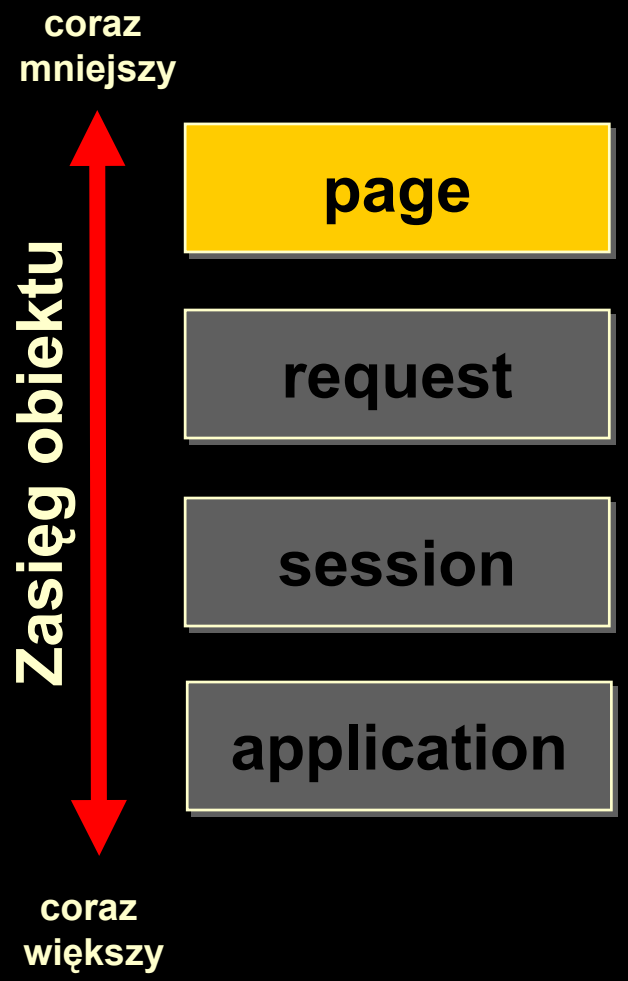
- tworzenie w sposób niejawny (obiekt może powstać w wyniku wywołania dyrektywy JSP)
- tworzenie z użyciem akcji JSP
- tworzenie jawne poprzez bezpośrednie zdefiniowanie obiektu w kodzie skryptu JSP

Zasięg obiektów (scope)



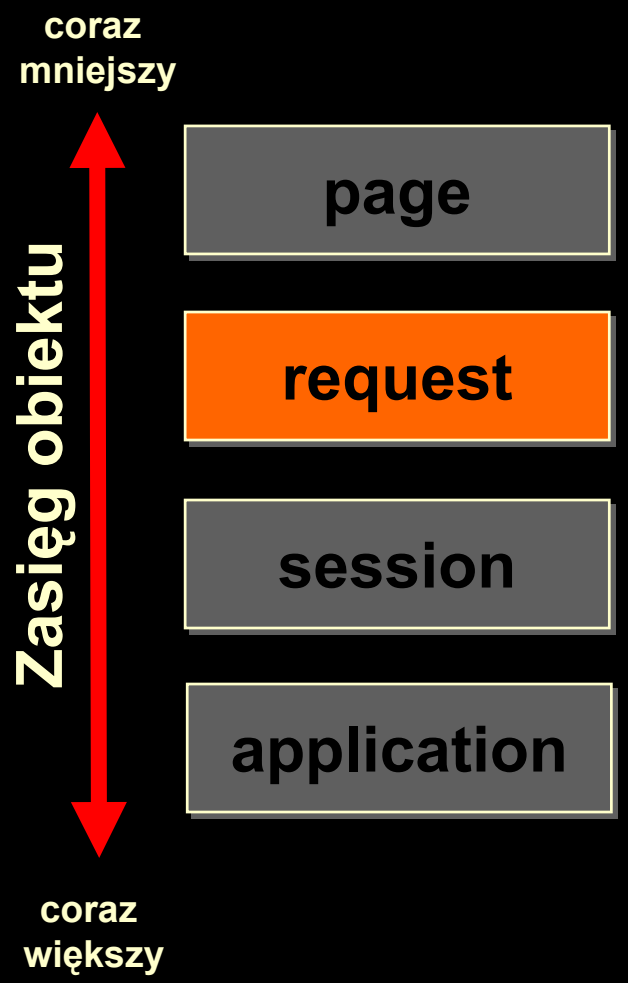
Zasięg widoczności obiektów określa, kiedy można utworzyć odniesienie (referencję) do obiektu

Zasięg obiektów (page)



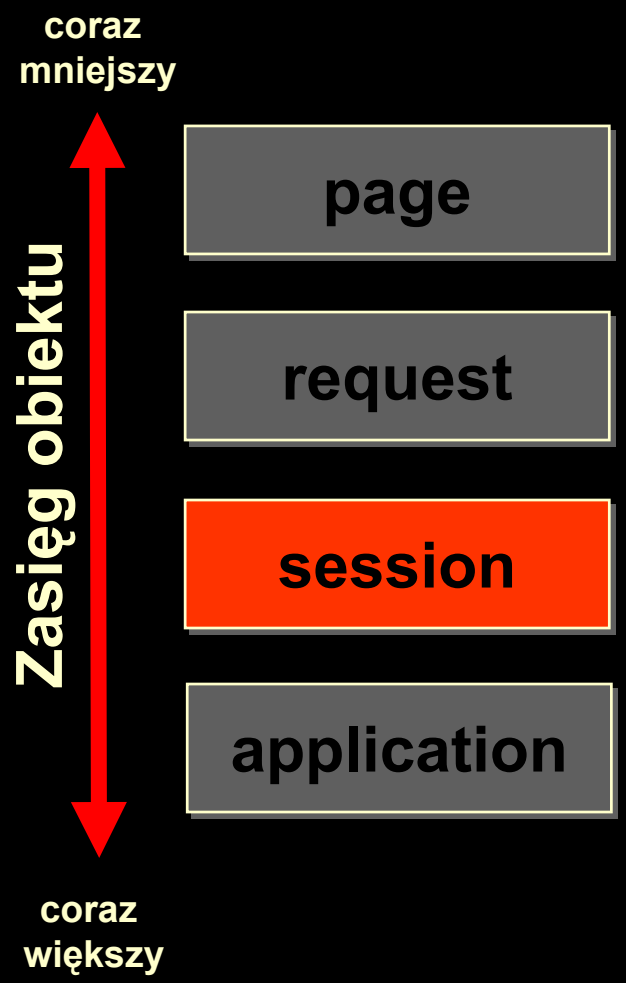
page - zasięg widoczności obiektów w obrębie strony tworzącej dany obiekt

Zasięg obiektów (request)



request - zasięg widoczności obiektów ogranicza się do stron przetwarzających żądanie, w którym zostały utworzone określone obiekty (przekazywanie żądania na inną stronę umożliwia korzystanie z takich obiektów)

Zasięg obiektów (session)

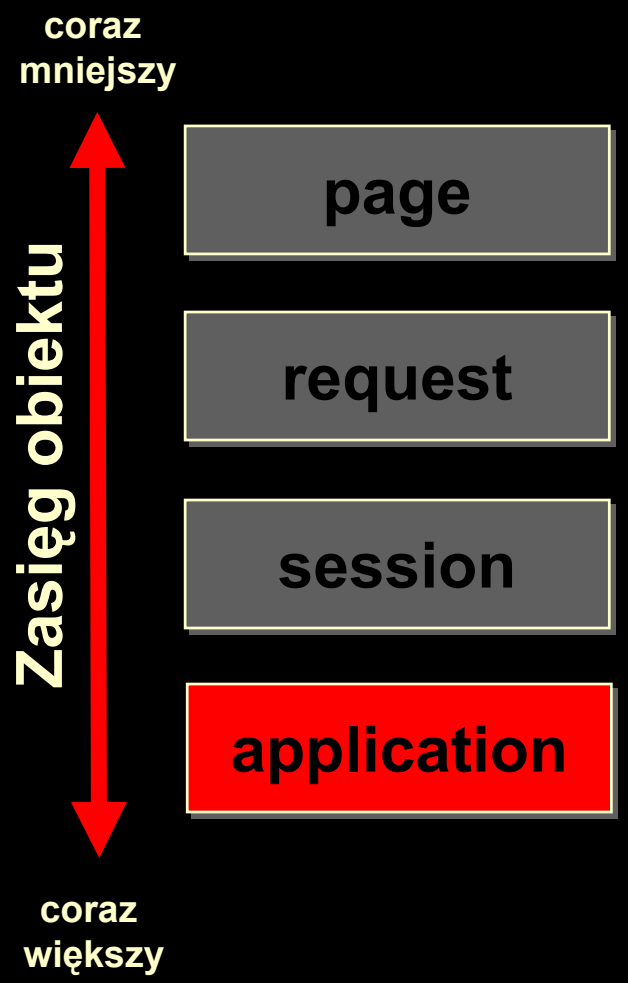


session - zasięg widoczności obiektów w obrębie sesji, która je utworzyła.

Deklarowanie stron należących do sesji dokonuje się dyrektywą page.

Należy pamiętać o zwalnianiu instancji obiektów przy kończeniu sesji.

Zasięg obiektów (application)



application - zasięg widoczności obiektów w całej aplikacji WWW, która je utworzyła

Obiekty niejawne

Dostęp do obiektów bez ich wcześniejszej deklaracji w kodzie JSP.

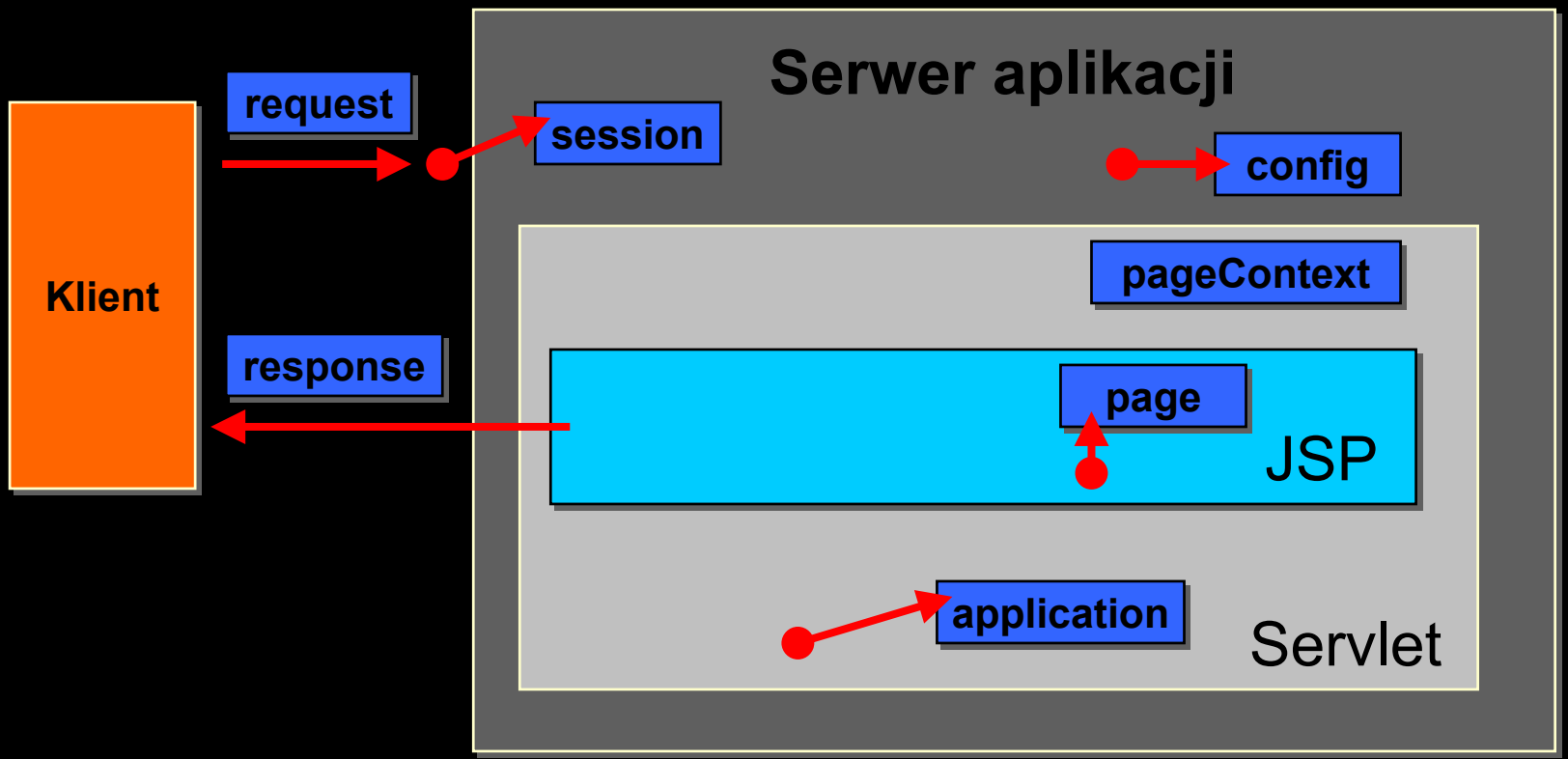
- request, response
- pageContext
- session
- application
- out,
- config, page, exception

Klasy obsługi obiektów

Wszystkie obiekty mają typ, który jest zdefiniowany klasą lub interfejsem

request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
pageContent	javax.servlet.jsp.pageContext
application	javax.servlet.http.ServletContext
config	javax.servlet.http.ServletConfig
page	java.lang.Object
exception	java.lang.Throwable

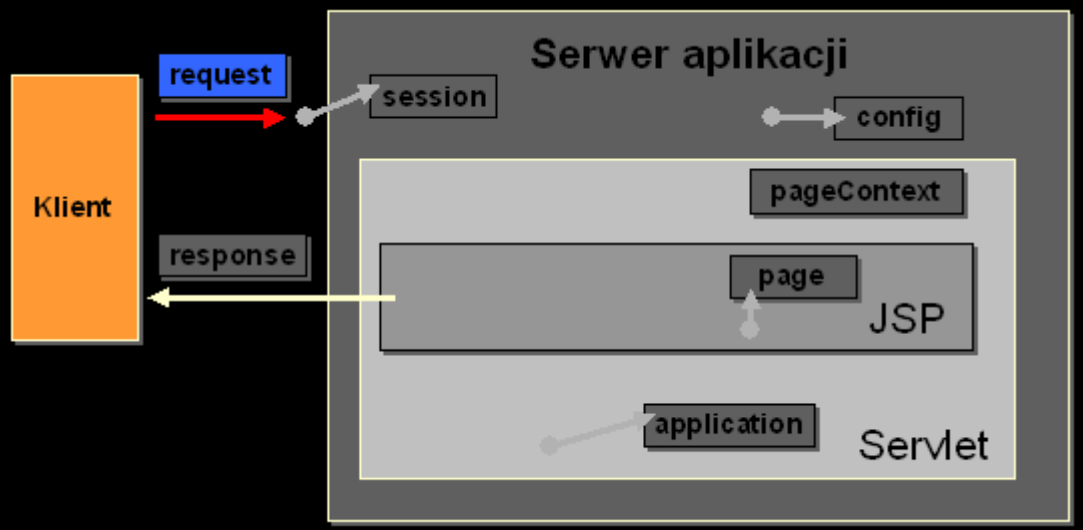
Zasięg obiektów (scope)



Zasięg definiuje obszar działania danego obiektu

Obiekt request

Obiekt request zawiera informacje umieszczone w żądaniu klienta.



Informacje te są przesyłane w nagłówku HTTP oraz przez zawartość żądania

Obiekt request

Parametry żądania - łańcuch tekstowy przesyłany razem z zapytaniem

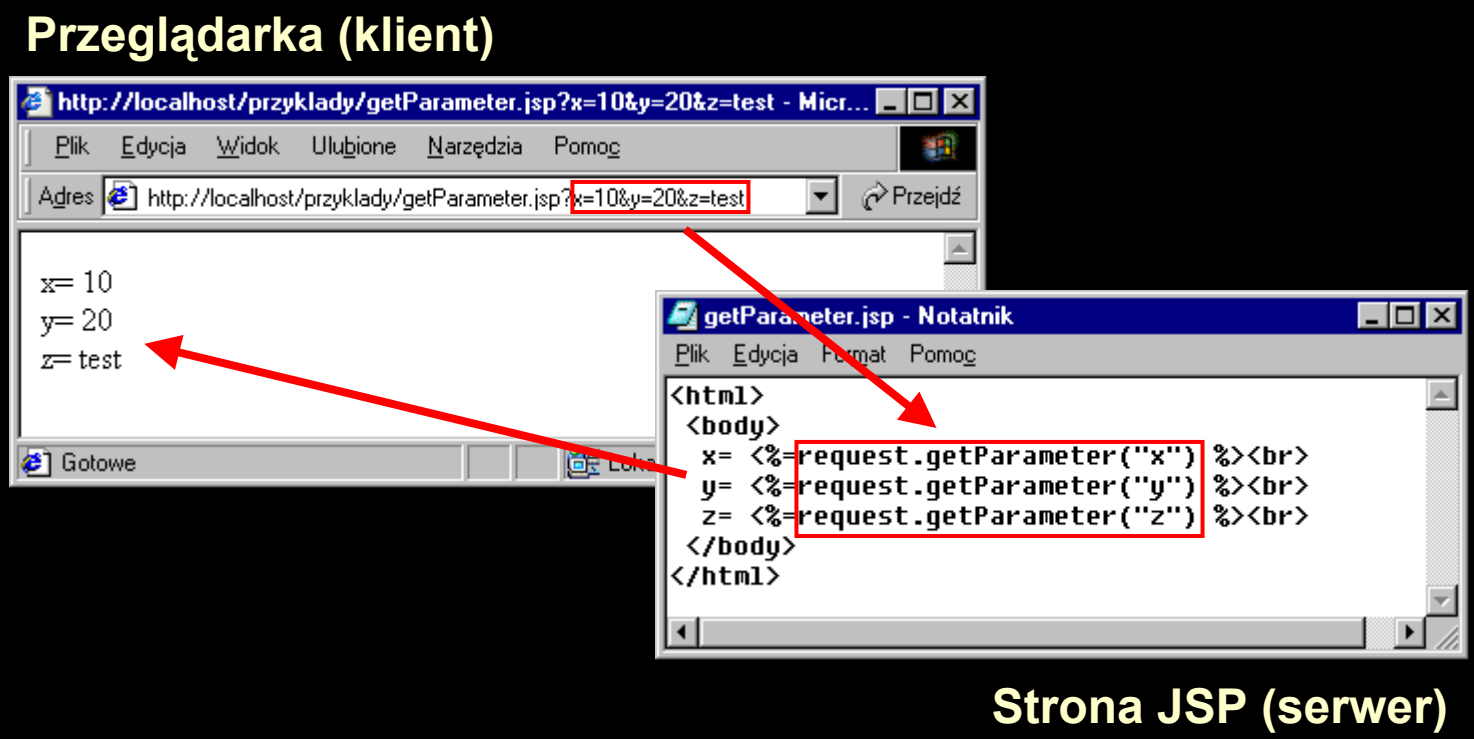
- getParameter
- getParameterNames
- getParameterValues

Przesłanie parametrów do żądania:

- ciąg znaków zapytania URL
- dane z formularza przesłanego metodą POST

Request.getParameter

getParameter - pobranie parametru z łańcucha URL



Obiekt request

Atrybuty - łańcuch tekstowy przesyłany razem z zapytaniem

- getAttribute
- getAttributeNames
- setAttribute

Pojedynczemu elementowi z tablicy odpowiada tylko jedna wartość

Obiekt request

Nagłówki HTTP - zestaw metod do odczytu nagłówków

- getHeader
- getHeaders
- getHeaderNames
- getIniHeader
- getDateHeader

request.getHeaderNames()
request.getHeaders()

Strona JSP (serwer)

request.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=windows-1250" %>

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


<h1>Obiekt request</h1>



<table border="2">
<tr><td><b>Atrybut</b></td><td><b>Wartość</b></td></tr>
<%
  String str;

  for(Enumeration e = request.getHeaderNames(); e.hasMoreElemer

    str = (String) e.nextElement();

     out.println("<tr><td>"+str+"</td><td>");

     for(Enumeration e2 = request.getHeaders(str); e2.hasMoreEl
       out.println((String) e2.nextElement());

     }

     out.println("</td>");
  }

%>
</table>
```

http://localhost/przyklady/request.jsp - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc Wstecz

Adres http://localhost/przyklady/request.jsp Przejdź

Obiekt request

Atrybut	Wartość
accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, application/vnd.ms-powerpoint, application/vnd.ms-excel, */*
referer	http://localhost/przyklady/
accept-language	pl
accept-encoding	gzip, deflate
user-agent	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
host	localhost
connection	Keep-Alive
cookie	JSESSIONID=66372FBEA3671AB1E14FDD1D71AE9A95

Gotowe Lokalny intranet

Przeglądarka (klient)

Obiekt request

Adres URI (Universal Resource Identifier)

- część adresu URL określająca ścieżkę do zasobu.

```
http://localhost:8080/portal/przyklady/witaj.jsp?test=ok
```

Adres URI składa się z trzech części:

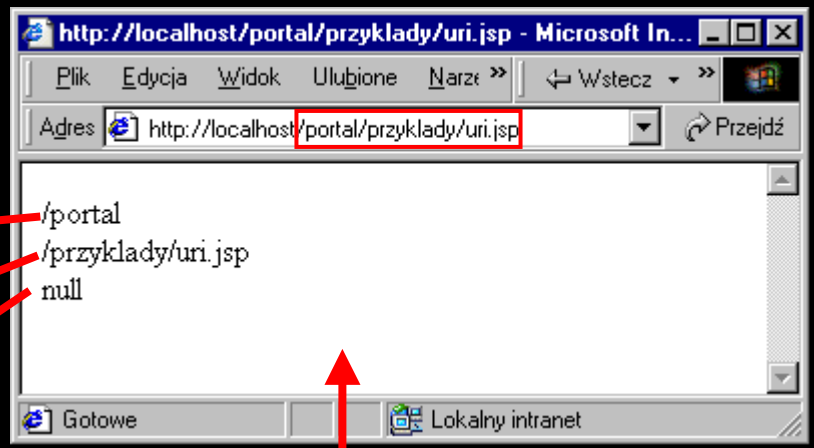
```
AdresURI = contextPath + ServletPath + pathInfo
```

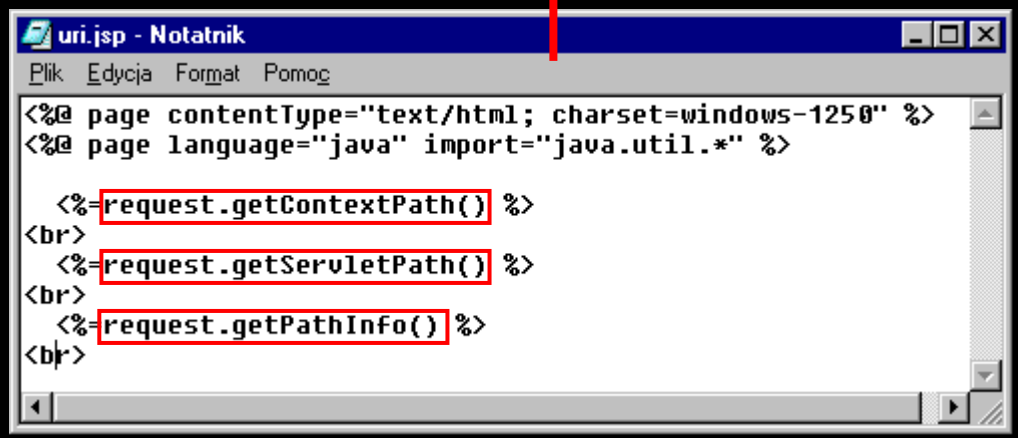
- getContextPath
- getServletPath
- getPathInfo

Obiekt request

AdresURI
=
contextPath
+
ServletPath
+
pathInfo

Przeglądarka (klient)





Strona JSP (serwer)

Obiekt request

- getScheme - nazwa protokołu (HTTP)
- getServerName - nazwa domenty serwera
- getServerPort - numer portu
- getRealPath - rzeczywista lokalizacja zasobu na serwerze
- getPathTranslated - przekształcenie części PathInfo adresu URI na ścieżkę lokalną
- getRemoteAddr - adres IP
- getRemoteHost - nazwa domeny Klienta

Obiekt request

Strona JSP (serwer)

path1.jsp - Notatnik

Plik Edycja Format Pomoc

```
<tr>
<td>request.getScheme()</td>
<td><%=request.getScheme() %></td>
</tr>

<tr>
<td>request.getServerName()</td>
<td><%=request.getServerName() %></td>
</tr>

<tr>
<td>request.getServerPort()</td>
<td><%=request.getServerPort() %></td>
</tr>

<tr>
<td>request.getRealPath(String adresURI)</td>
<td><%=request.getRealPath(request.getServletPath()) %></td>
</tr>

<tr>
<td>request.getPathTranslated(String PathInfo)</td>
<td><%=request.getPathTranslated() %></td>
</tr>

<tr>
<td>request.getRemoteAddr()</td>
<td><%=request.getRemoteAddr() %></td>
</tr>
```

http://localhost/przyklady/path1.jsp - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

Adres http://localhost/przyklady/path1.jsp Przejdź

request.getScheme()	http
request.getServerName()	localhost
request.getServerPort()	80
request.getRealPath(String adresURI)	C:\java\tomcat\webapps\ROOT\przyklady\path1.jsp
request.getPathTranslated(String PathInfo)	null
request.getRemoteAddr()	127.0.0.1
request.getRemoteHost()	krypton

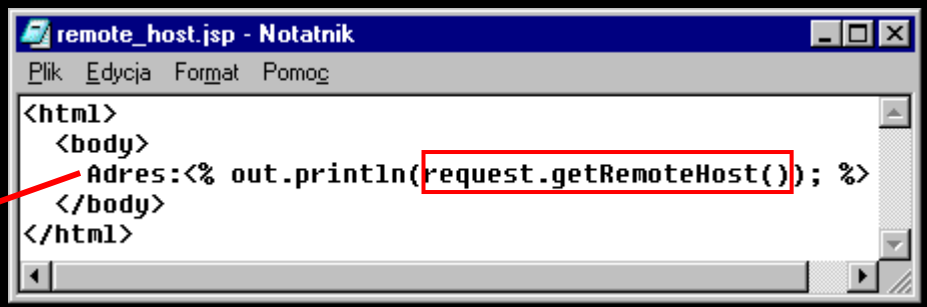
Gotowe Lokalny intranet

Przeglądarka (klient)

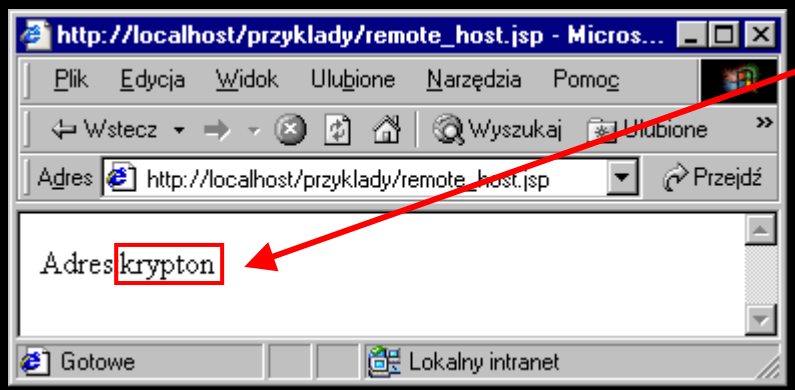
request.getRemoteHost

getRemoteHost - nazwa domeny Klienta

```
<html>
  <body>
    Adres:<% out.println(request.getRemoteHost()); %>
  </body>
</html>
```



Strona JSP (serwer)



Przeglądarka (klient)

Obiekt request

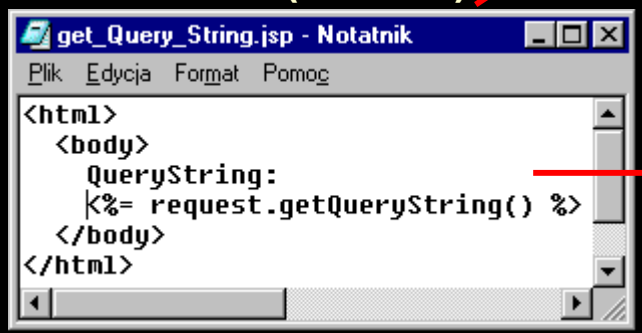
- getQueryString - pobranie łańcucha tekstowego z adresu URL

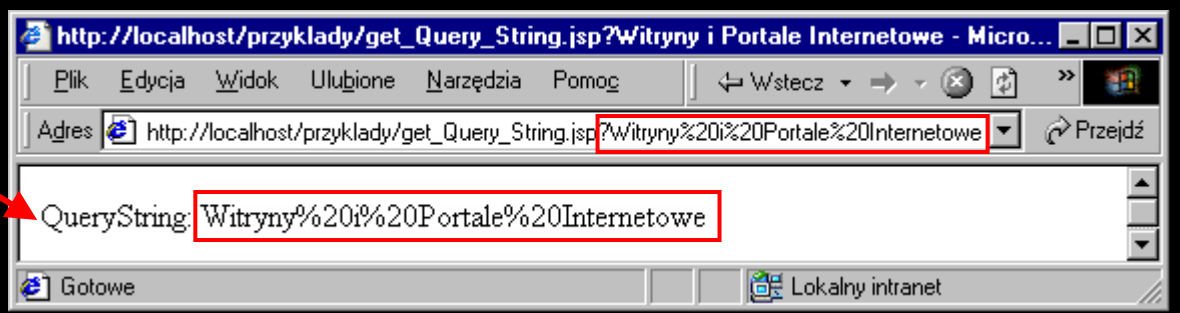
```
http://localhost/przyklady/test?x=20&y=20
```

Następuje pobranie całego łańcucha po znaku zapytania

request.getQueryString

Strona JSP (serwer)



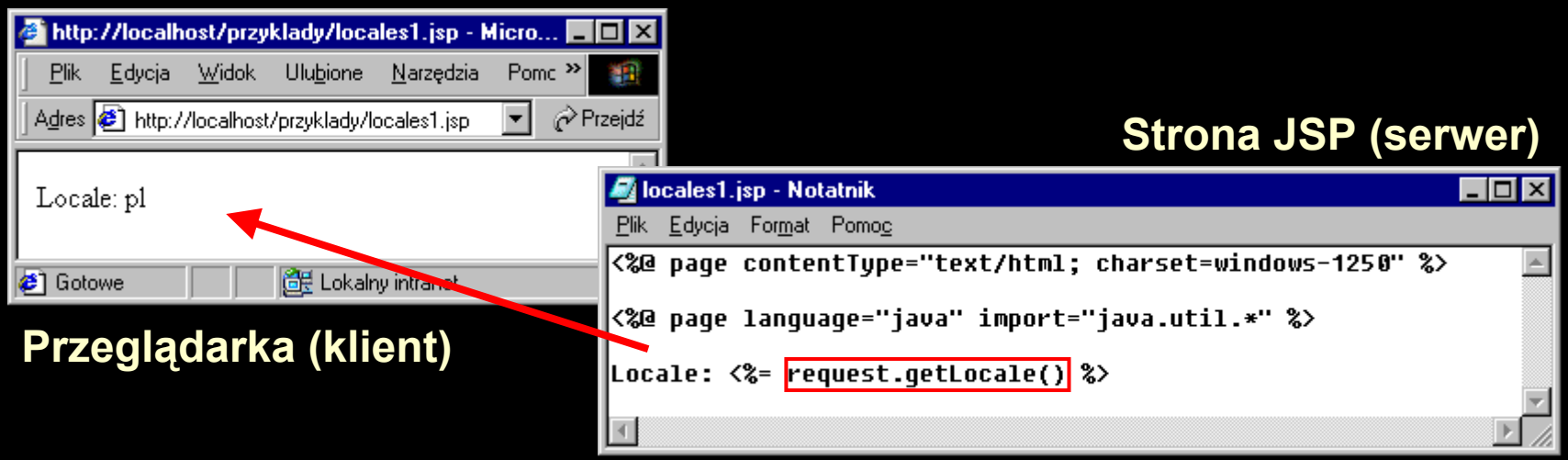


Następuje zamiana znaków specjalnych (spacje) na ich kody (%20)

Przeglądarka (klient)

Obiekt request

- getLocale, getLocales - odczytywanie ustawień lokalnych klienta

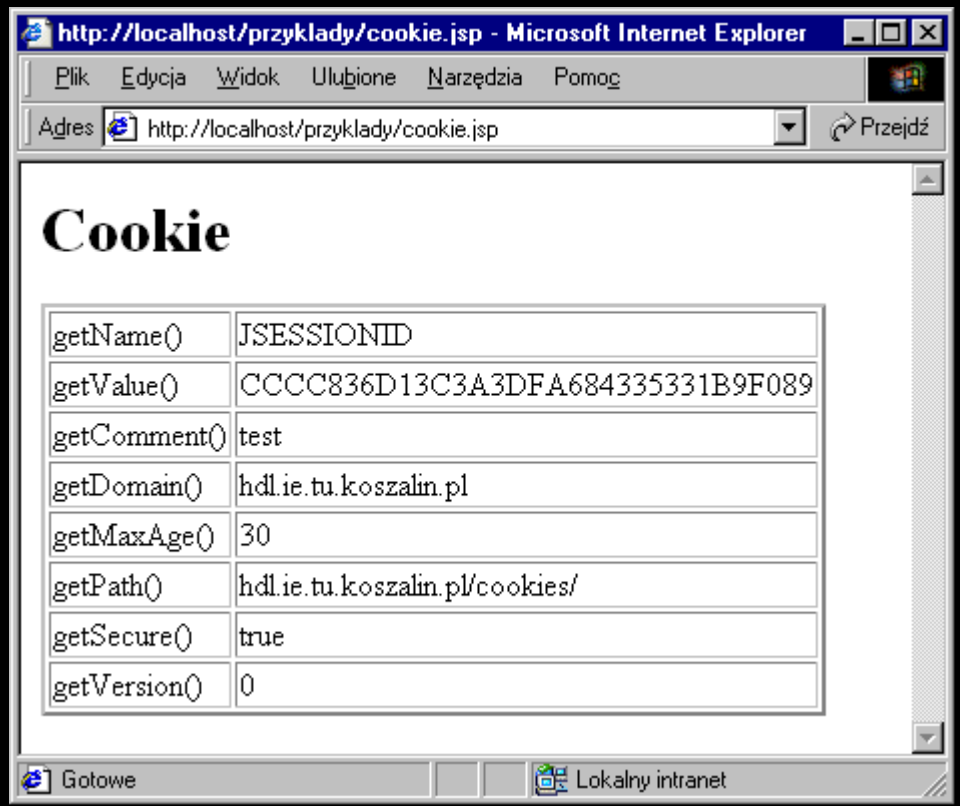


Dokonując sprawdzenia getLocale w łatwy sposób można realizować lokalizowane wersje stron WWW

request.getCookies

- getCookies - pobieranie ciasteczek

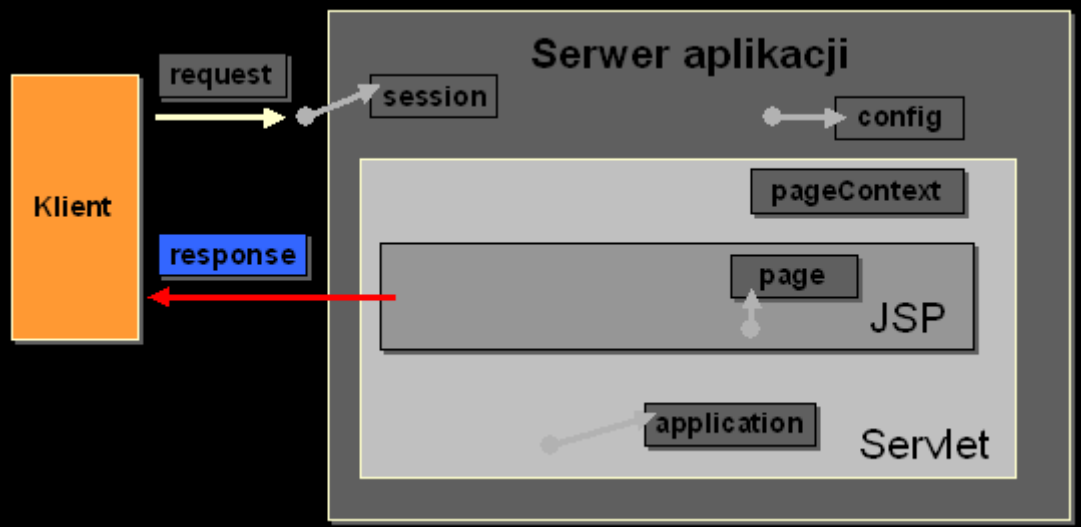
cookie.
- getName
- getValue
- getComment
- getDomain
- getMaxAge
- getPath
- getSecure
- getVersion



Obiekt response

Obiekt response umożliwia:

- manipulowanie zawartością nagłówka HTML
- zwracanie danych do przeglądarki klienta
- buforowanie



Interfejs opisujący strumień wyjściowy kontenera JSP

Obiekt response

Buforowanie - poprawa wydajności strony poprzez buforowanie strumienia wyjściowego

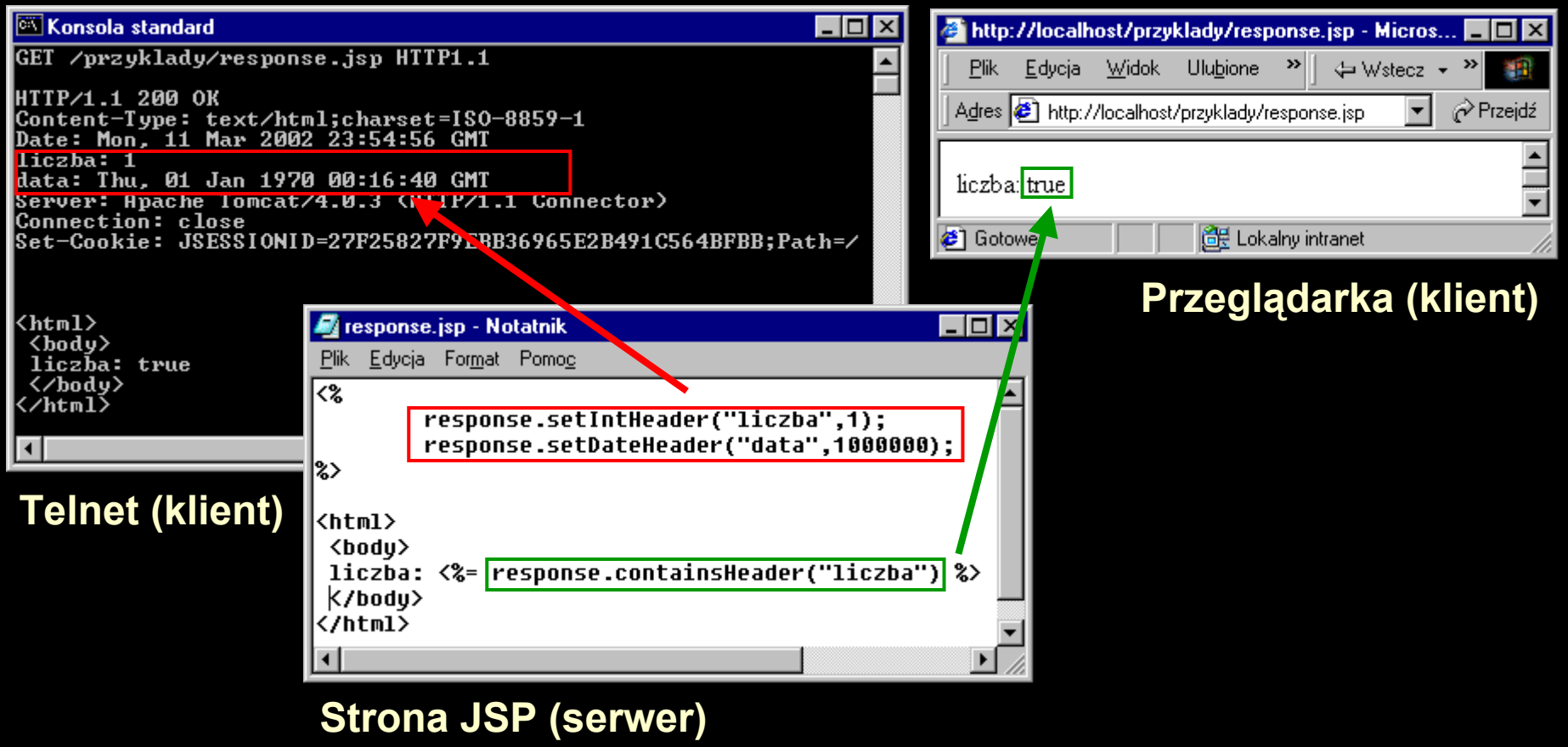
- getBufferSize - pobranie rozmiaru bufora
- setBufferSize - ustawienie rozmiaru bufora
- isCommitted - informacja o tym, czy już wysłano jakieś dane do klienta
- reset - czyszczenie zawartości bufora

Obiekt response

Nagłówki - możliwość tworzenia i ustawiania nagłówków odpowiedzi HTTP

- setHeader - ustawienie nagłówka
- addHeader - dodanie własnego nagłówka
- setIntHeader - ustaw nagłówek numeryczny
- setDateHeader - ustaw nagłówek daty
- addIntHeader - dodaj własny nagłówek numeryczny
- addDateHeader - dodaj własny nagłówek daty

Obiekt response



containsHeader - sprawdzenie istnienia nagłówka

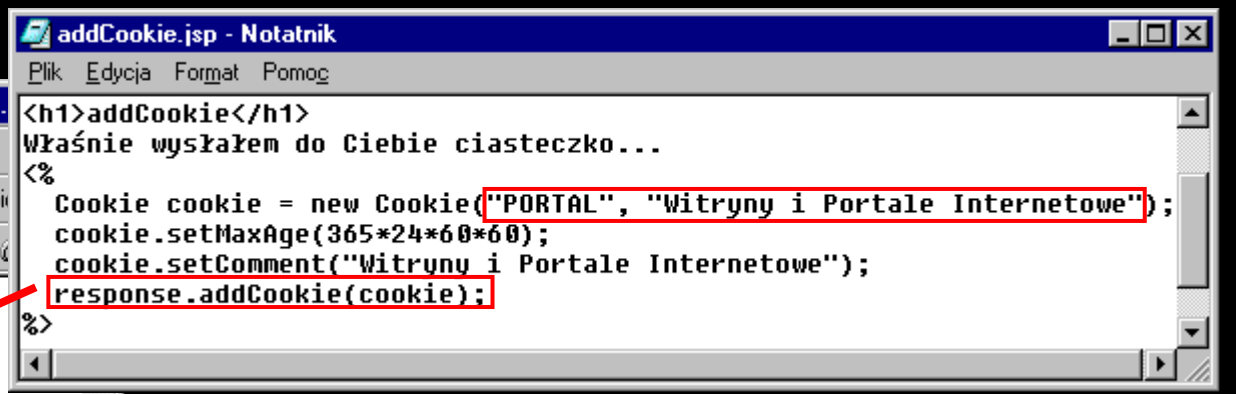
response.addCookie

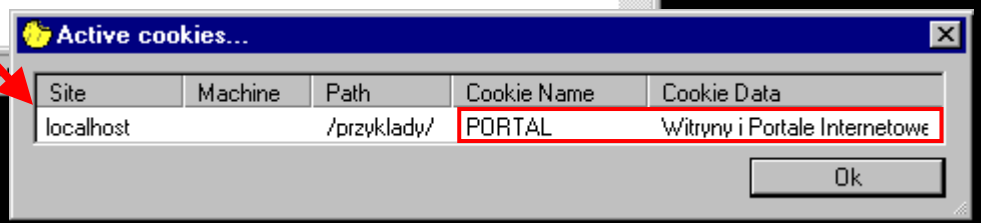
Cookies - tworzenie i wysyłanie ciasteczek do klienta

- addCookie - utworzenie i dołączenie nowego ciasteczka do odpowiedzi serwera

Strona JSP (serwer)

Przeglądarka (klient)

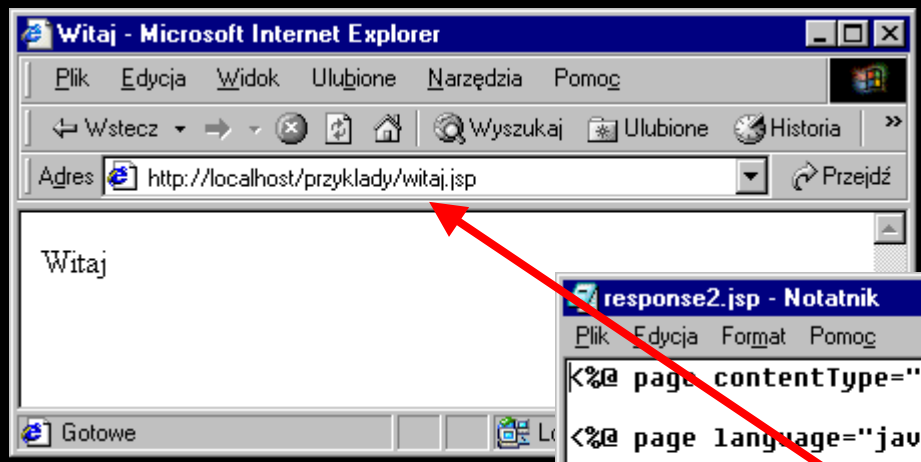




Podgląd ciasteczek
AnalogX CookieWall (klient)
www.analogx.com

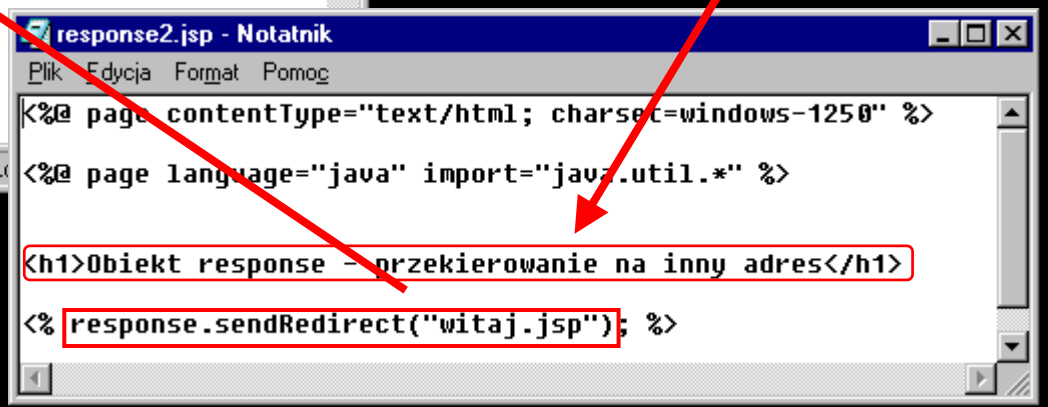
Obiekt response

- sendRedirect - przekierowanie strony na stronę o padanym adresie URL



Przeglądarka (klient)

Zawartość strony nie zostaje przesłana do klienta

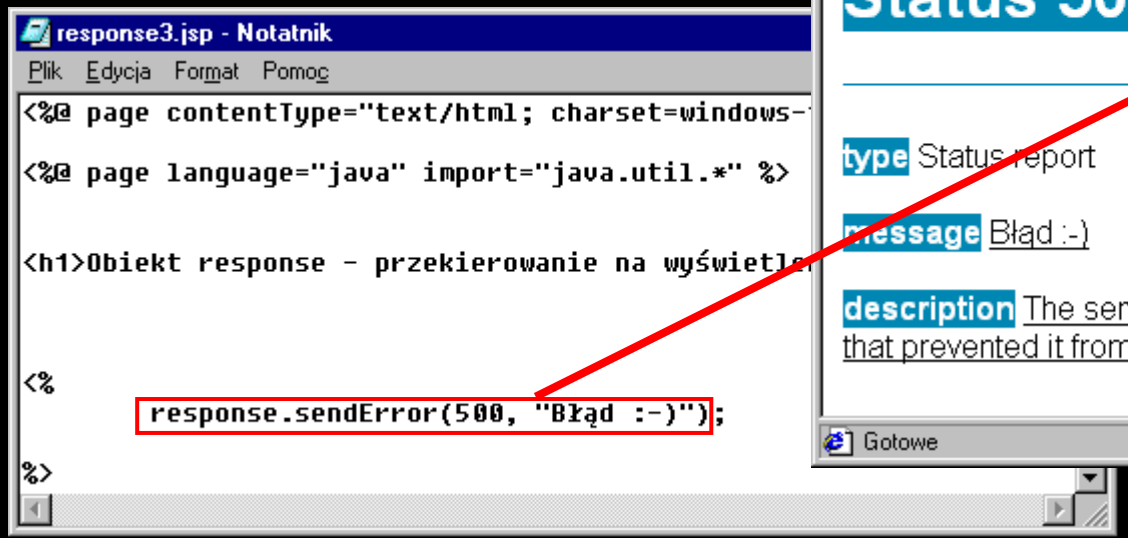


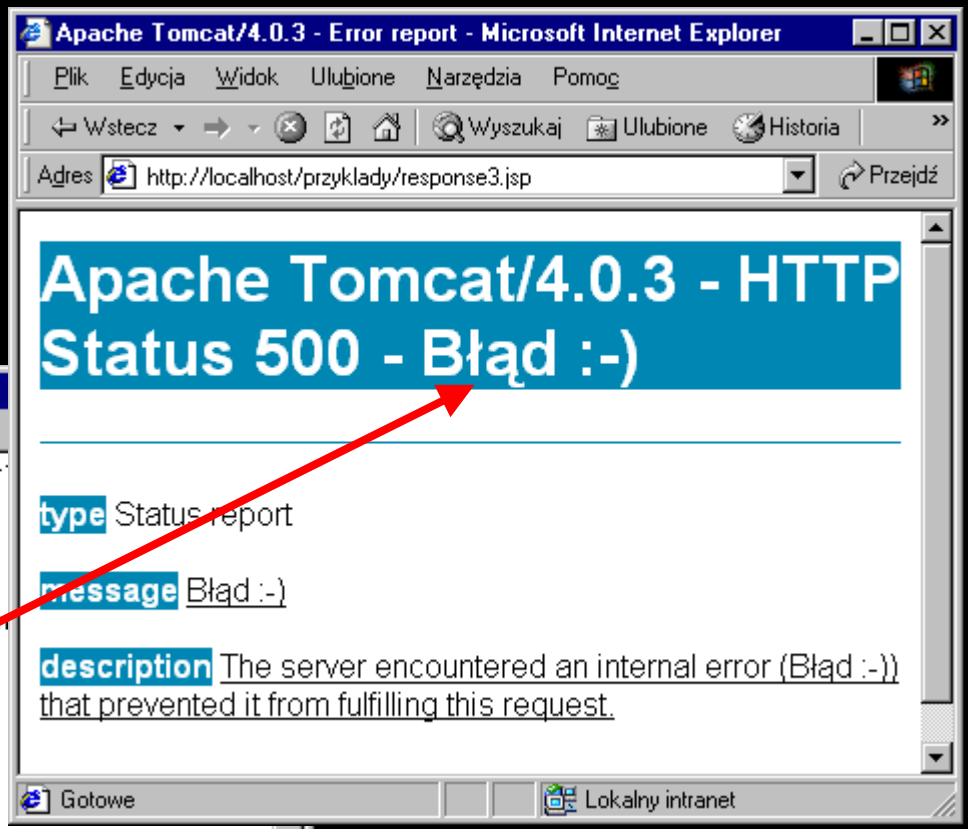
Strona JSP (serwer)

Obiekt response

- sendError - generacja kodu statusu o podanym numerze

Strona JSP (serwer)

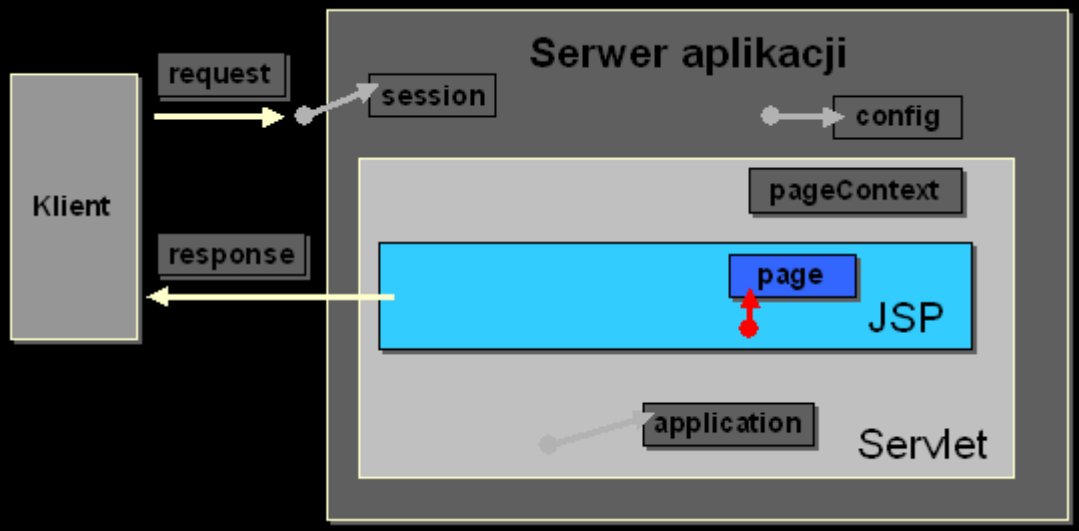




Przeglądarka (klient)

Obiekt page

Obiekt page reprezentuje bieżący dokument JSP

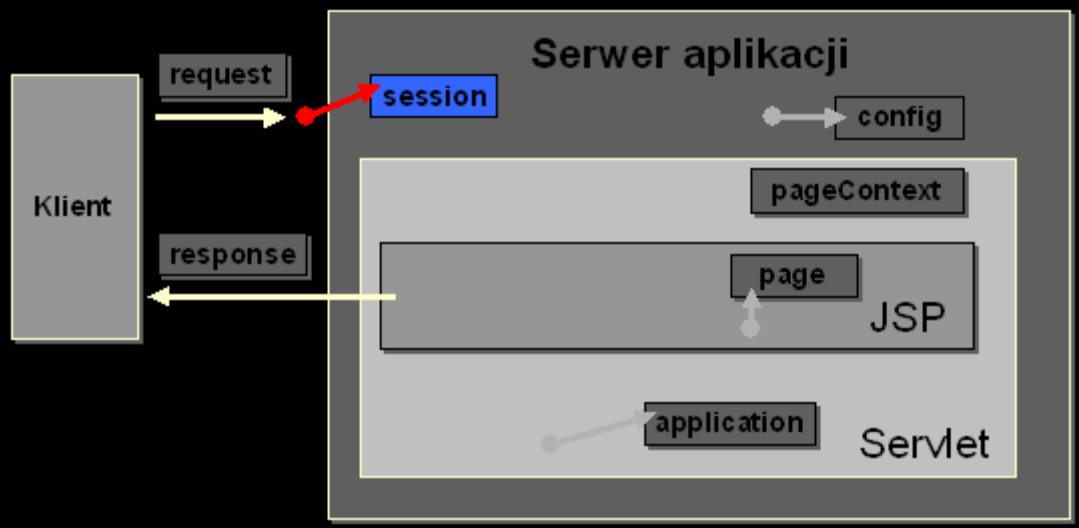


Obiekt page jest inicjalizowany wartością this odpowiednią dla danego serwletu (obiektu można używać tak samo jak obiektu this - oba reprezentują bieżącą instancję wygenerowanego servletu)

Obiekt session

Obiekt session pozwala zapisywać obiekty pomiędzy kolejnymi zapytaniami klienta

Bezstanowość protokołu HTTP - niemożliwość przechowywania danych dla kilku żądań wybranego klienta



Sesje - możliwość komunikowania się kolejnych żądań wybranego klienta

Obiekt session

Sesja rozpoczyna się w momencie wejścia klienta na daną stronę i kończy się po:

- wyjściu z witryny
- upłynięciu czasu trwania sesji
- wywołaniu na serwerze metody zakończenia sesji
- zamknięciu przeglądarki

Obiekt session

setAttribute - ustawienie atrybutu dla sesji

getAttribute - pobranie atrybutu dla sesji

removeAttribute - usunięcie atrybutu dla sesji

setMaxInactiveInternal - ustawianie czasu dla trwania sesji

getMaxInactiveInternal - odczyt aktualnej wartości czasu trwania sesji

Obiekt session

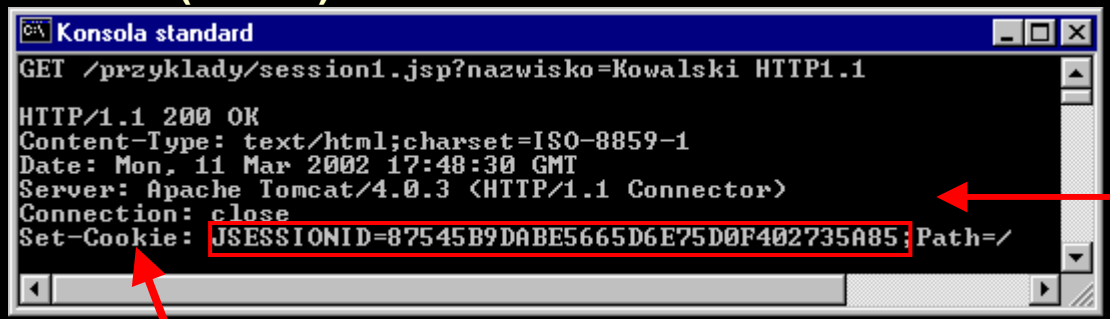
isNew - sprawdzenie, czy klient ma już połączenie z daną sesją

invalidate - wymazanie atrybutu sesyjności strony

getId - pobranie identyfikatora sesji

Obiekt session

Telnet (klient)



Zapis identyfikatora JSESSIONID jako Cookie

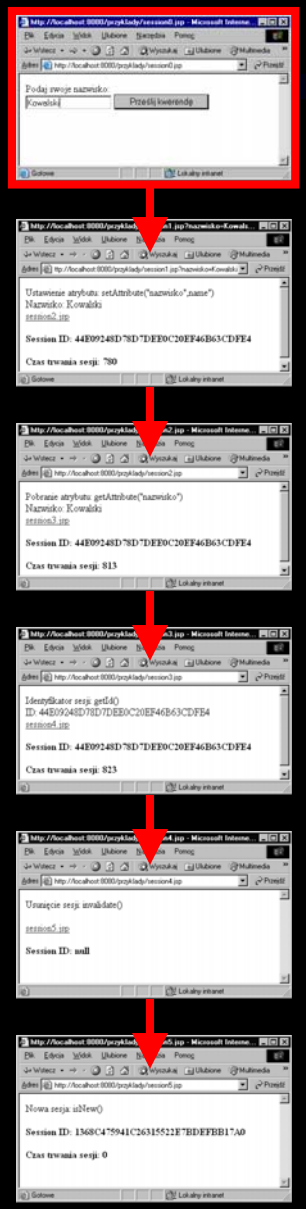
Kontener (serwer)

JSESSIONID 8754B9D...	sessionObj

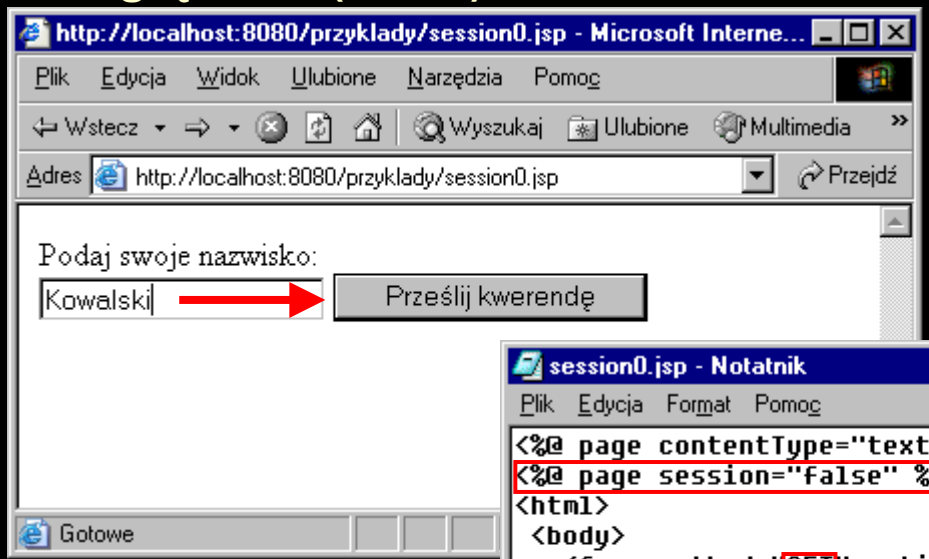
sessionObj	
nazwisko	Kowalski
wiek	29
. . .	. . .

Sesja tworzona jest przez kontener i klient jest dołączany do sesji po odesłaniu żądania

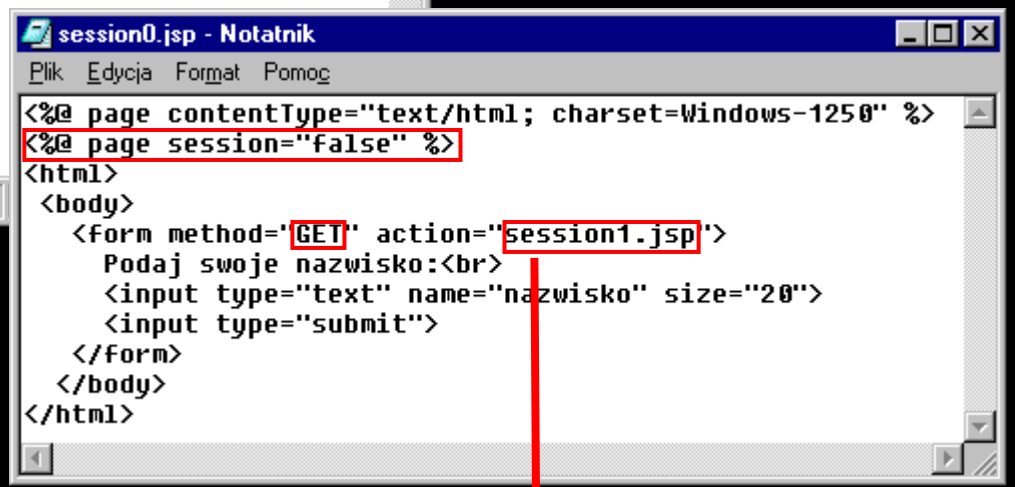
Session



Przeglądarka (klient)

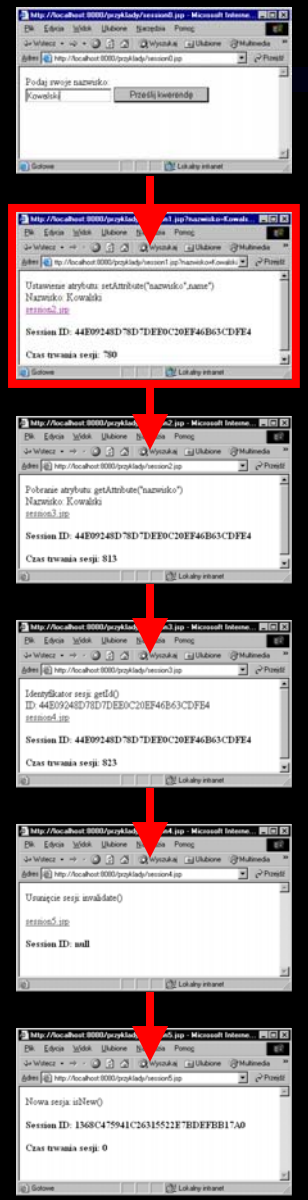


Strona JSP (serwer)

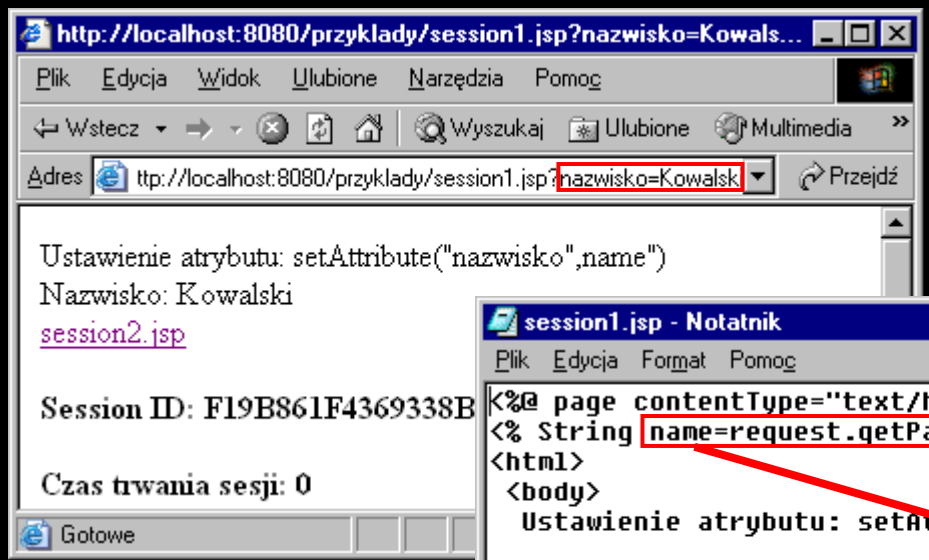


Wysłanie żądania do strony session1.jsp

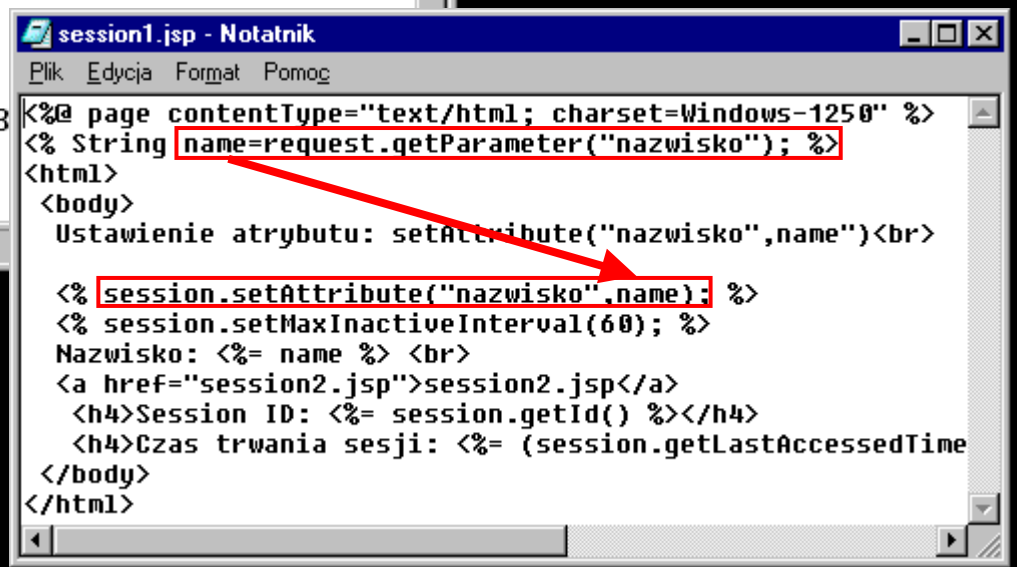
session.setAttribute



Przeglądarka (klient)

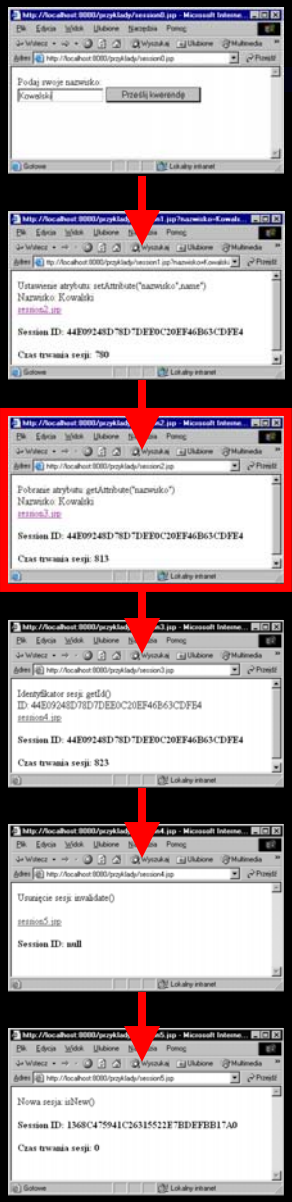


Strona JSP (serwer)



setAttribute - ustawienie atrybutu "nazwisko"

session.getAttribute



Przeglądarka (klient)

http://localhost:8080/przyklady/session2.jsp - Microsoft Interne...

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Multimedia

Adres http://localhost:8080/przyklady/session2.jsp Przejdź

Pobranie atrybutu: getAttribute("nazwisko")
Nazwisko: Kowalski
session3.jsp

Session ID: F19B861F4369338B6242EE83A502947B

Czas trwania sesji: 0

Strona JSP (serwer)

session2.jsp - Notatnik

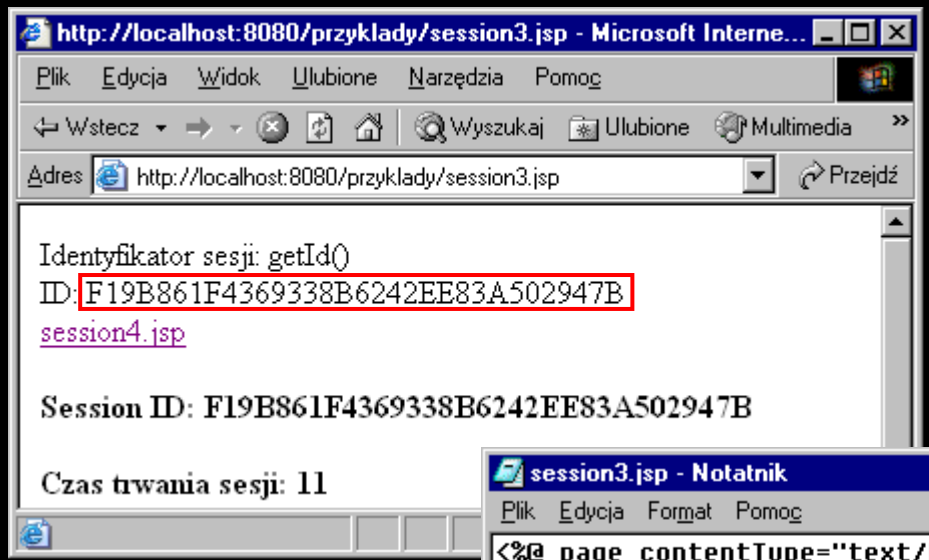
Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<html>
 <body>
  Pobranie atrybutu: getAttribute("nazwisko")<br>
  Nazwisko: <%= session.getAttribute("nazwisko") %><br>
  <a href="session3.jsp">session3.jsp</a>
   <h4>Session ID: <%= session.getId() %></h4>
   <h4>Czas trwania sesji: <%= (session.getLastAccessedTime
 </body>
</html>
```

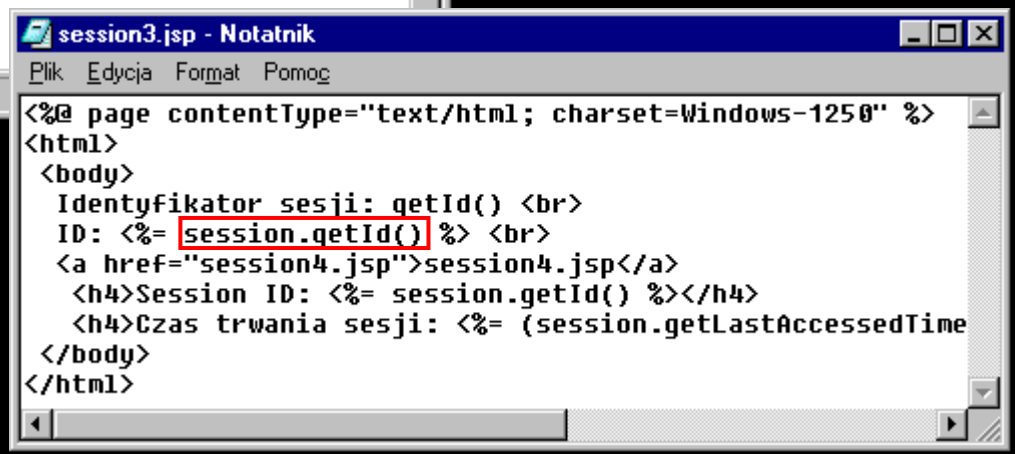
getAttribute - pobranie atrybutu "nazwisko"

session.getId

Przeglądarka (klient)

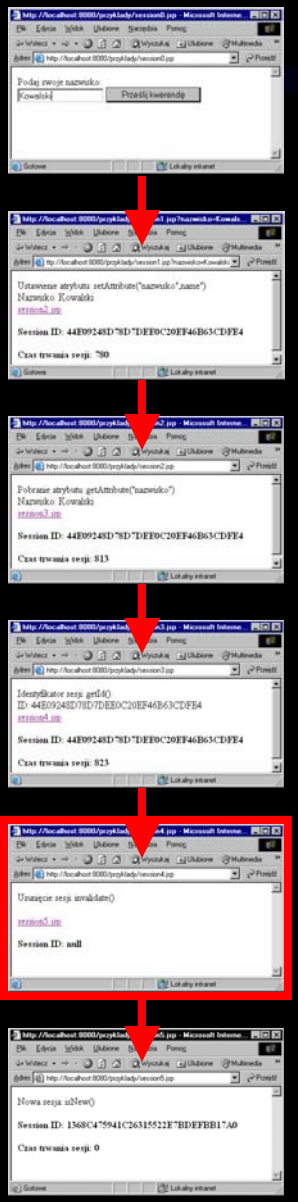


Strona JSP (serwer)



getId - pobranie identyfikatora sesji

session.invalidate



Przeglądarka (klient)

http://localhost:8080/przyklady/session4.jsp - Microsoft Interne...

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Multimedia

Adres http://localhost:8080/przyklady/session4.jsp Przejdź

Usunięcie sesji: invalidate()

session5.jsp

Session ID: null

session4.jsp - Notatnik

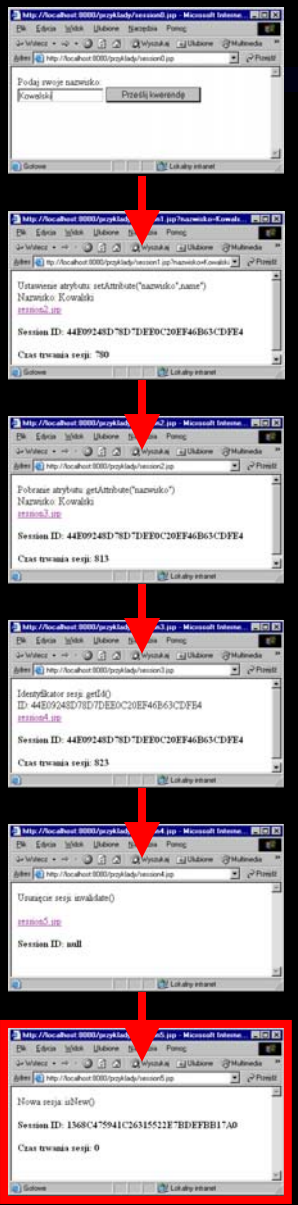
Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<html>
 <body>
  Usunięcie sesji: invalidate() <% session.invalidate(); %>
  <br>
  <br>
  <a href="session5.jsp">session5.jsp</a>
   <h4>Session ID: <%= session.getId() %></h4>
 </body>
</html>
```

Strona JSP (serwer)

invalidate - usunięcie sesji

session.isNew



Przeglądarka (klient)

http://localhost:8080/przyklady/session5.jsp - Microsoft Interne...

Plik Edycja Widok Ulubione Narzędzia Pomoc

Wstecz Wyszukaj Ulubione Multimedia

Adres http://localhost:8080/przyklady/session5.jsp Przejdź

Nowa sesja: isNew()

Session ID: 9F06796806566CAC48E0F2B3589AA886

Czas trwania sesji: 0

Gotowe

session5.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<html>
 <body>
  Nowa sesja: isNew()  <% session.isNew(); %> <br>
  <br>
   <h4>Session ID: <%= session.getId() %></h4>
   <h4>Czas trwania sesji: <%= (session.getLastAccessedTime
 </body>
</html>
```

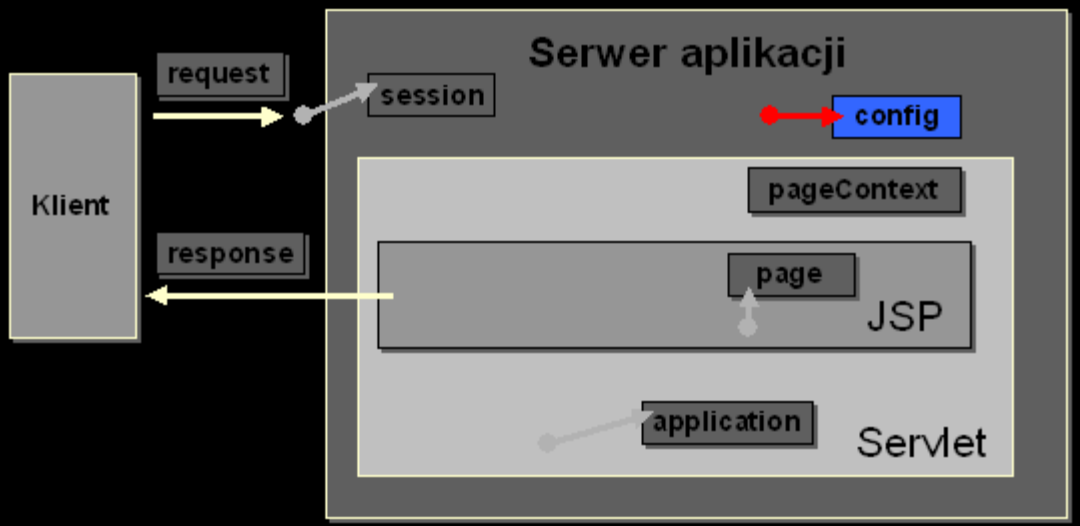
Strona JSP (serwer)

isNew - tworzenie nowej sesji

Obiekt config

Obiekt config jest generowany przez kontener JSP (pobieranie parametrów konfiguracyjnych przy inicjalizacji servleta)

parametry - pary nazwa i wartość zmiennych aplikacji ustawionych przez serwer



Obiekt config

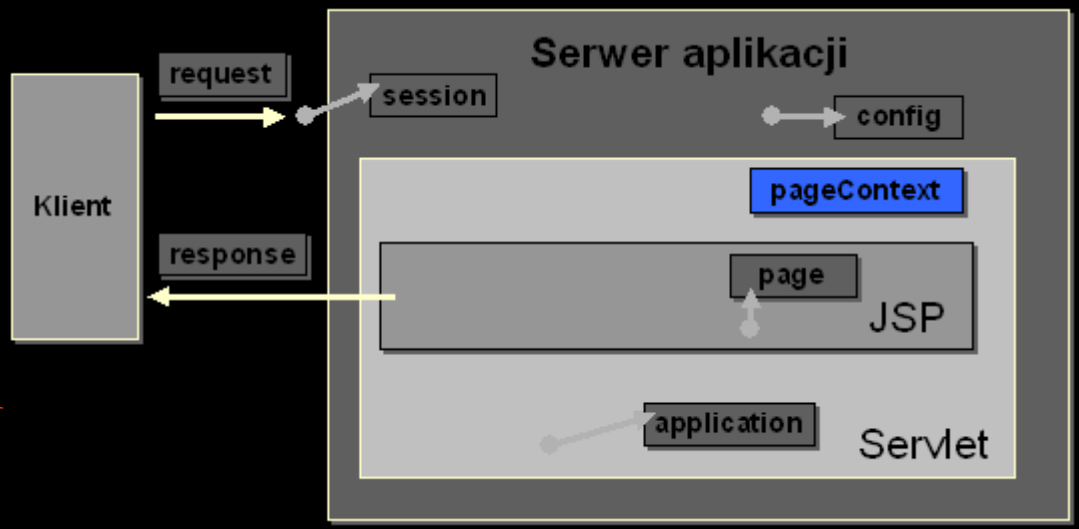
getInitParameter - zwracanie wartości parametru zadeklarowanego na serwerze (zwracanie wartości null w przypadku braku podanego parametru)

getInitParameterNames - zwracanie wszystkich parametrów globalnych

getServletName - zwracanie nazwy servletu wyświetlającego stronę

Obiekt pageContext

Kontekst strony przechowuje odwołania (referencje) do obiektów o zasięgu page



Mechanizm przechowywania danych lokalnych dla stron JSP

Każda strona JSP ma swój obiekt pageContext, który jest tworzony w momencie otwarcia strony i zamykany przy jej opuszczaniu

Obiekt pageContext

findAttribute - zwracanie wartości atrybutu
(w przypadku braku atrybutu o podanej nazwie zwracana jest wartość null)

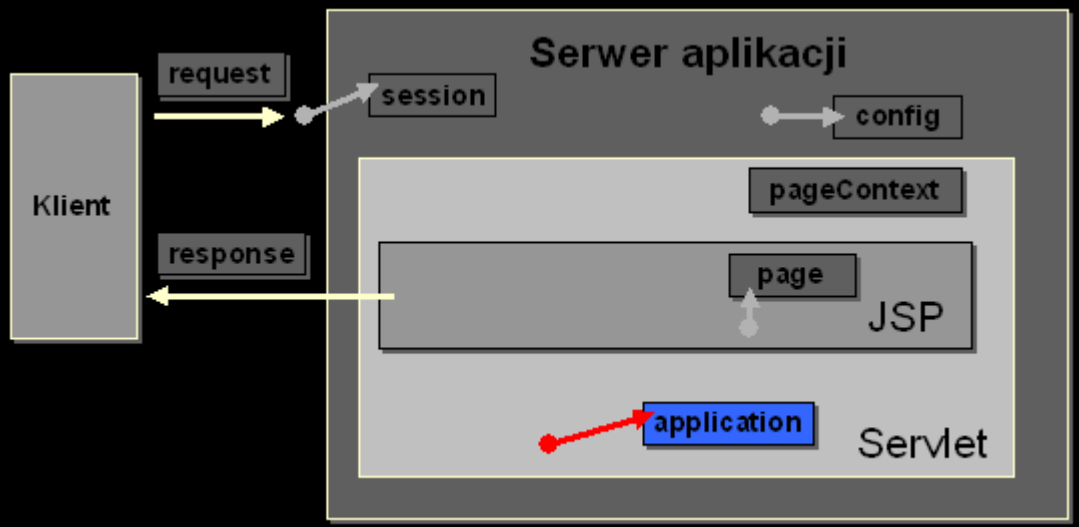
getAttribute - zwracanie wartości atrybutu

removeAttribute - usunięcie atrybutu o podanej nazwie

setAttribute - ustawienie wartości dla atrybutu

Obiekt application

Obiekt dostępny przez cały czas działania aplikacji



Obiekt umożliwia odczyt parametrów deklarowanych na serwerze

Obiekt application

getInitParameter,

getInitParametersNames - odczyt danych, które nie zmieniają się podczas działania aplikacj

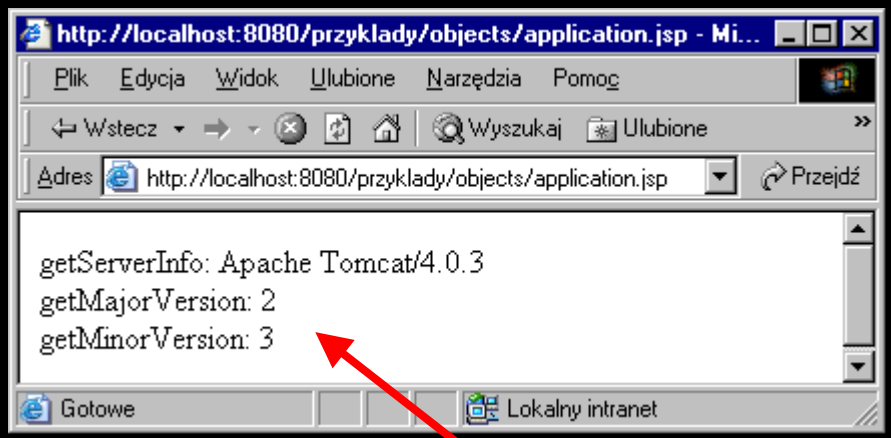
getServerInfo,

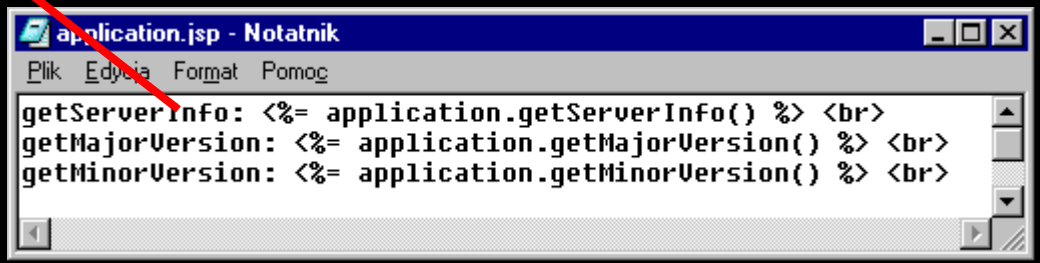
getMajorVersion,

getMinorVersion - dodatkowe informacje o serwerze, specyfikacjach serwletów/jsp

Obiekt application

Przeglądarka (klient)





Strona JSP (serwer)

Obiekt out

Obiekt służy do zapisywania danych do strumienia wyjściowego, który zostaje odesłany do klienta (kod HTML wyświetlany w przeglądarce)

```
out.println();
```

print - wysłanie tekstu do strumienia wyjściowego bez znaków nowej lini

println - wysłanie tekstu do strumienia wyjściowego ze znakami nowej lini

Obiekt out

clear - wyczyszczenie bufora strumienia wyjściowego bez wysłania zawartości do klienta (generacja wyjątku przy wcześniejszym wywołaniu metody flush)

clearBuffer - wyczyszczenie zawartości bufora strumienia wyjściowego (bez wysłania zawartości bufora do klienta)

Obiekt out

flush - opróżnienie bufora (następuje wysłanie zawartości bufora do klienta)

getBufferSize - wyświetlenie wielkości bufora

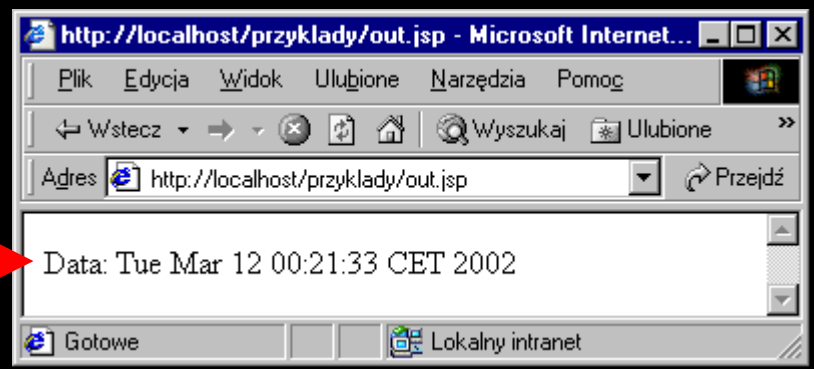
getRemaining - ilość pustego miejsca w buforze

isAutoFlush - sprawdzenie włączenia automatycznego opróżniania bufora

newLine - wpisanie znaku nowej lini do strumienia wyjściowego

Obiekt out

```
<html>
  <body>
    <%
      System.out.println("Pobranie dzisiejszej daty");
      java.util.Date data = new java.util.Date();
    %>
      Data:
    <% out.println(String.valueOf(data)); %>
  </body>
</html>
```



Obiekt exception

Obiekt wyjątku dostępny tylko w obrębie stron diagnostycznych zawierający odwołanie (referencje) do nieprzechwyconych wyjątków

getMessage - wyświetlenie komunikatu o błędzie

printStackTrace - wysłanie obiektu exception oraz przebiegu jego wystąpienia, do standardowego strumienia błędów

toString - opis wygenerowanego wyjątku

Artybut isErrorPage=„true” dyrektywy page tworzy obiekt exception

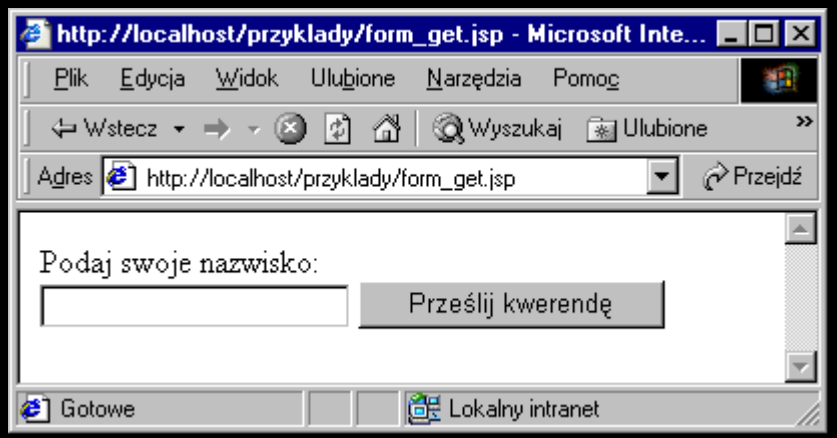
Forms

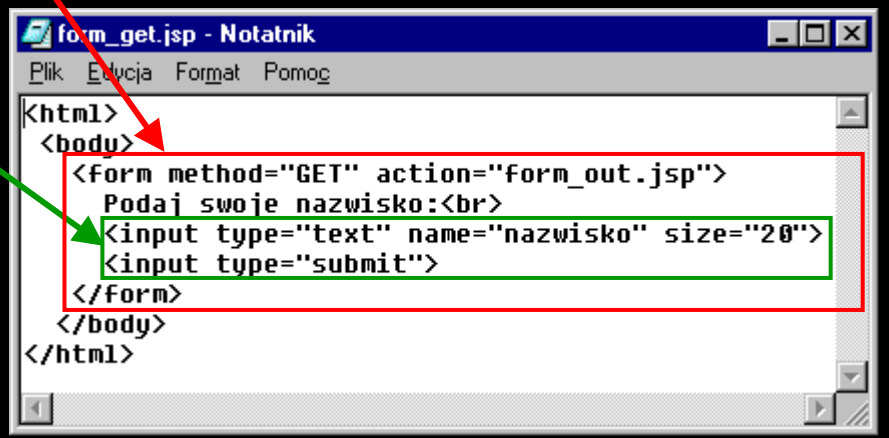
Formularze HTML

<form> - znacznik służący do definiowania formularza

<input> - znacznik pozwala przechwytywać dane

Przeglądarka (klient)

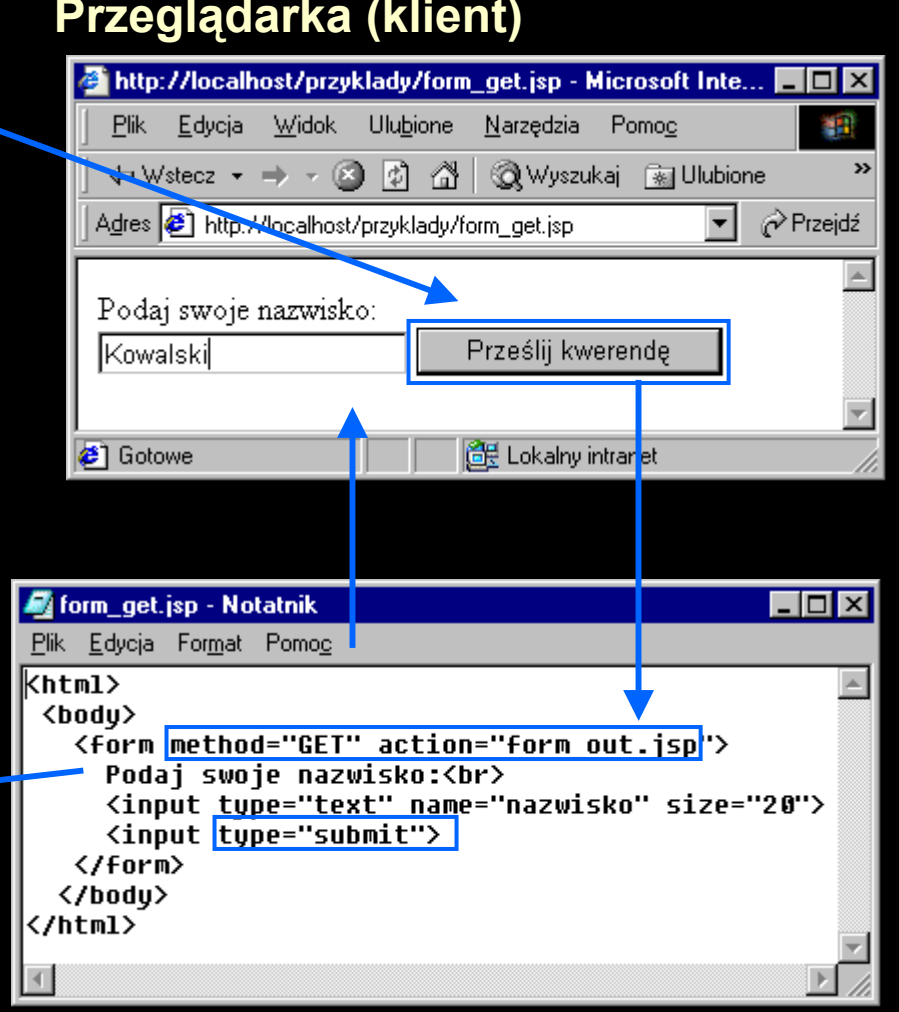




Strona JSP/HTML (serwer)

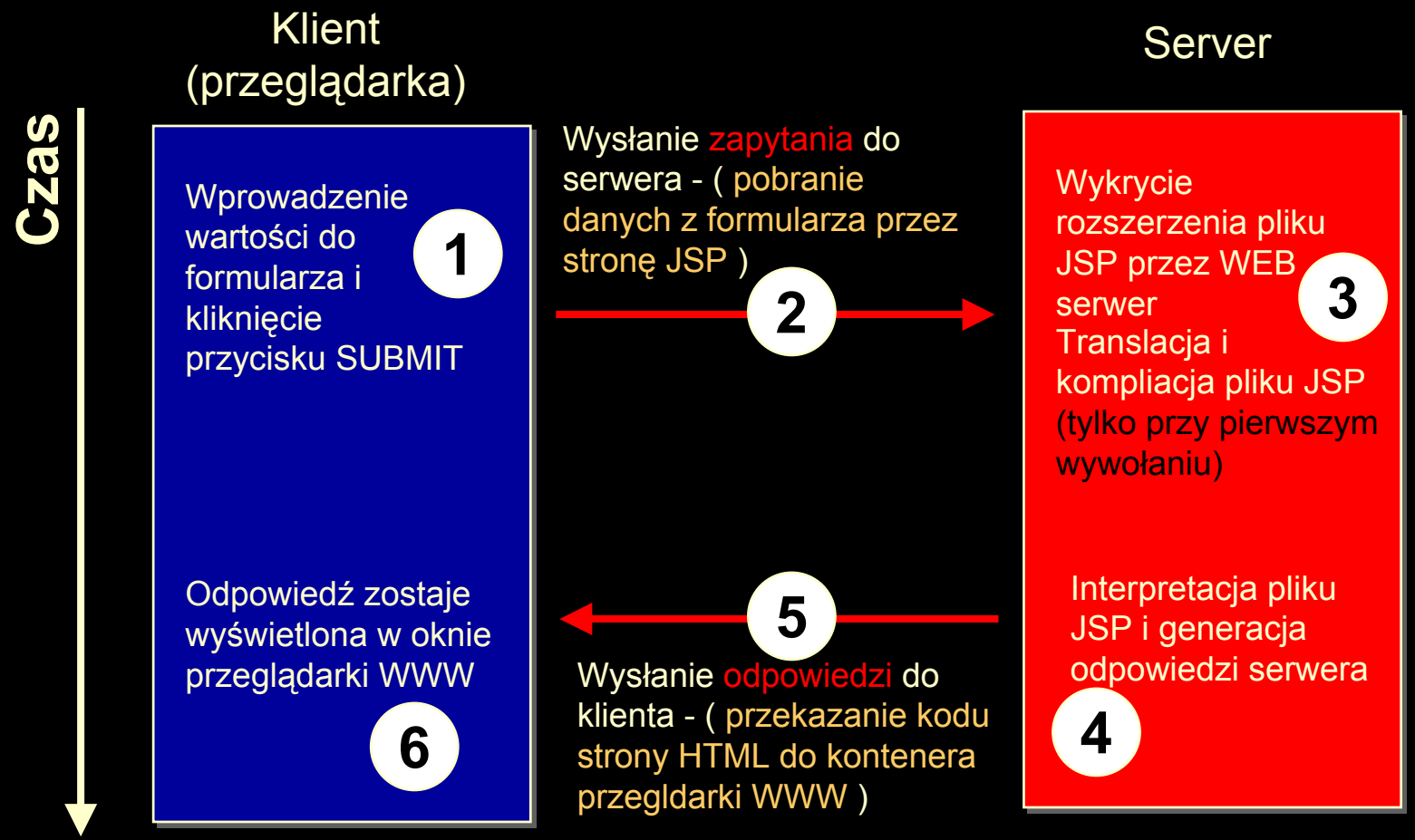
Formularze HTML

Przycisk **submit** Prześlij kwerendę powoduje wysłanie danych pod wskazany adres atrybutu **action**



Metody **GET** i **POST** są metodami bezpiecznymi - realizują tylko pobieranie informacji

Obsługa formularzy



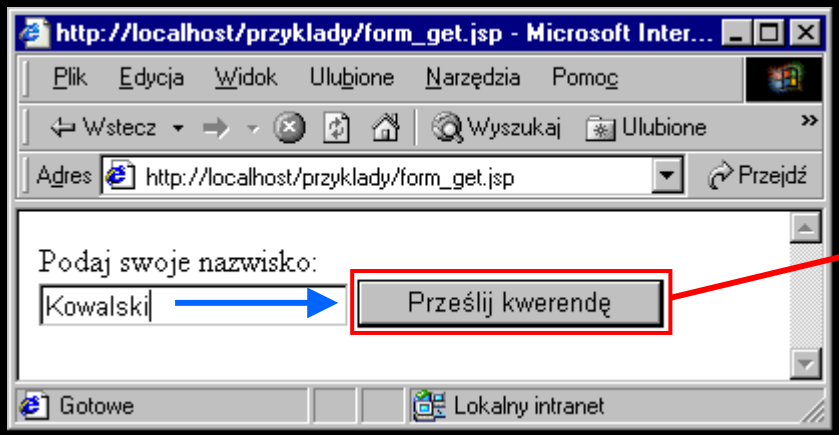
Metoda GET

Metoda GET jest główną metodą do pobierania plików

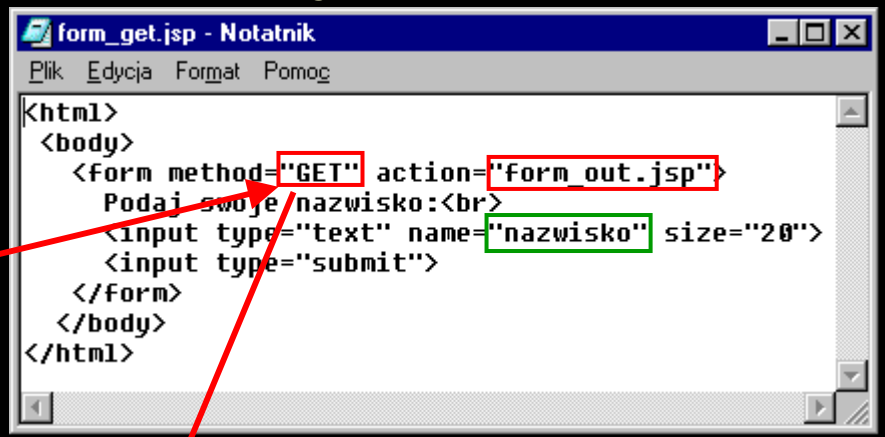
- przesyłanie danych z formularza
  - zmniejszenie obciążenia (brak potrzeby wykonania wielu żądań)
  - jawność danych (możliwość modyfikacji)
  - ograniczenie w długości przesyłanych informacji

Metoda GET

Przeglądarka (klient)



Źródło strony HTML (klient)



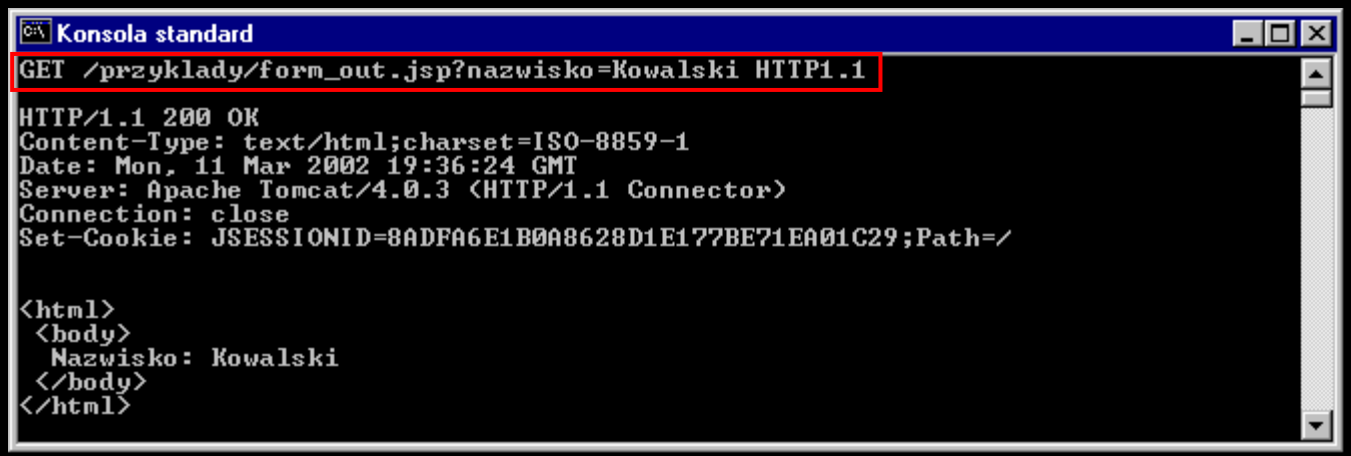
Wysłanie żądania do serwera (metoda GET)

```
GET form_out.jsp?nazwisko=Kowalski
```

Metoda GET

```
GET form_out.jsp?nazwisko=Kowalski
```

Wysłanie żądania do serwera (metoda GET)

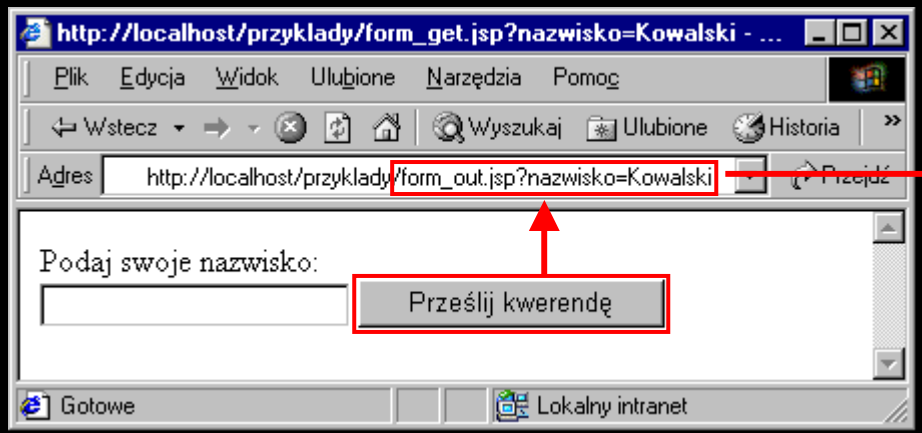


telnet - bezpośrednie wysłanie żądania bez udziału formularza (metoda GET)

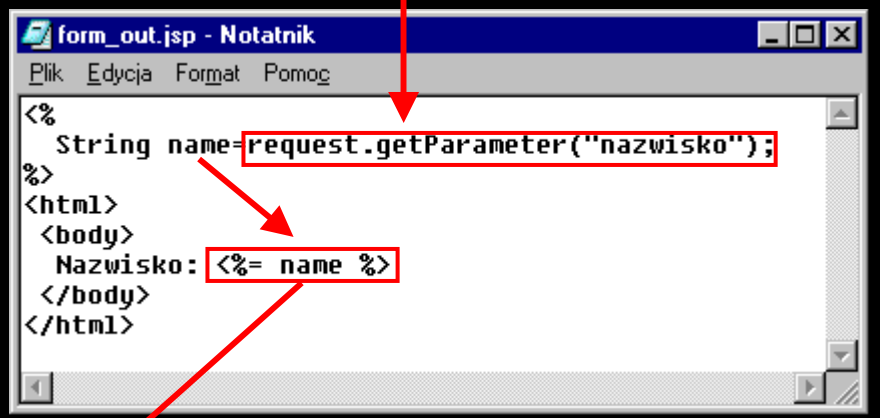
Metoda GET

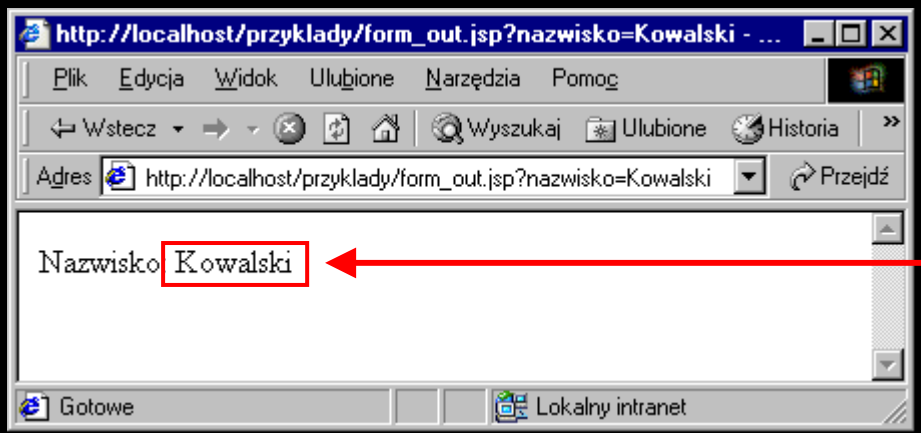
GET form_out.jsp?nazwisko=Kowalski

Przeglądarka (klient)



Strona JSP (serwer)





Pobranie danych żądania (obiekt request)

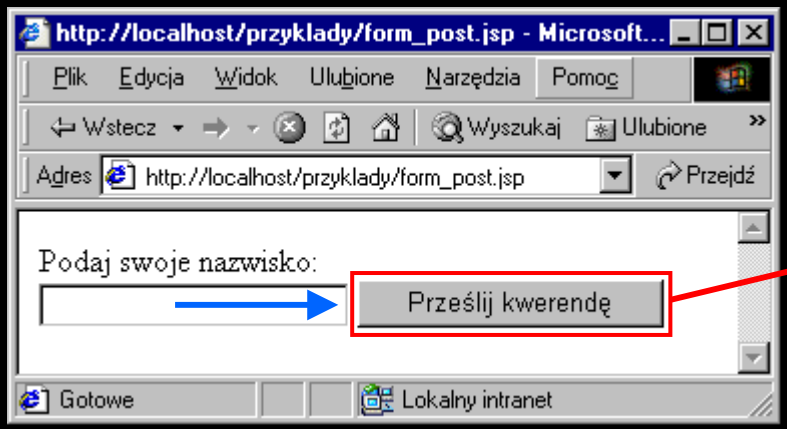
Metoda POST

Metoda POST wysyła dane do serwera

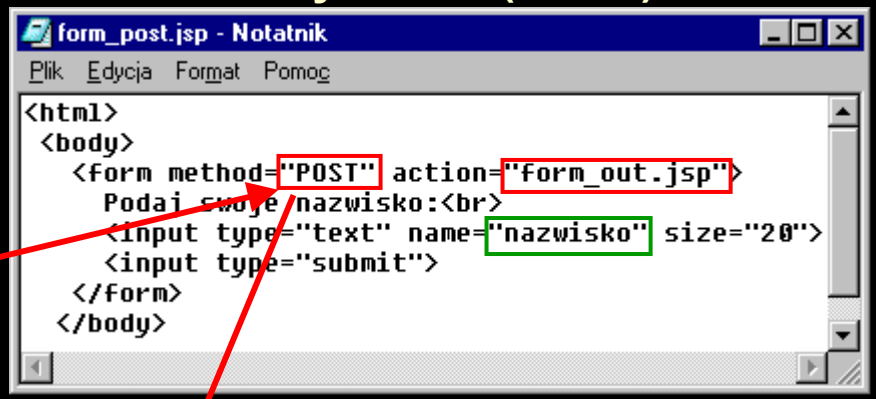
- przesłanie bloku danych jako wynik przesłania formularza
  - dane wysyłane są do serwera w treści żądania
  - po przetworzeniu żądania serwer może przekazać treść żądania innej stronie określonej przez adres URL zajmującej się dalszym przetwarzaniem żądania

Metoda POST

Przeglądarka (klient)



Źródło strony HTML (klient)

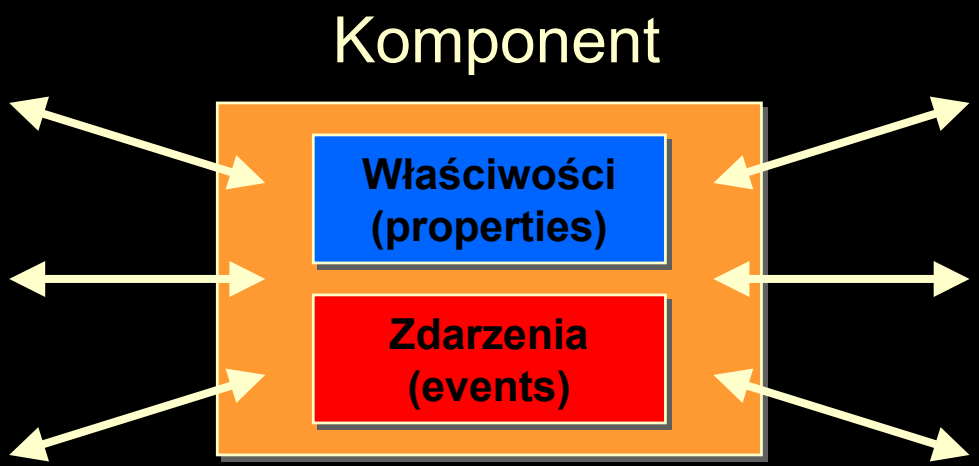


Wysłanie żądania do serwera (metoda POST)

Wysłanie atrybutów i ich wartości dokonywane jest za pośrednictwem nagłówka HTTP

Java Bean

Komponent



- Właściwości - charakterystyka komponentu
- Zdarzenia - zbiór zachowań

Czym jest Bean?

- Bean - zapis komponentu w postaci klasy
- Określone nazewnictwo metod definiuje właściwość, zdarzenie lub zwyczajną metodę
  - właściwość xxx
  - metoda setXxx()
  - metoda getXxx()
  - metoda isXxx()

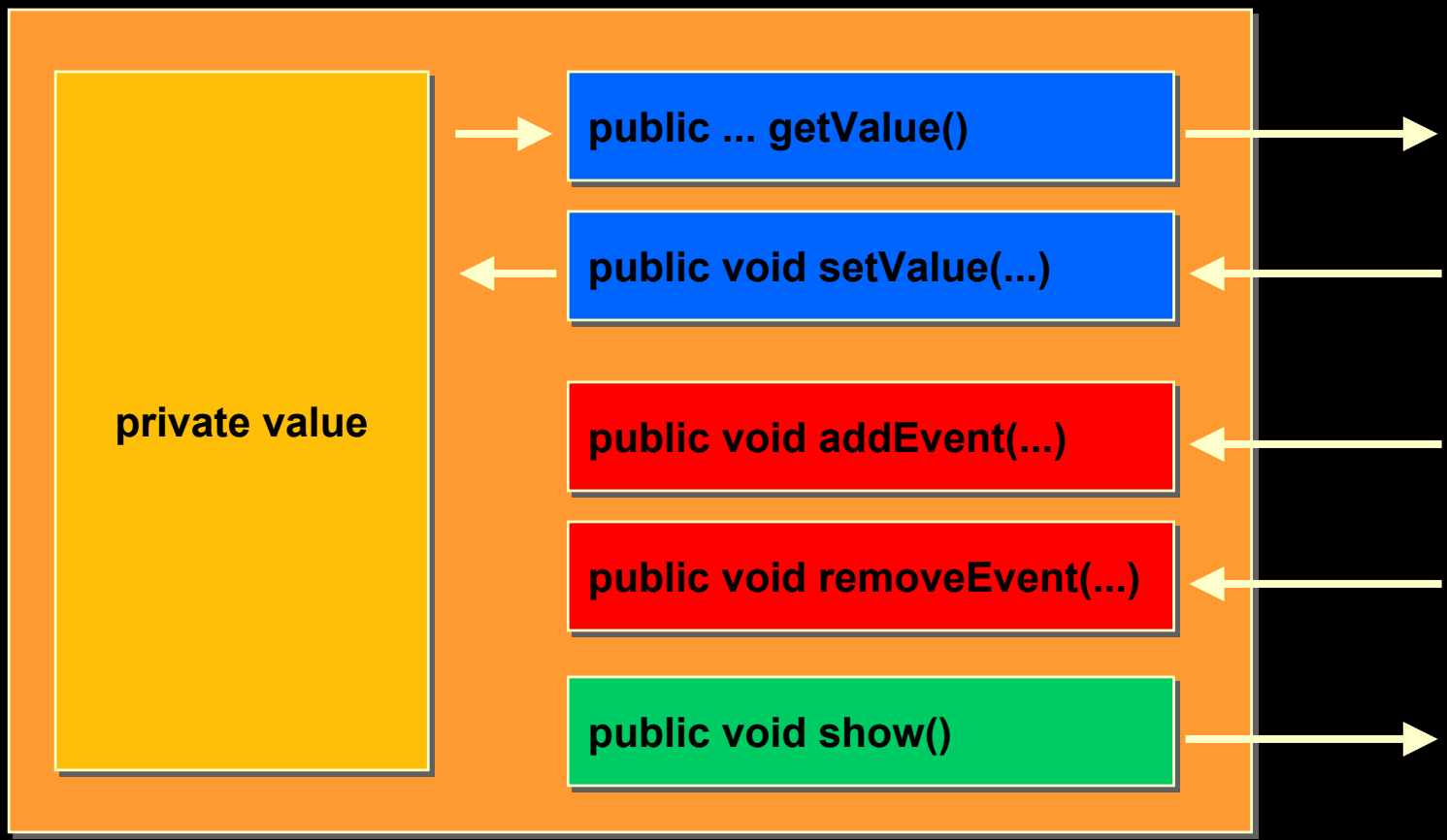
Dla właściwości typu boolean zamiast get można użyć metody is.

Java Bean

- Zdarzenia
  - metoda addXxx()
  - metoda removeXxx()
- Zwyczajne metody Bean nie stosują się do nazewnictwa set/get, ale są to metody publiczne

Java Bean

Komponent JavaBean



Serializacja

- Implementacja interfejsu Serializable
  - odtwarzanie i zachowywanie wartości właściwości komponentu

  (trwałość - zachowanie stanu obiektu dłużej niż czas pojedynczego uruchomienia programu)

JavaBean - Licznik.java

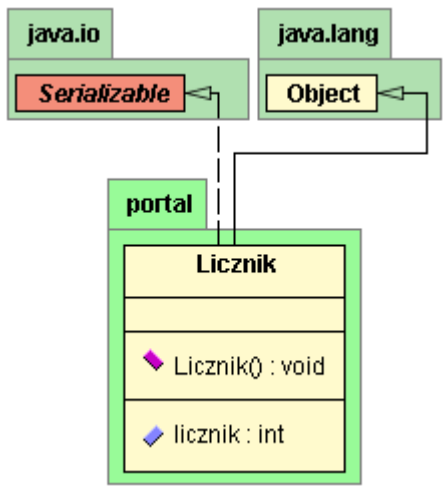
classes - specjalny katalog do umieszczania klas aplikacji

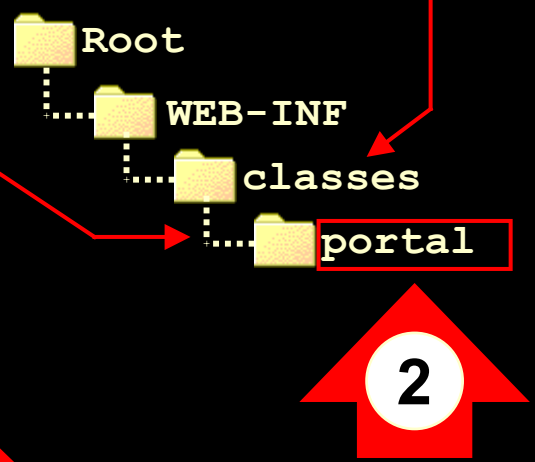
Plik klasy Licznik

Licznik.java - Notatnik

Plik Edycja Format Pomoc

```
package portal;

import java.beans.*;

public class Licznik extends Object implements java.io.Serializable {

    int licznik = 0;

    public Licznik() {

    }

    public int getLicznik() {
        licznik++;
        return this.licznik;
    }

    public void setLicznik (int value) {
        this.licznik = value;
    }
}
```





2

1

Licznik.class

Wywołanie JavaBean'a z JSP

Strona JSP (serwer)

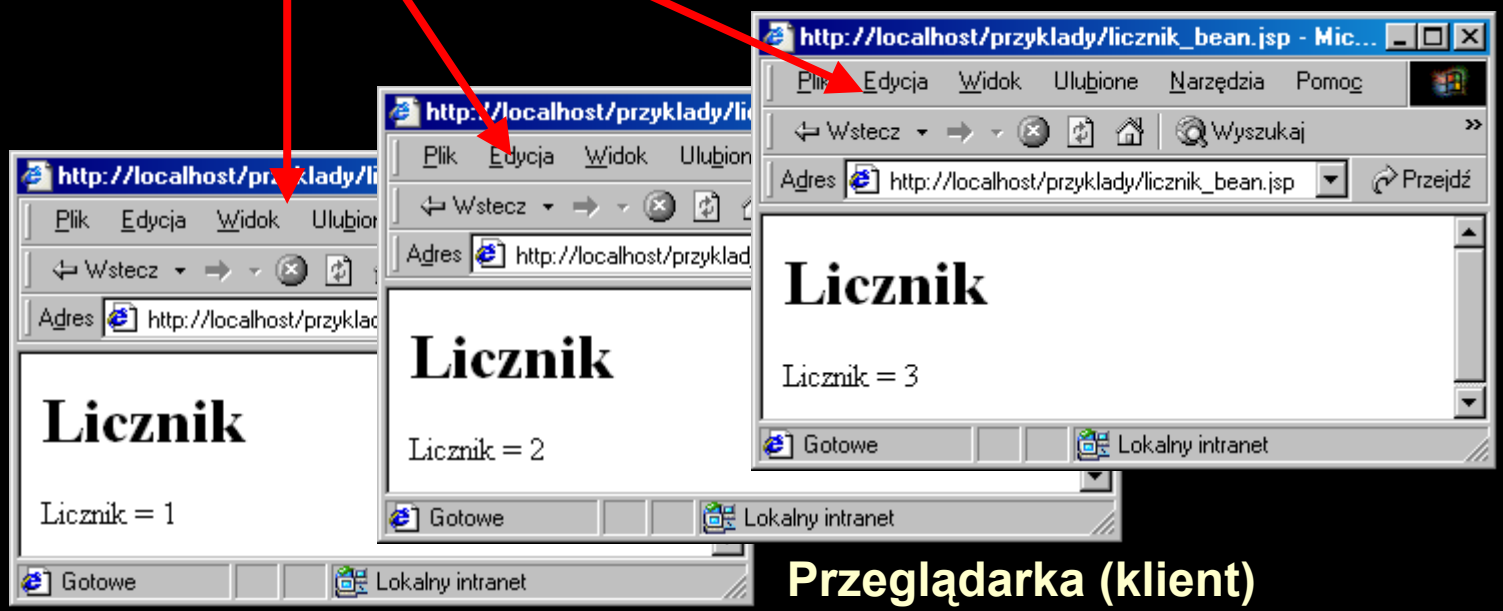
licznik_bean.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page language="java" import="portal.*" %>
<jsp:useBean id="licznik" scope="session" class="portal.Licznik" />

<h1>Licznik</h1>
Licznik = <jsp:getProperty name="licznik" property="licznik" />
```

Klasa JavaBean

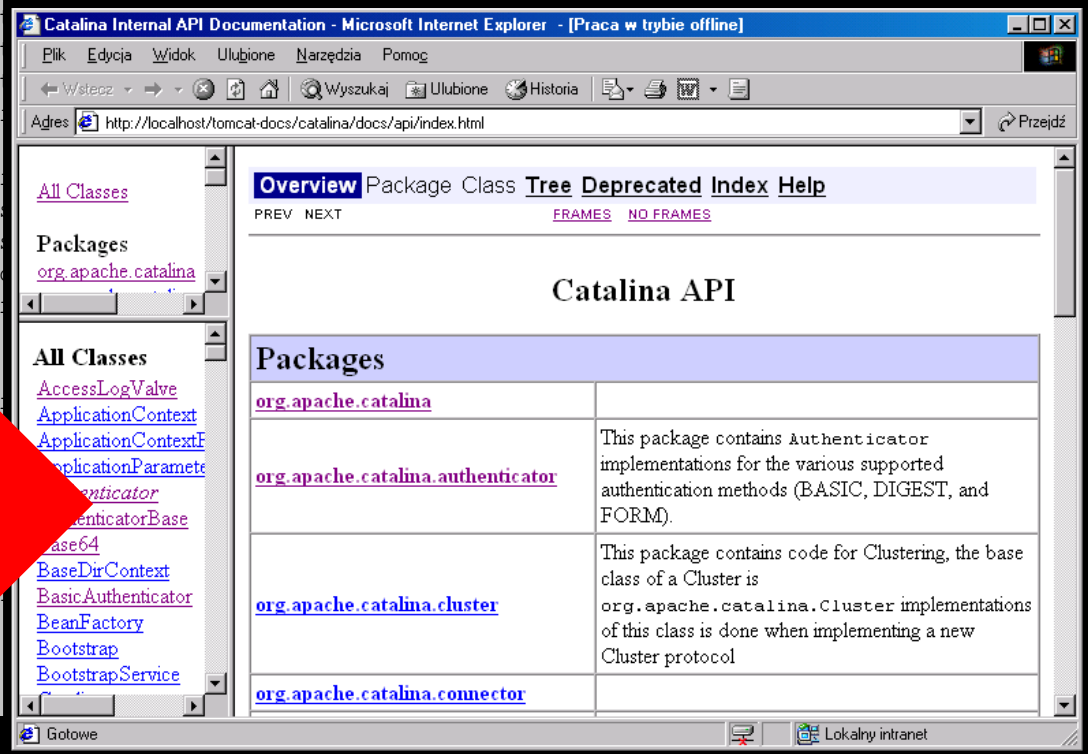
Licznik.class



Przeglądarka (klient)

Dokumentacja - javadoc

```
javadoc [options] [packagenames] [sourcefiles] [classnames] [@files]
-overview <file>          Read overview documentation from HTML file
-public                   Show only public classes and members
-protected                Show protected/public classes and members (default)
-package                  Show package/protected/public classes and members
-private                  Show all classes and members
-help                     Display command line options
-doclet <class>           Generate output via alternate doclet
-docletpath <path>        Specify where to find doclet class files
-1.1                      Generate outpu
-sourcepath <pathlist>    Specify where
-classpath <pathlist>     Specify where
-bootclasspath <pathlist> Override locat
                                      by
-extdirs <dirlist>        Override locat
-verbose                  Output message
-locale <name>            Locale to be u
-encoding <name>          Source file en
-J<flag>                  Pass <flag> di

Provided by Standard doclet:
-d <directory>            Destination d
-use                      Create class
-version                  Include
-author                   Include
-splitindex               Split in
-windowtitle <text>       Browser
-doctitle <html-code>     Include title
-header <html-code>       Include header
...
...
```



JSP error?!

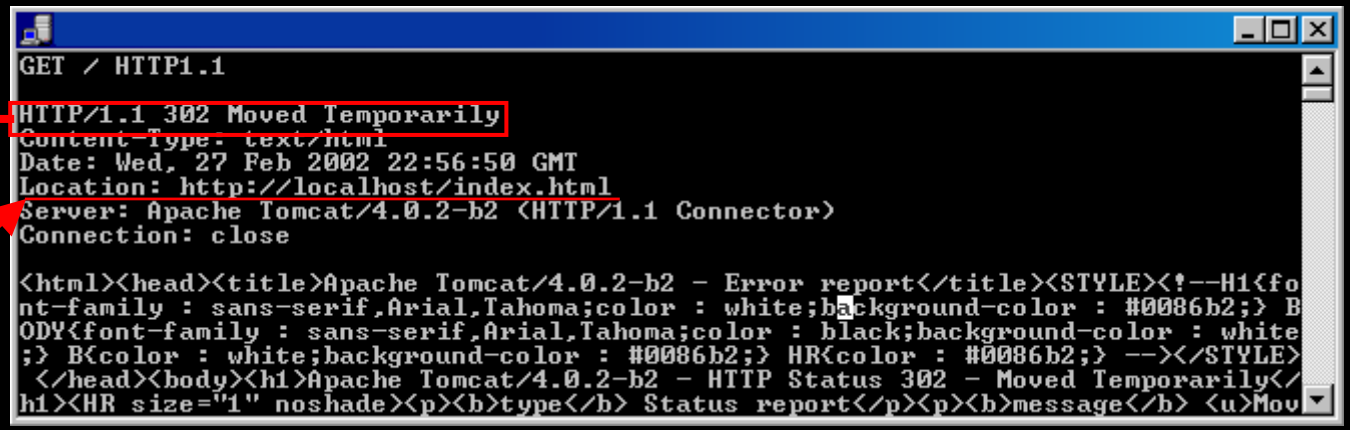
Rodzaje błędów:

- błędy kompilacji stron
- błędy przetwarzania żądań HTTP
  - mechanizm wyjątków zaimplementowany w języku Javie
  - przekierowanie na stronę obsługi błędów (errorPage)

Kody stanu HTTP

Realizując połączenie HTTP1.1 serwer WWW przesyła do klienta 3 cyfrowy kod stanu

302

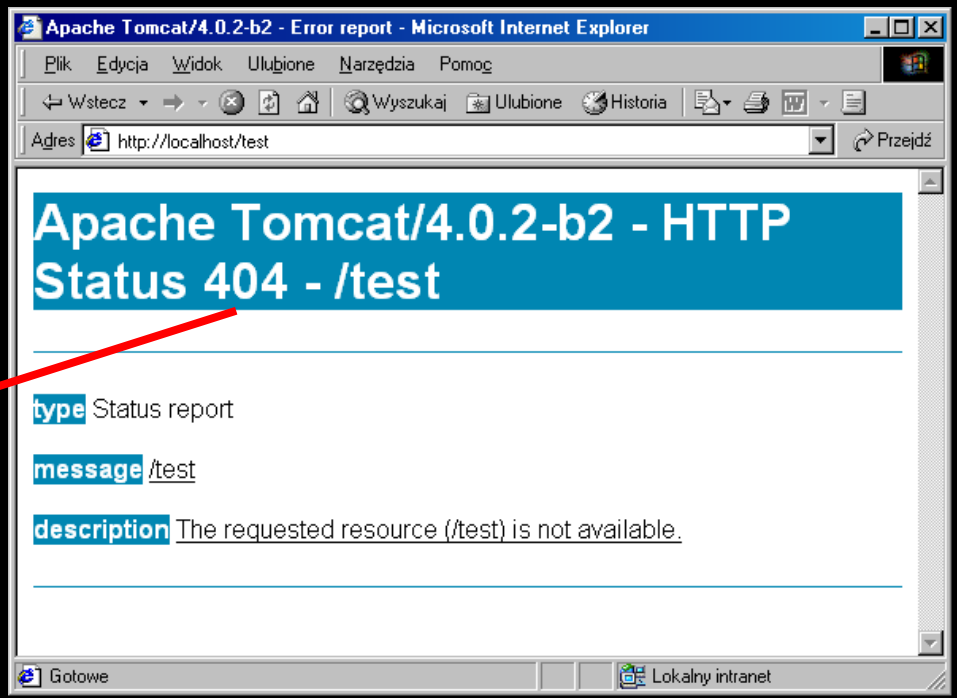


W przykładzie uzyskano kod 302 Moved Temporararily oznaczający, że dany zasób został przeniesiony pod adres nagłówka Location

Kody stanu HTTP

Reakcja serwera TOMCAT na kody błędów

404



Kod 404 Not Found oznacza, że serwer nie znalazł danego zasobu.

Klasy kodów stanu HTTP

Pierwsza cyfra kodu określa klasę, a pozostałe dwie cyfry określają konkretną sytuację w danej klasie

1xx - Informational (informacja)

2xx - Successful (prawidłowa realizacja żądań)

3xx - Redirection (przekierowanie)

4xx - Client Error (błąd klienta)

5xx - Server Error (błąd servera)

Cookies

Cookies

- Mechanizm przechowywania danych w przeglądarce klienta w celu jego śledzenia i identyfikacji użytkowników odwiedzających witrynę
- Ciasteczko stanowi część nagłówka wysyłanego do przeglądarki klienta

Cookies

Parametry ciasteczek

- name - nazwa ciasteczka
- Expires - data stworzenia lub modyfikacji
- Domain - adres domeny serwera
- Path - ścieżka
- Secure

JDBC

JDBC

JDBC (Java Database Connectivity)

- interfejs języka JAVA (zestaw funkcji) przeznaczony do wykonywania poleceń SQL

SQL (Structured Query Language)

- strukturalny język zapytań SQL pracy z bazami danych (ANSI SQL-2 1992)

JDBC API

JDBC API umożliwia:

- otwarcie połączenia z bazą danych
- wykonanie kwerendy
- przetworzenie uzyskanych wyników

Interfejsy, klasy i wyjątki

Interfejsy:

- Callable
- Connection
- StatementDatabaseMetaData
- Driver
- PreparedStatement
- ResultSet
- ResultSetMetaData
- Statement

Klasy:

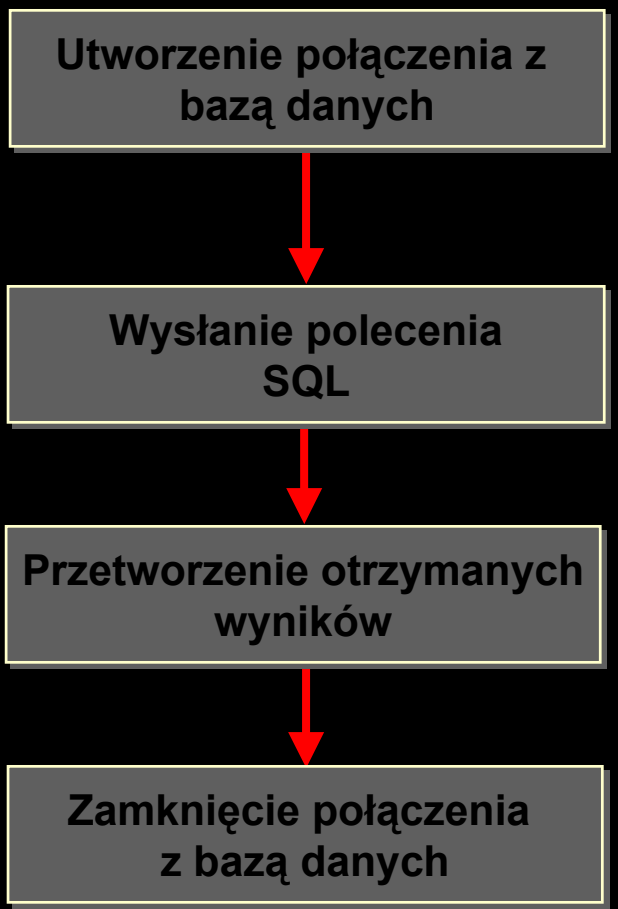
- Date
- DriverManager
- DriverPropertyInfo
- Numeric
- Time
- Timestamp
- Types

Wyjątki (exception):

- DataTruncation
- SQLException
- SQLWarning

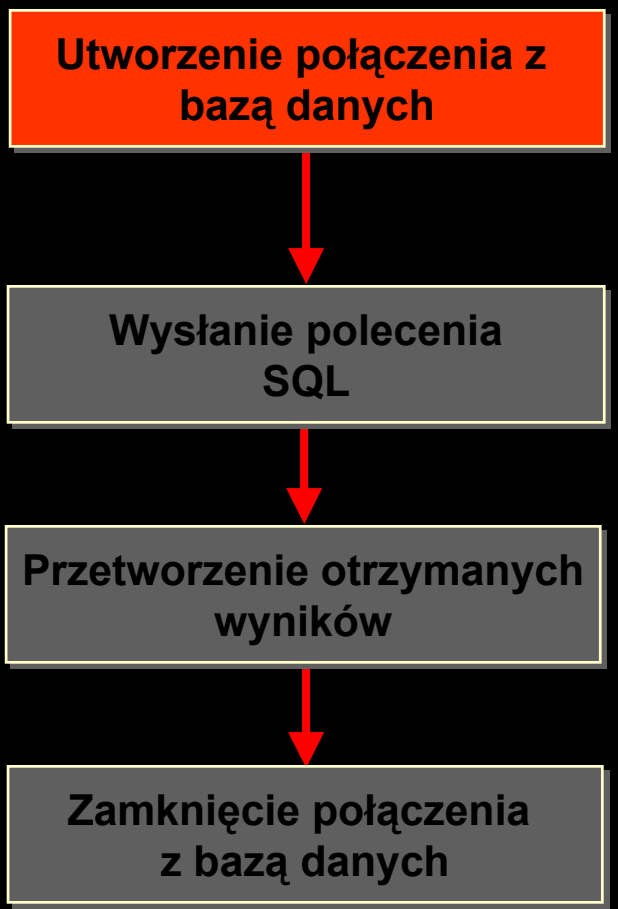
DriverManager - zarządzanie ładowaniem sterowników
Connection - połączenie z konkretną bazą danych
Statement - kontener do wywoływania poleceń SQL
ResultSet - zbiór rekordów

java.sql




```
<%@ page language="java" import="java.sql.*"%>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection dbConn =
DriverManager.getConnection("jdbc:odbc:portal",
"username", "password");

Statement stmt = dbConn.createStatement();
ResultSet rs = stmt.executeQuery(
                        "Select * from users");

if (rs != null) {
 while(rs.next()) {
        out.println(rs.getString("login"));}
}

stmt.close();
dbConn.close();
%>
```

java.sql




```
<%@ page language="java" import="java.sql.*"%>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection dbConn =
DriverManager.getConnection("jdbc:odbc:portal",
"username", "password");

Statement stmt = dbConn.createStatement();
ResultSet rs = stmt.executeQuery(
                        "Select * from users");

if (rs != null) {
 while(rs.next()) {
        out.println(rs.getString("login"));}
}

stmt.close();
dbConn.close();
%>
```

Utworzenie połączenia

Ładowanie sterownika:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Nazwa sterownika

Tworzenie połączenia:

```
Connection dbConn =
  DriverManager.getConnection(
    "jdbc:odbc:portal","username", "password");
```

Format adresu URL

`<protocol>:<subprotocol>:<subname>`

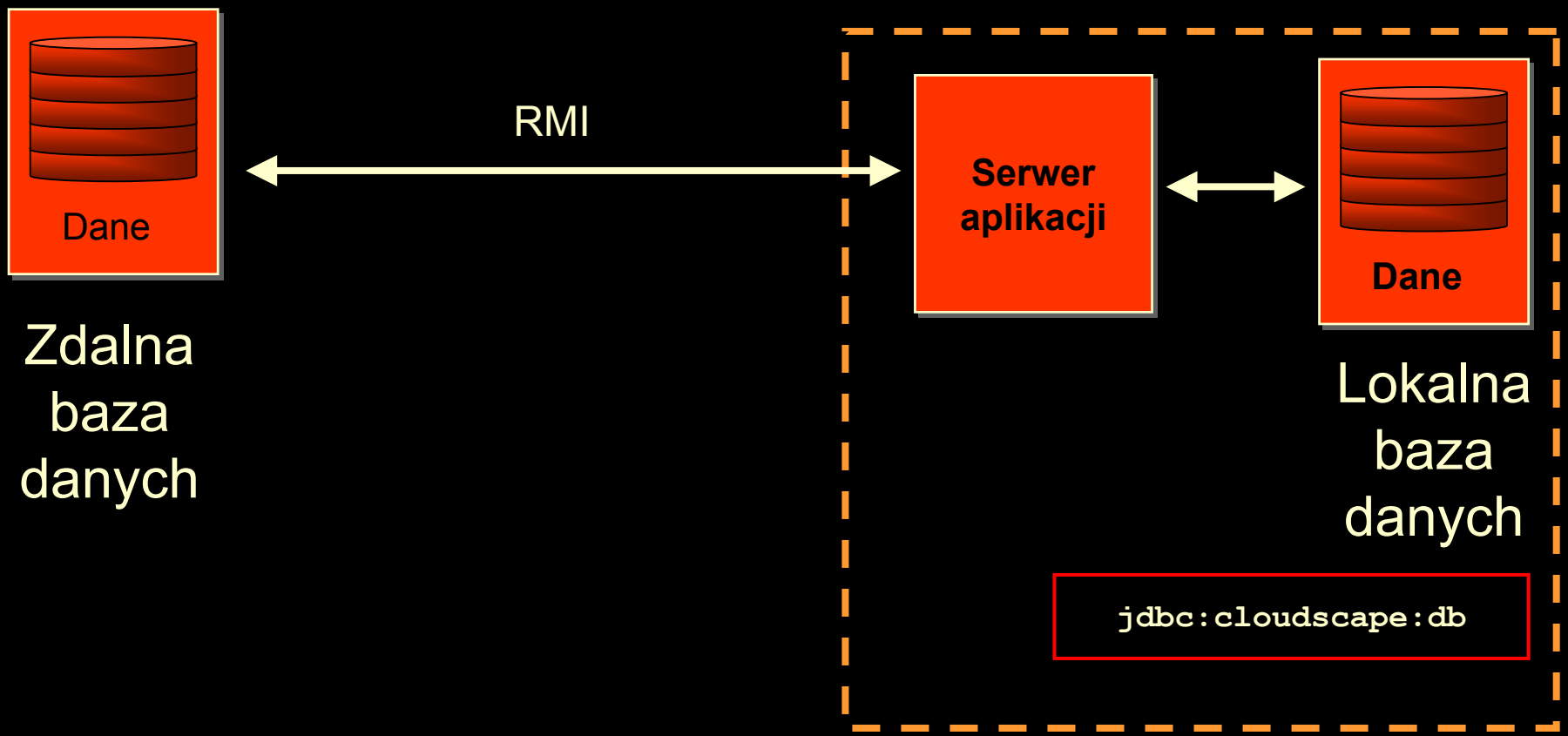
protocol - stały element jdbc

subprotocol - nazwa sterownika lub mechanizmu połączenia z bazą danych

subname - opis lokalizacji bazy danych zależny od sterownika lub od mechanizmu bazy danych (może zawierać dodatkowe rozszerzenia)

URL bazy danych

```
jdbc:rmi://192.168.170.27:1099/jdbc:cloudscape:db
```



Login i hasło

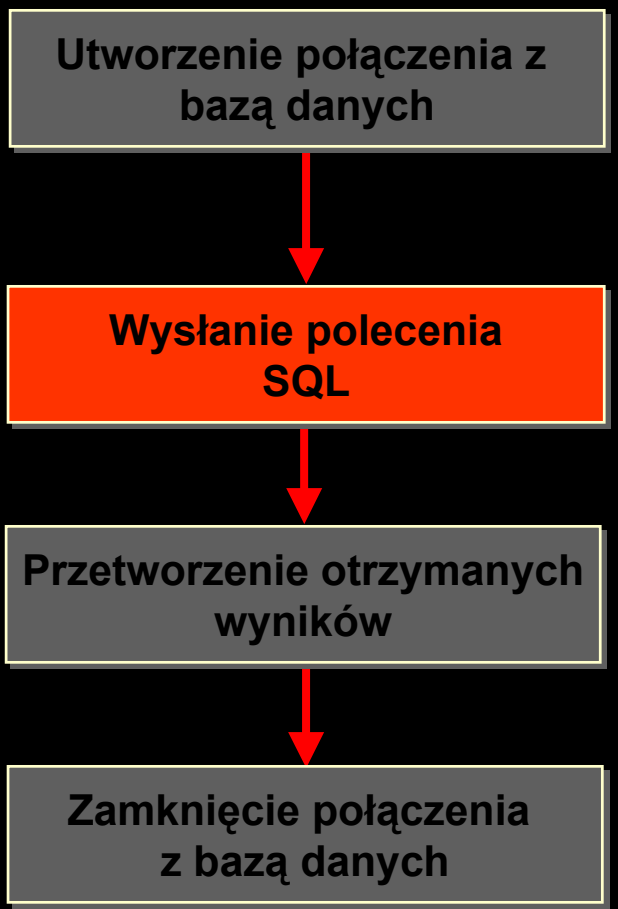
Za pomocą metody:

```
getConnection "jdbc:odbc:portal","username",
   "password");
```

istnieje możliwość przekazywania login'u i hasła do bazy danych

Całym procesem połączenia, wysyłania hasła itd... zajmuje się zarządca DriveManager

java.sql




```
<%@ page language="java" import="java.sql.*"%>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection dbConn =
DriverManager.getConnection("jdbc:odbc:portal",
"username", "password");

Statement stmt = dbConn.createStatement();
ResultSet rs = stmt.executeQuery(
                        "Select * from users");

if (rs != null) {
 while(rs.next()) {
        out.println(rs.getString("login"));}
}

stmt.close();
dbConn.close();
%>
```

Wysyłanie polecenia SQL

Przed wysłaniem polecenia do DBMS należy stworzyć obiekt Statement

```
Statement stmt = dbConn.createStatement();
```

Wysłanie polecenia SQL:

```
ResultSet rs = stmt.executeQuery(
                    "Select * from users");
```

Zwracany jest obiekt ResultSet, czyli zbiór rekordów powstały w wyniku zapytania

Rezultat zapytania

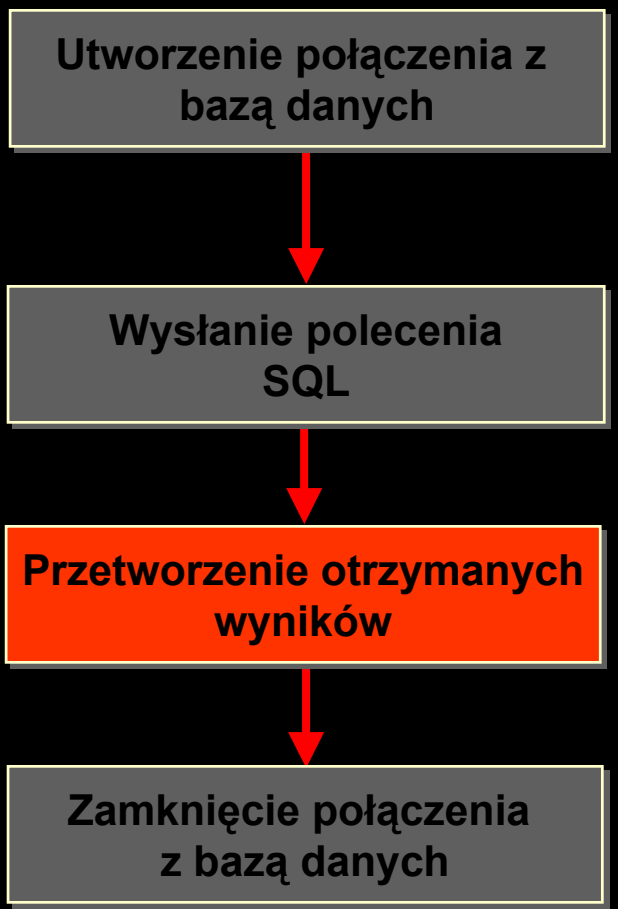
W zależności od typu polecenia SQL możliwe są do uzyskania różne rezultaty wykonanych operacji `executeXXX()`

Dla `XXX = Query`:

```
executeQuery("Select * from users")
```

otrzymuje się wszystkie rekordy będące wynikiem zapytania polecenia SELECT

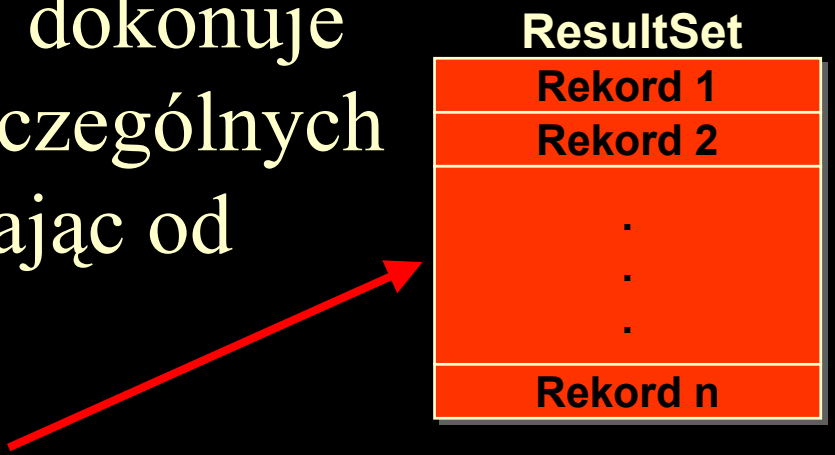
java.sql




```
<%@ page language="java" import="java.sql.*"%>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection dbConn =
DriverManager.getConnection("jdbc:odbc:portal",
"username", "password");

Statement stmt = dbConn.createStatement();
ResultSet rs = stmt.executeQuery(
                        "Select * from users");

if (rs != null) {
 while(rs.next()) {
        out.println(rs.getString("login"));}
}

stmt.close();
dbConn.close();
%>
```

Rezultat zapytania

Za pomocą metody `next()` dokonuje się przesuwania po poszczególnych rekordach bazy (poczynając od pierwszego rekordu)

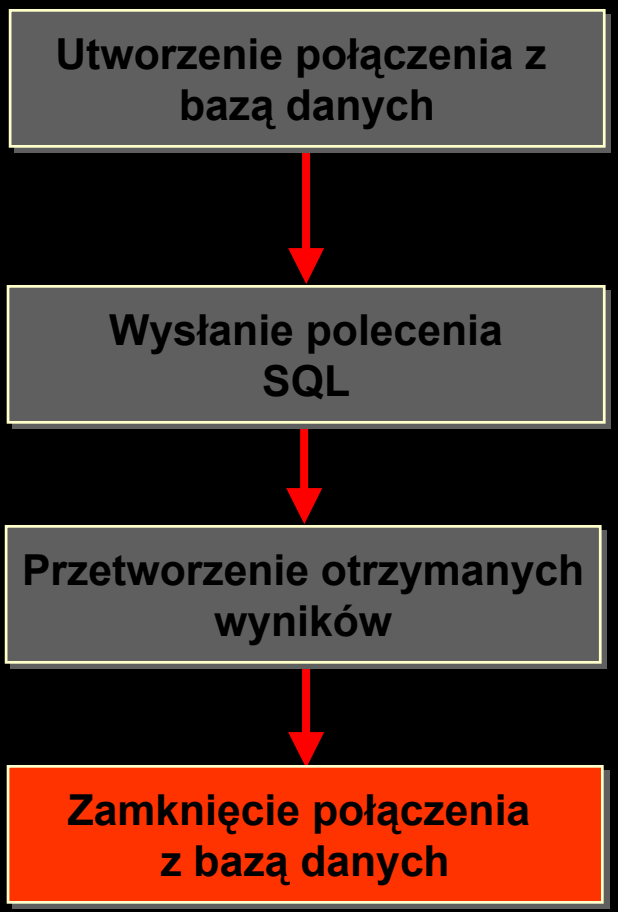



```
if (rs != null) {
  while(rs.next()) {
    out.println(rs.getString("login"));}
}
```

Pobieranie pól rekordu za pomocą metody `getString()`

null - koniec listy rekordów lub pusty ResultSet)

java.sql




```
<%@ page language="java" import="java.sql.*"%>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection dbConn =
DriverManager.getConnection("jdbc:odbc:portal",
"username", "password");

Statement stmt = dbConn.createStatement();
ResultSet rs = stmt.executeQuery(
                        "Select * from users");

if (rs != null) {
 while(rs.next()) {
        out.println(rs.getString("login"));}
}

stmt.close();
dbConn.close();
%>
```

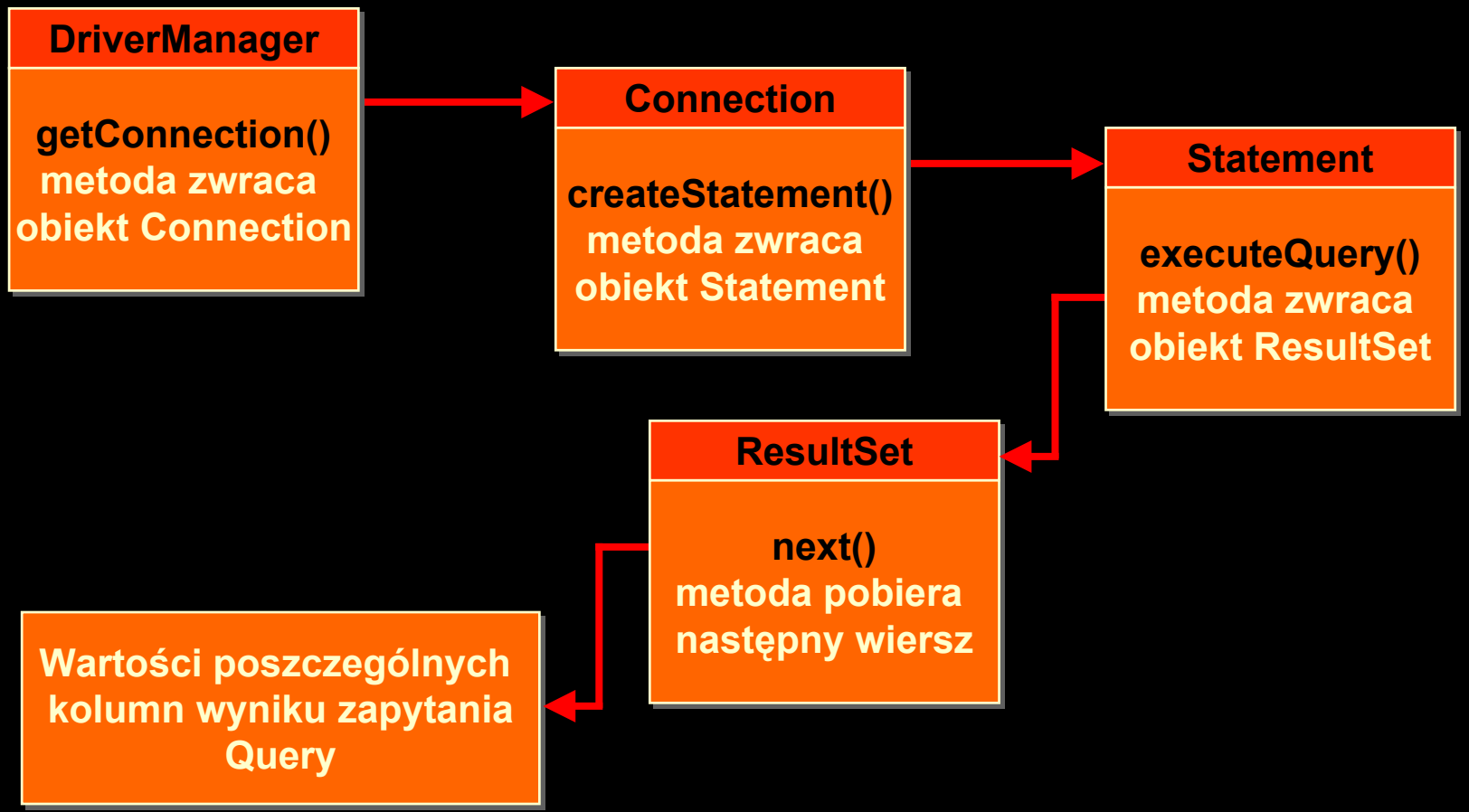
Zamknięcie połączenia z bazą danych

Zamknięcie połączenia bazy danych wywołuje się poprzez wywołanie metody `close()` dla obiektu Statement i Connection

```
stmt.close();
dbConn.close();
```

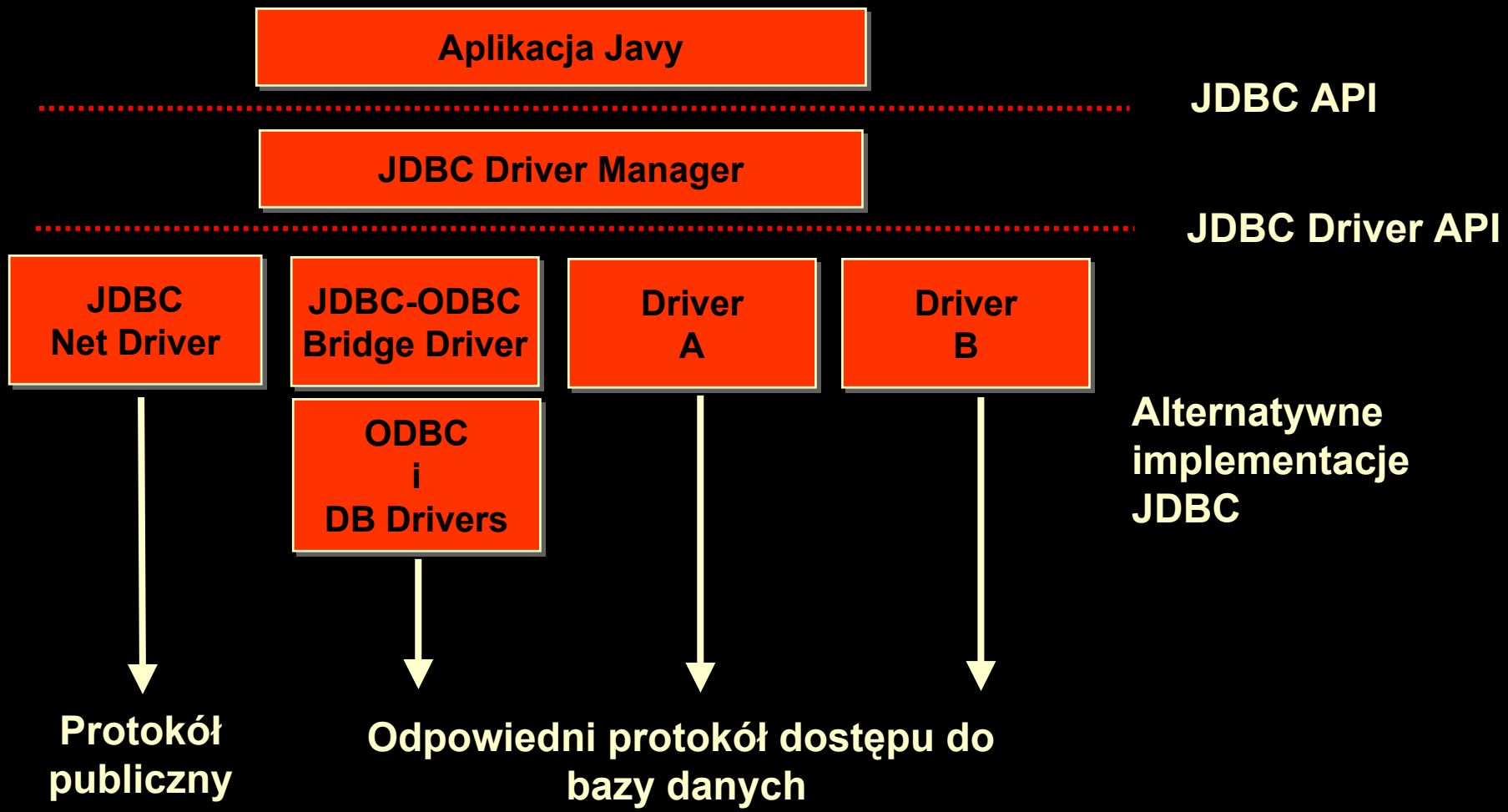
Connection - połączenie z konkretną bazą danych
Statement - kontener do wywoływania poleceń SQL

JDBC



Obiekty i metody biorące udział w połączeniu JDBC

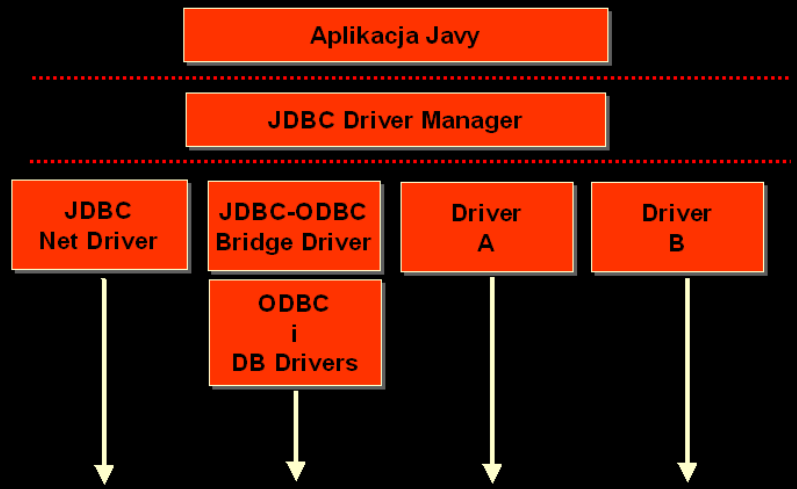
JDBC


Aplikacja Javy
JDBC API
JDBC Driver Manager
JDBC Driver API
JDBC
Net Driver
JDBC-ODBC
Bridge Driver
Driver
A
Driver
B
ODBC
i
DB Drivers
Alternatywne
implementacje
JDBC
Protokół
publiczny
Odpowiedni protokół dostępu do
bazy danych

Komunikacja sterownika

Proces połączenia z bazą danych:

- wywołanie sterownika JDBC z kodu Javy



- odszukanie odpowiedniego sterownika przez mechanizm JDBC DriverManager
- połączenie z użyciem odpowiedniego sterownika (dedykowanego dla konkretnej bazy danych)

JDBC drivers

Typy sterowników:

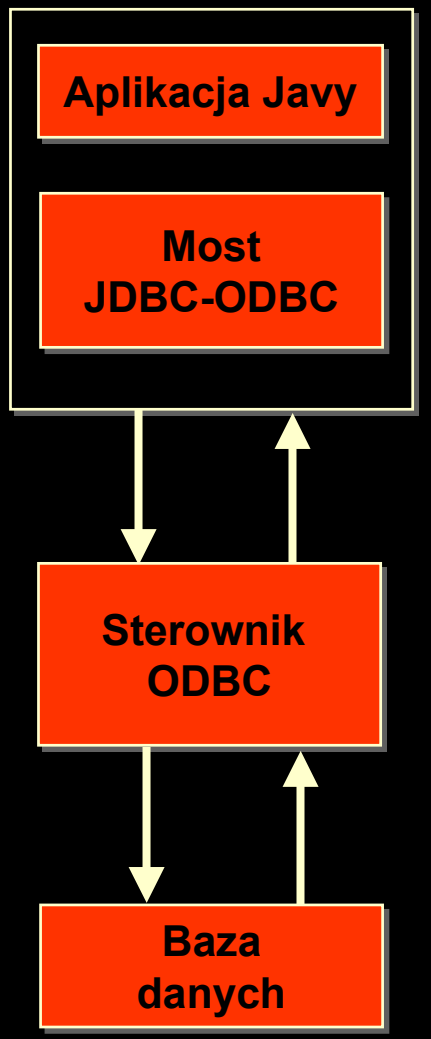
- typ 1: most JDBC-ODBC i sterownik ODBC
- typ 2: Native-API - sterownik napisany częściowo w Javie
- typ 3: JDBC-Net - sterownik napisany w całości w Javie
- typ 4: Native-protocol - sterownik napisany w całości w Javie

Most JDBC-ODBC

Typ1: most JDBC-ODBC i sterownik ODBC

Sterownik wchodzi w skład standardowego JDK

Dostęp do baz danych za pośrednictwem ODBC (Open DataBase Conectivity)



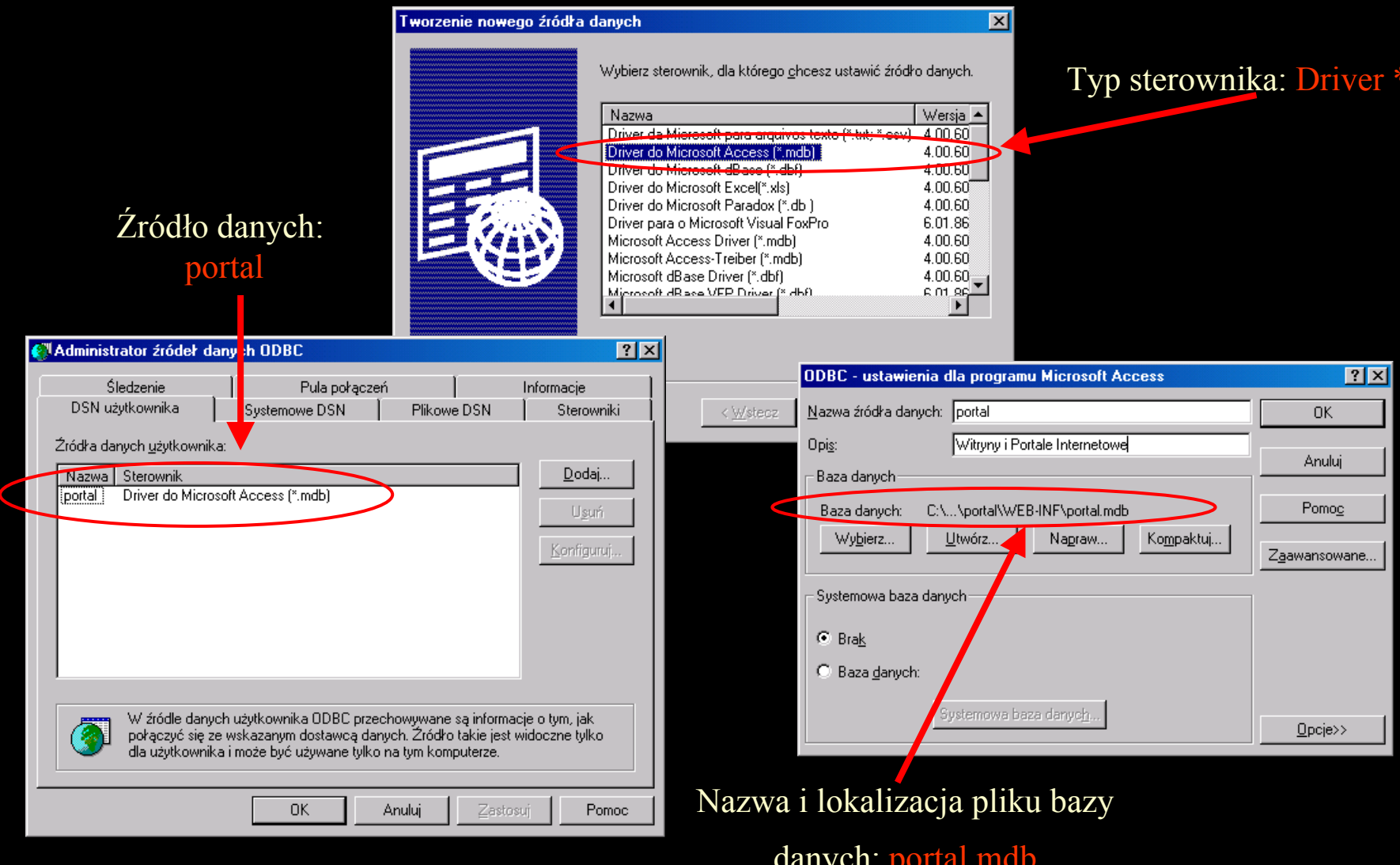
Most JDBC-ODBC

Komunikacja poprzez ODBC (Open DataBase Conectivity)

- komunikacja z bazami danych nie posiadających własnych sterowników JDBC (obniżenie wydajności)
- standardowy sterownik wchodzący w skład JDK

```
sun.jdbc.odbc.JdbcOdbcDriver
```

Konfiguracja nowego źródła danych ODBC



Most JDBC-ODBC

sun.jdbc.odbc.JdbcOdbcDriver

```
<%@ page errorPage=”error.jsp" %>
<%@ page import="java.sql.*" %>
<html>
<body>
<%
    // driver typ 1 (jdbc-odbc bridge)
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    String url="jdbc:odbc:portal";
    Connection conn = DriverManager.getConnection(
          url, "userid", "password” );

    String query = "select * from TABELA";
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(query);
%>
```

Most JDBC-ODBC

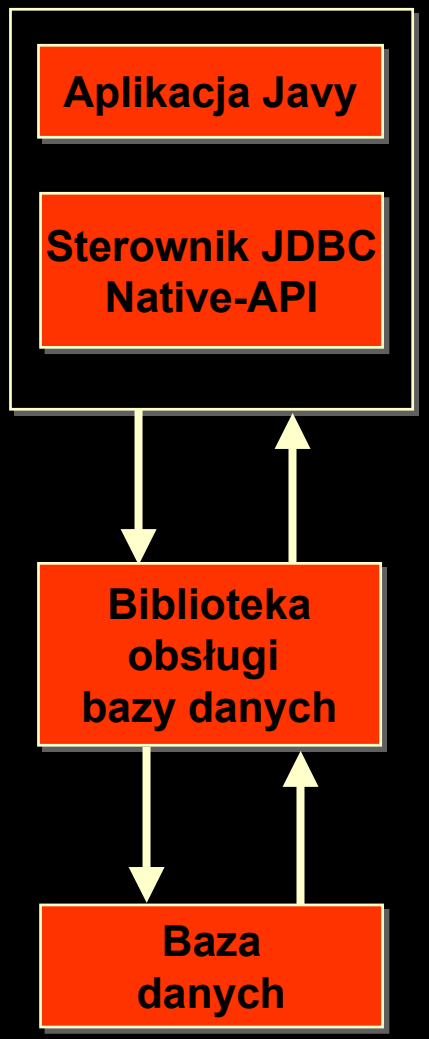
```
    <table border="1">
      <tr><td>Nazwisko</td><td>Imię</td></tr>

<%  while ( rs.next() ) { %>
      <tr>
         <td><%= rs.getString(2) %></td>
         <td><%= rs.getString(3) %></td>
      </tr>
<%  } // end of while
    rs.close();
    conn.close();
%>
    </table>
</body>
</html>
```

Native-API

Typ2: Native-API - sterownik napisany częściowo w Javie

Sterownik konwertuje polecenia JDBC do postaci zrozumiałej przez system zarządzana bazą danych (DBMS)

- dodatkowe programy po stronie klienta
- bezpośrednia komunikacja z bazą danych

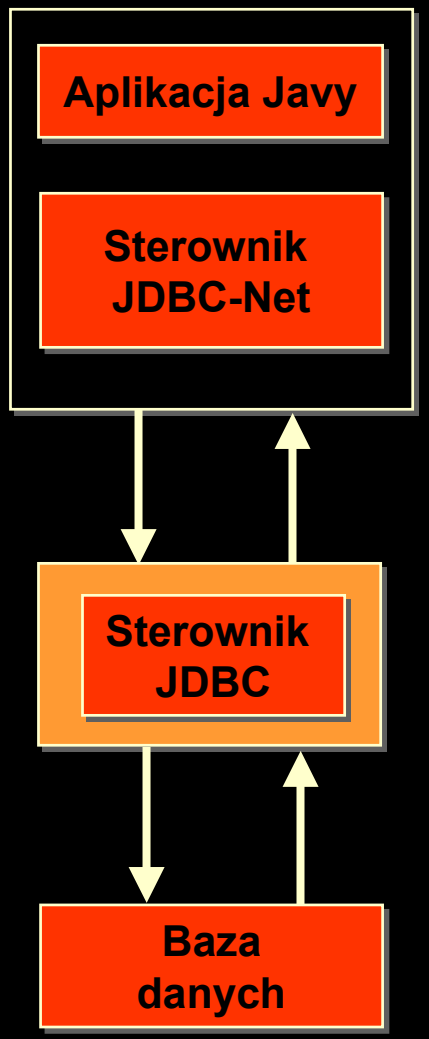


JDBC-Net

Typ3: JDBC-Net - sterownik napisany w całości w Javie

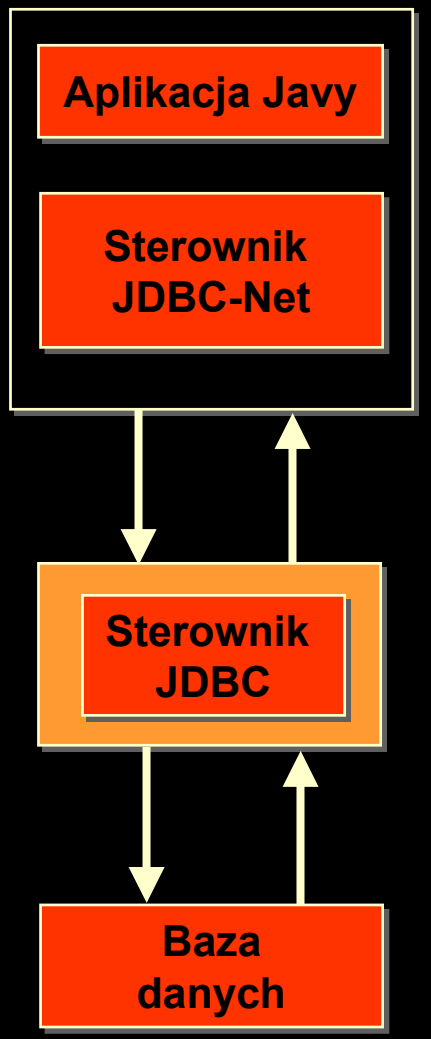
Sterownik oparty jest na 3 poziomowym modelu dostępu do bazy danych

Elastyczność zmiany typu bazy danych bez modyfikacji klienta



JDBC-Net

- tłumaczenie poleceń JDBC na niezależny protokół od bazy danych (poziom pośredni)
- zamian poleceń protokołu na polecenia zrozumiałe przez DBMS
- Zwrot rezultatów zapytania poprzez serwer poziomu pośredniego

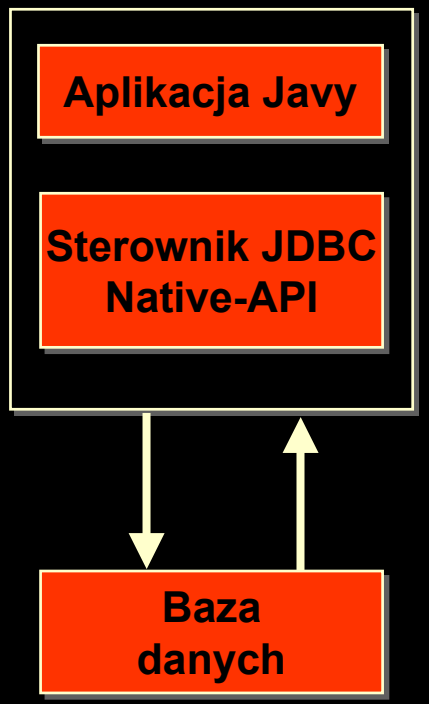


Native-protocol

Typ4: Native-protocol - sterownik napisany w całości w Javie

Bezpośrednia komunikacja z bazą danych

– brak pośrednich translacji (duża wydajność)



JDBC API

API JDBC składa się z dwóch bibliotek:

- JDBC 2.0 core API (standardowo dołączona do JDK)
- JDBC 2.0 extension API (rozszerzenie biblioteki standardowej)

Możliwość ciągłego rozwoju biblioteki JDBC, przy zachowaniu standardowych narzędzi dostarczanych ze środowiskiem JAVA

Interfejsy wyższego rzędu

JDBC - interfejs niskiego poziomu wywołujący bezpośrednio polecenia SQL

Interfejsy wyższego rzędu:

- mieszanie elementów SQL i Javy (JSQL)
- odwzorowanie tabel bazy danych jako klasy java (każdy rekord staje się obiektem)

JDBC w architekturze klient-server

Architektura klient-server:

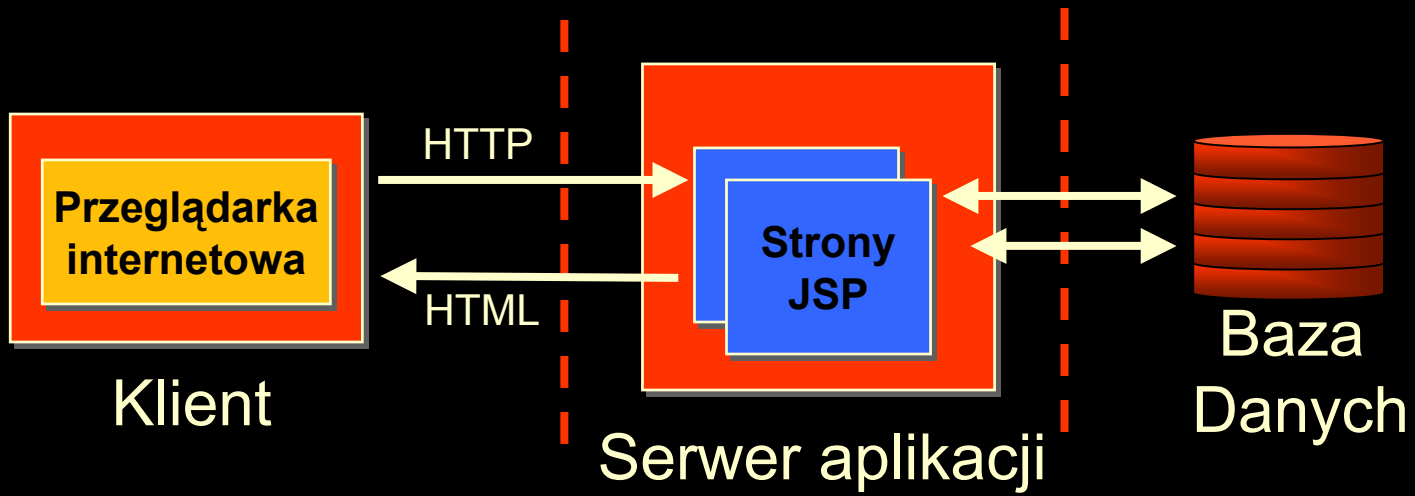
- zmiana klienta wymaga znajomości formatu baz danych
- zmiana bazy danych (modyfikacja) wymaga ingerencji w kod klienta

JDBC w architekturze wielowarstwowej N-Tier

Wprowadzenie architektury wielowarstwowej umożliwia odseparowanie bazy danych od klienta:

- tworzenie różnych klientów bez zagłębianie się w szczegóły wymiany danych z bazą danych
- łatwość modyfikacji baz danych (rozproszenie)

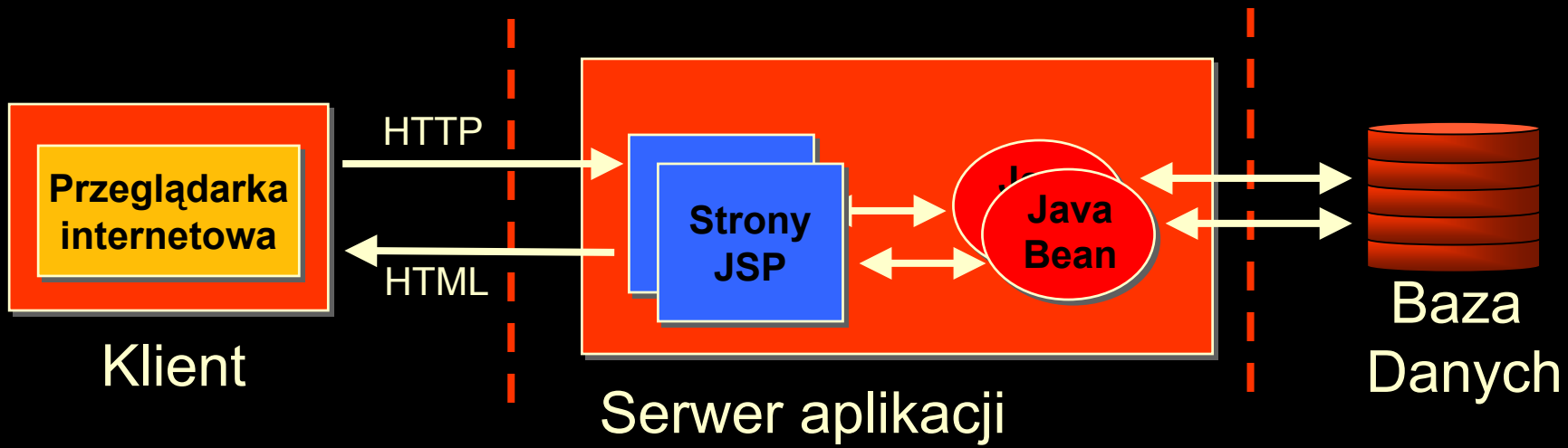
Bazy danych i JSP



Integrowanie obsługi JDBC wewnątrz JSP utrudnia modyfikację widoków i zmianę systemu baz danych (integracja danych z prezentacją).

Możliwość uproszczenia zażądzania bazami danych poprzez zastosowanie biblioteki znaczników TagLibs

Bazy danych i JavaBean



Użycie komponentów JavaBean do separacji bazy danych od stron JSP umożliwia łatwiejszą modyfikacje warstwy danych i warstwy prezentacji

Przykład

Zadanie:

- stworzenie zapytania SQL do budowy tabli przykładowej bazy danych (w oparciu o MySQL)
- stworzenie pośredniej warstwy za pomocą komponentów JavaBean (odwzorowanie rekordów bazy danych jako obiekty)
- stworzenie plików JSP warstwy prezentacji (realizacja wprowadzania danych, ich wyświetlania i obsługi błędów)

Przykład

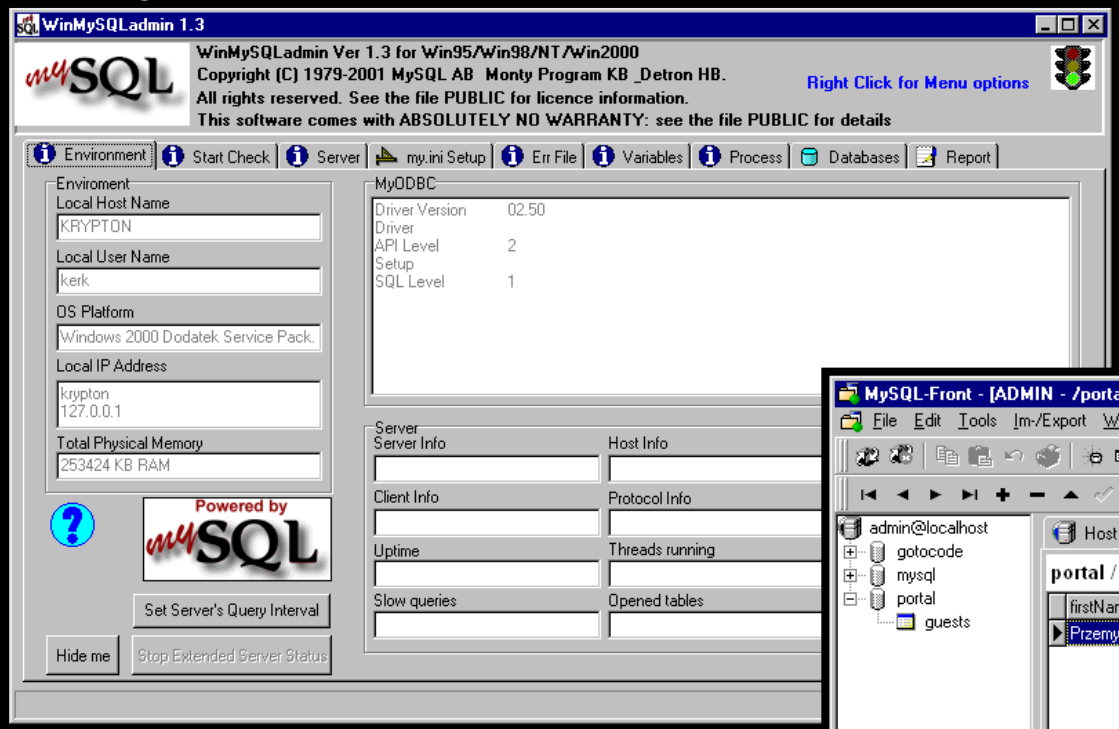
Pliki:

- users.sql (tabela bazy danych)
- UsersBean.java (klasa JavaBean będąca odwzorowaniem rekordu bazy danych)
- UsersDataBean.java (klasa JavaBean pośrednicząca w pobieraniu i wstawianiu danych do bazy danych)
- users_view.jsp (plik JSP wyświetlający zawartość tabeli bazy danych)
- users_login.jsp (plik JSP do wprowadzania nowego rekordu do bazy danych)

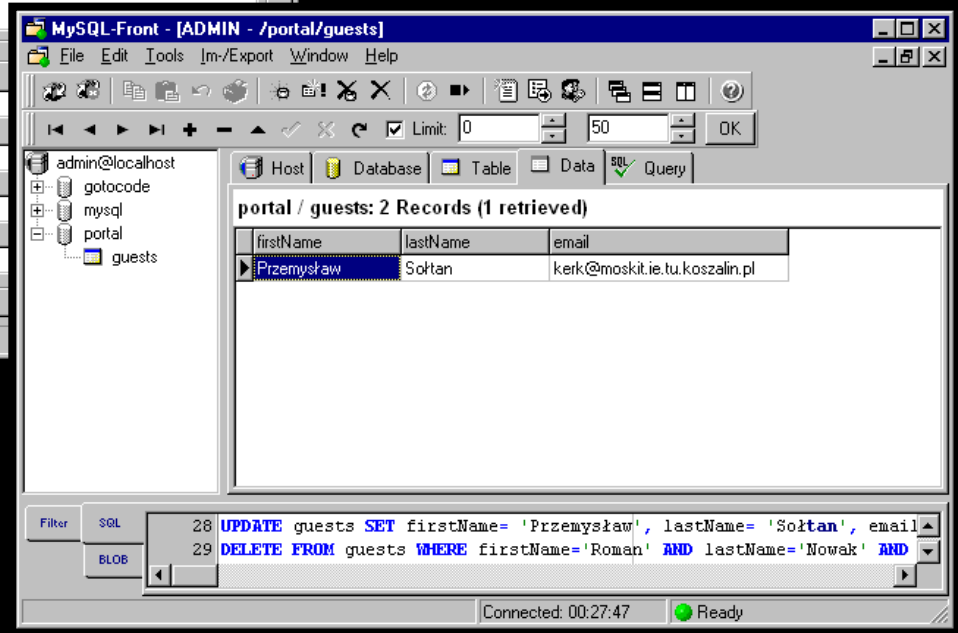
błędu)

MySQL

WinMySQLadmin

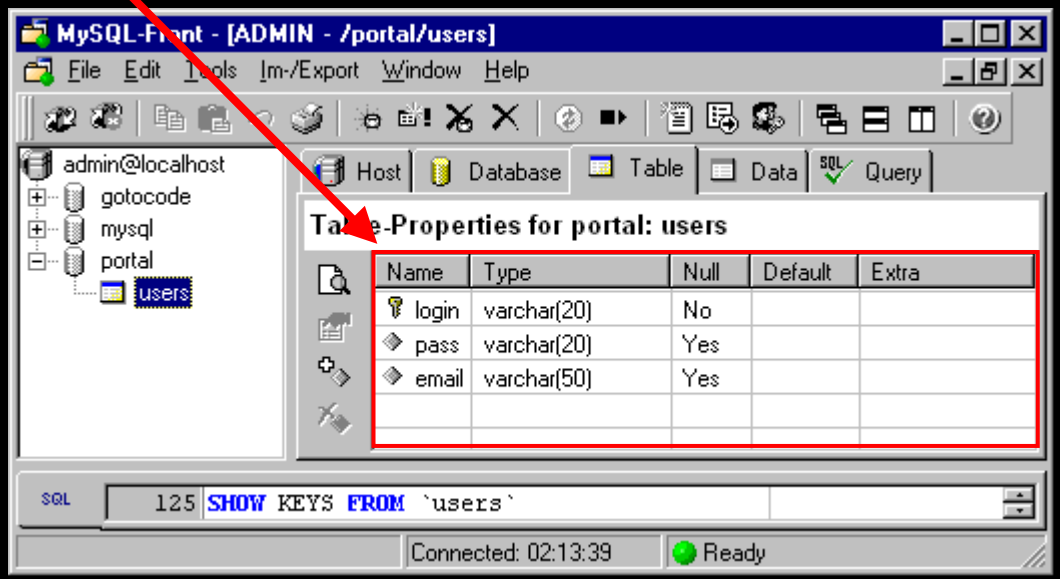


MySQL-Front



users.sql

```
CREATE TABLE users (
  login varchar(20) NOT NULL default '',
  pass varchar(20) default NULL,
  email varchar(50) default NULL,
  PRIMARY KEY  (login)
);
```



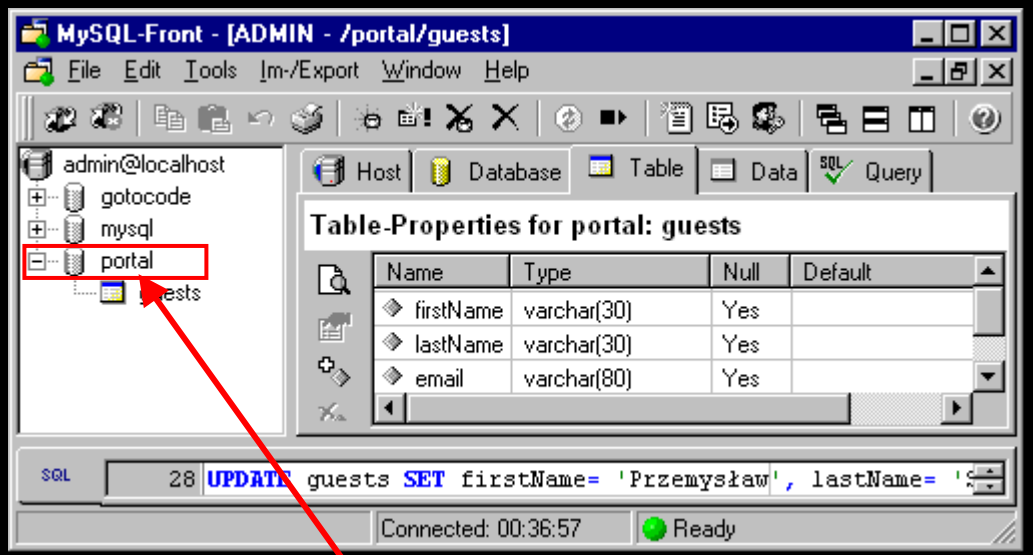
Sterownik JDBC dla MySQL

mm.mysql.jdbc-2.0.4

Sterownik: org.gjt.mm.mysql.Driver

URL bazy danych: jdbc:mysql://localhost/portal

Instalacja sterownika polega na umieszczeniu pliku mm.mysql.jdbc-2.0.4-bin.jar w katalogu lib serwera TOMCAT




```
Class.forName("org.gjt.mm.mysql.Driver");
DriverManager.getConnection("jdbc:mysql://localhost/portal","login","passwd");
```

UsersBean.java

```
package portal;
import java.io.*;

public class UsersBean
                extends Object {

  private String login;
  private String pass;
  private String email;

  public UsersBean() { }

  public void setLogin(String login) { this.login = login; }
  public String getLogin() { return login; }

  public void setPass(String pass) { this.pass = pass; }
  public String getPass() { return pass; }

  public void setEmail(String email) { this.email = email; }
  public String getEmail() { return email; }
}
```

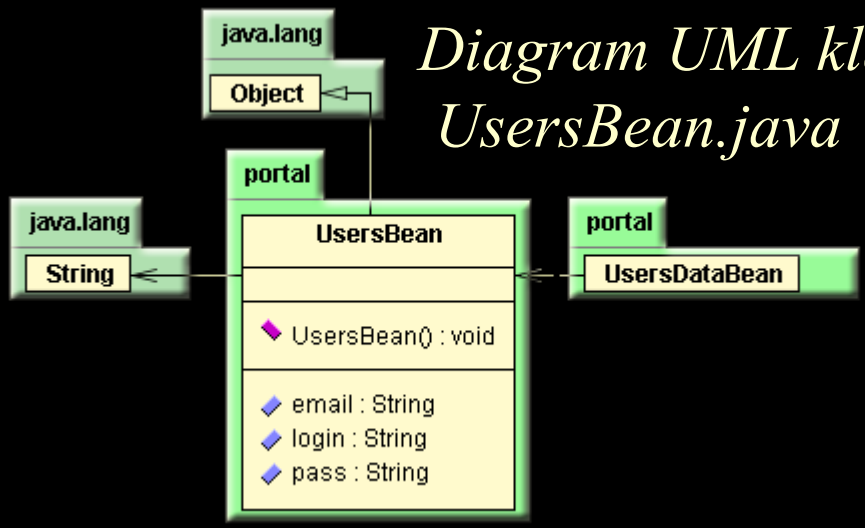


Diagram UML klasy UsersBean.java

UsersDataBean.java 1/5

```
package portal;

import java.io.*;
import java.sql.*;
import java.util.*;

public class UsersDataBean extends Object {

  private Connection conn;
  private PreparedStatement addRecord, getRecord, deleteRecord;
```

UsersDataBean.java 2/5

```
public UsersDataBean() throws Exception {
  Class.forName( "org.gjt.mm.mysql.Driver" );
  conn = DriverManager.getConnection(
           "jdbc:mysql://localhost/portal","login","password");

  getRecord = conn.prepareStatement(
              "SELECT login,pass,email FROM users");
  addRecord =  conn.prepareStatement(
              "INSERT INTO users (login,pass,email) VALUES (?,?,?)");
  deleteRecord = conn.prepareStatement(
              "DELETE * FROM users WHERE login = ?");
}
```

UsersDataBean.java 3/5

```
public List getUserList() throws SQLException
{
   List userList = new ArrayList();

   ResultSet results = getRecord.executeQuery();

   while ( results.next() ) {
      UsersBean user = new UsersBean();
      user.setLogin( results.getString(1) );
      user.setPass( results.getString(2) );
      user.setEmail( results.getString(3) );
      userList.add(user);
   }
   return userList;
}
```

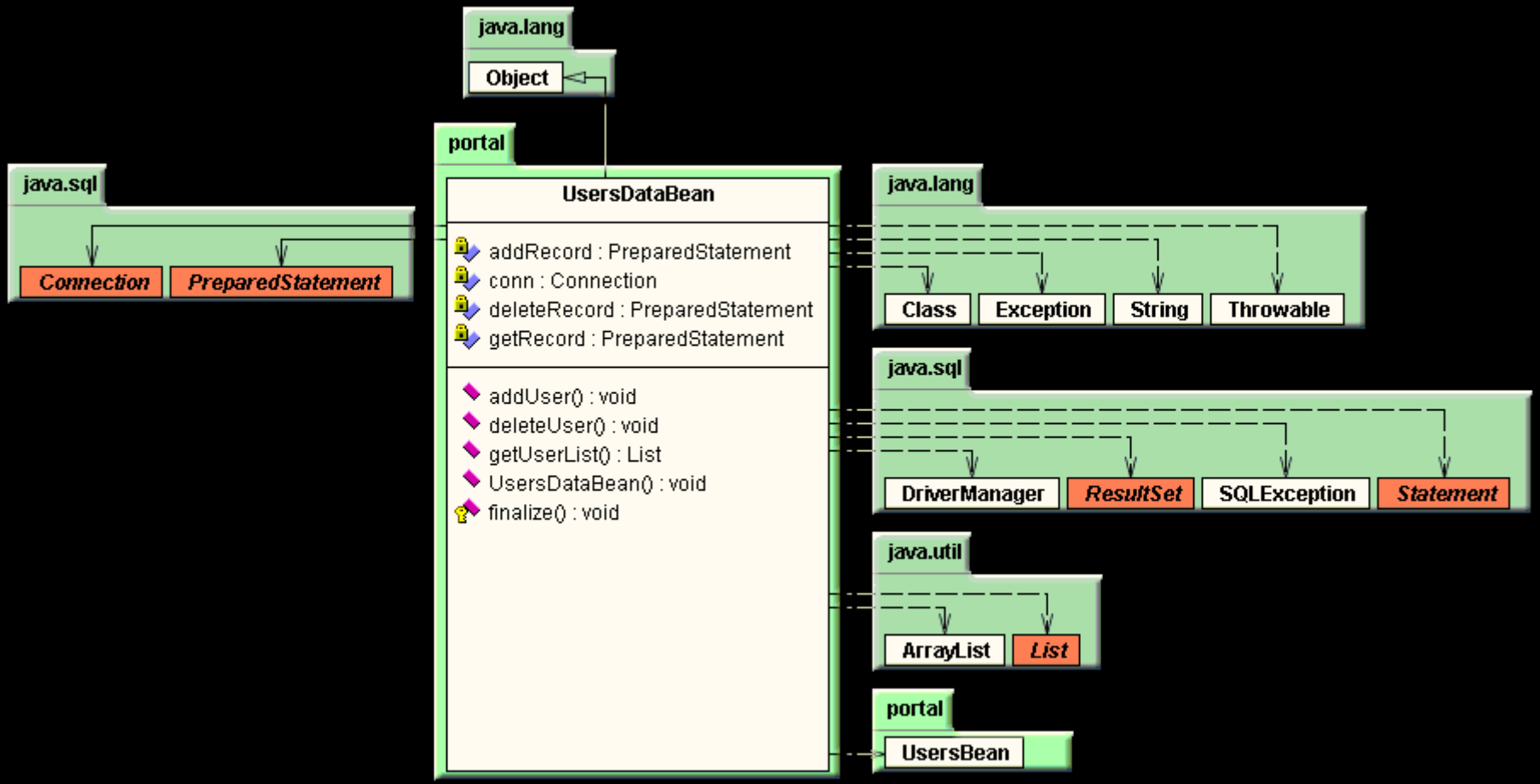
UsersDataBean.java 4/5

```
public void addUser( UsersBean user ) throws SQLException
{
    addRecord.setString(1,user.getLogin());
    addRecord.setString(2,user.getPass());
    addRecord.setString(3,user.getEmail());
    addRecord.executeUpdate();
}

public void deleteUser( UsersBean user ) throws SQLException
{
    deleteRecord.setString(1,user.getLogin());
    deleteRecord.executeUpdate();
}
```

UsersDataBean.java 5/5

```
    protected void finalize()
    {
        try {
            getRecord.close();
            addRecord.close();
            conn.close();
        }
        catch ( SQLException sqlException ) {
            sqlException.printStackTrace();
        }
    }

} //koniec klasy UsersDataBean
```

Diagram UML
UsersDataBean.java



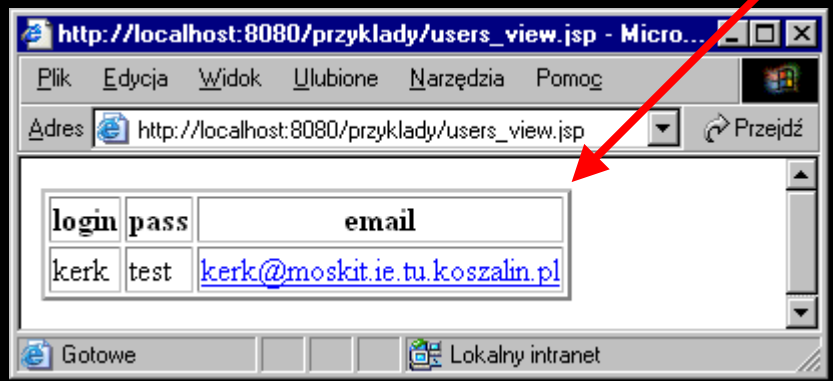
users_view.jsp 1/2

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<%@ page errorPage = ”users_error.jsp" %>
<%@ page import = "java.util.*,portal.*" %>

<jsp:useBean id="usersData" scope="request"
class="portal.UsersDataBean" />
<html>
 <head></head>
<body>
  <table border="2">
     <thead>
        <tr><th>login</th><th>pass</th><th>email</th></tr>
     </thead>
     <tbody>
       <%
          List userList = usersData.getUserList();
          Iterator iterator = userList.iterator();
```

users_view.jsp 2/2

```
        UsersBean user;

        while (iterator.hasNext() ) {
            user = (UsersBean) iterator.next();
        %>
            <tr>
                <td><%= user.getLogin() %></td>
                <td><%= user.getPass() %></td>
                <td><a href = "mailto:<%= user.getEmail() %>">
                <%= user.getEmail() %></a></td>
            </tr>

        <% } %>
    </tbody>
</table>
</body>
</html>
```

login	pass	email
<%= user.getLogin() %>	<%= user.getPass() %>	<%= user.getEmail() %>



users_login.jsp 1/2

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<%@ page errorPage = "users_error.jsp" %>

<jsp:useBean id = "user" scope = "page" class = "portal.UsersBean" />
<jsp:useBean id = "usersData" scope = "request"
                              class = "portal.UsersDataBean" />

<html>
  <head>
  </head>
<body>
  <jsp:setProperty name = "user" property = "*" />

   <% if ( user.getLogin() == null ||
           user.getPass() == null ||
           user.getEmail() == null ) {

   %>
```

users_login.jsp 2/2

```
        <form method = "POST" action = "users_login.jsp">
           <table>
              <tr>
                <td>Login<input type = "text" name = "login" /></td>
              </tr>
              <tr>
                <td>Pass<input type = "text" name = "pass" /></td>
              </tr>
              <tr>
                <td>Email <input type = "text" name = "email" /></td>
              </tr>
              <tr>
                <td><input type = "submit" value = "Submit" /></td>
              </tr>
           </table>
        </form>
    <% }  else { usersData.addUser(user); %>
        <jsp:forward page = "users_view.jsp" />
   <% } %>
</body>
</html>
```

users_error.jsp

```
<%@ page contentType="text/html; charset=windows-1250" %>

<%@ page isErrorPage = "true" %>
<%@ page import = "java.util.*" %>
<%@ page import = "java.sql.*" %>
<html>
  <head>
  </head>
<body>
 BŁĄD: <%= exception.getMessage() %>
</body>
</html>
```

Proces pobierania pierwszego rekordu z java beana

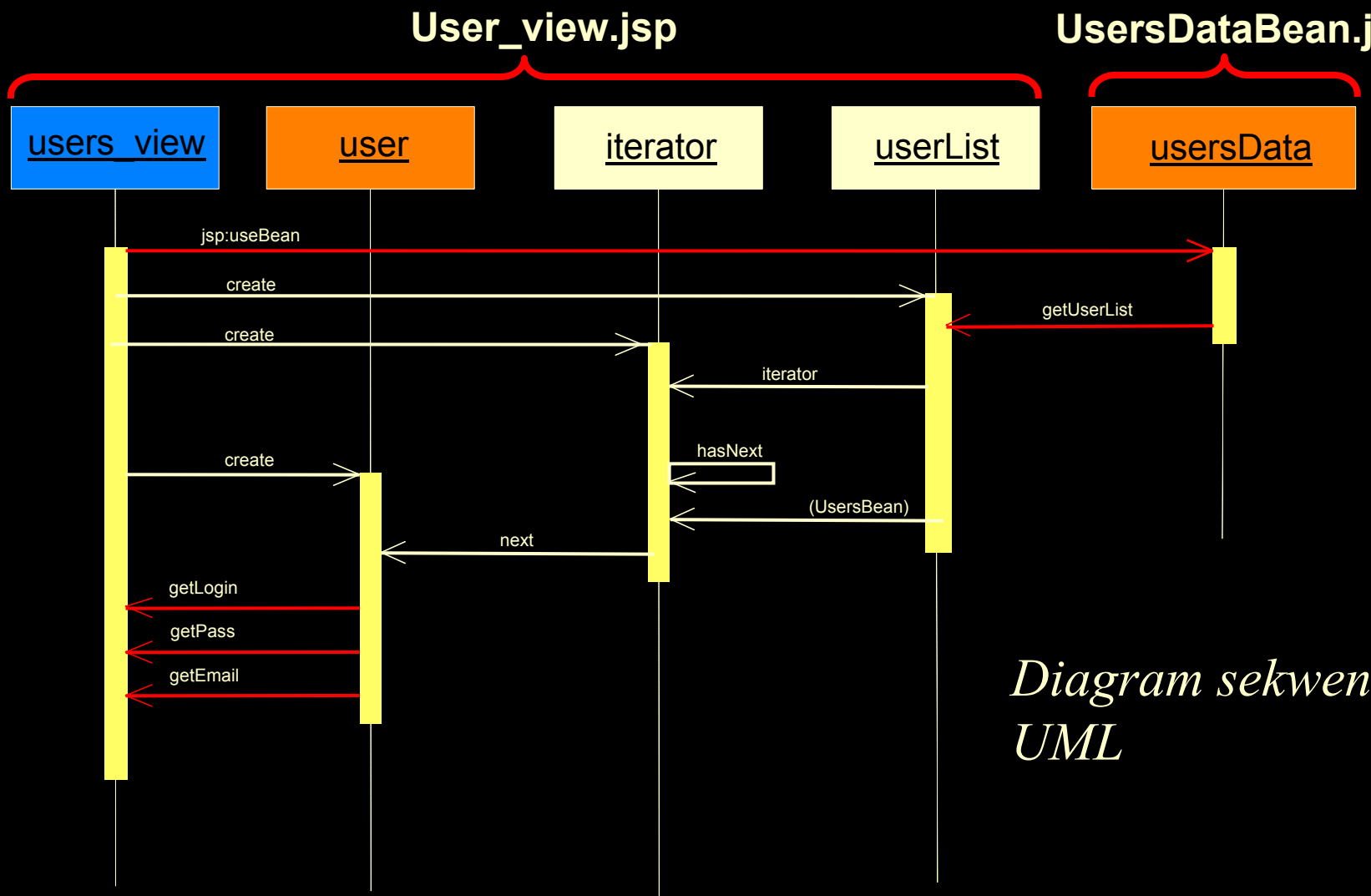
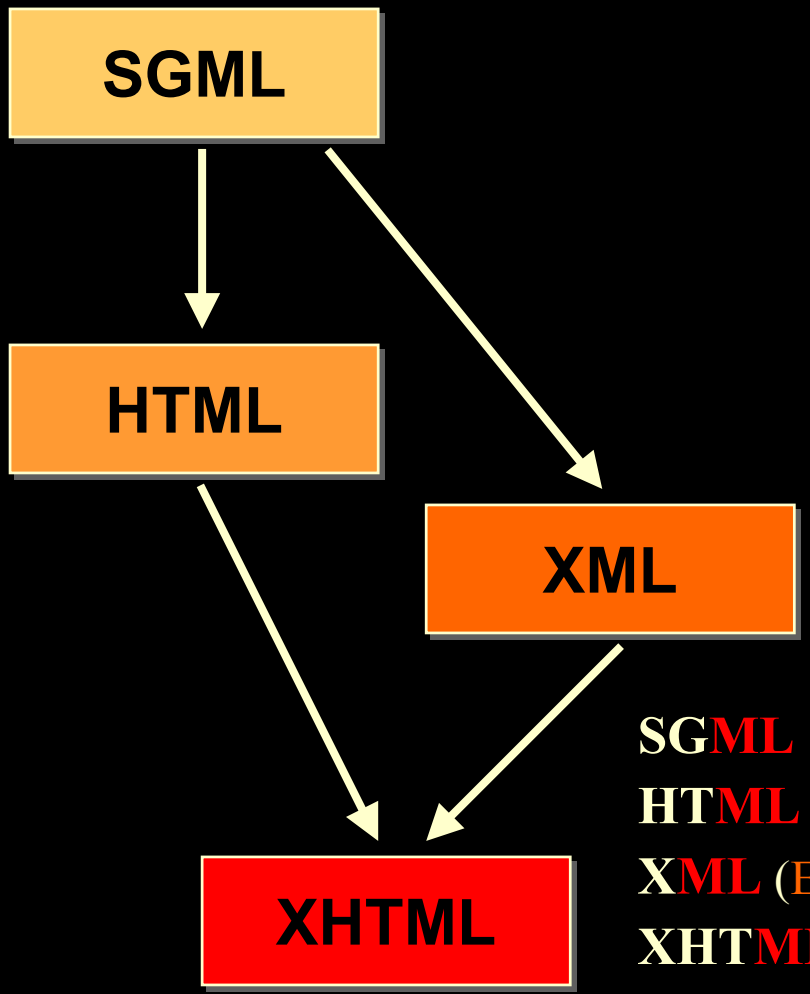


Diagram sekwencji UML

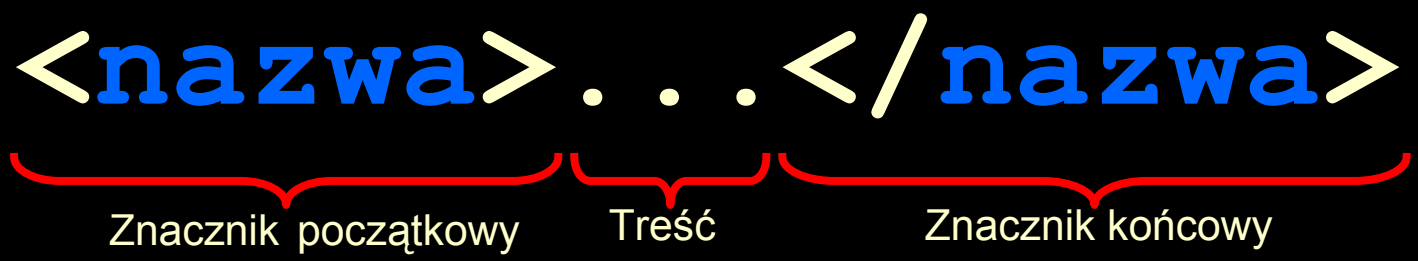
..ML

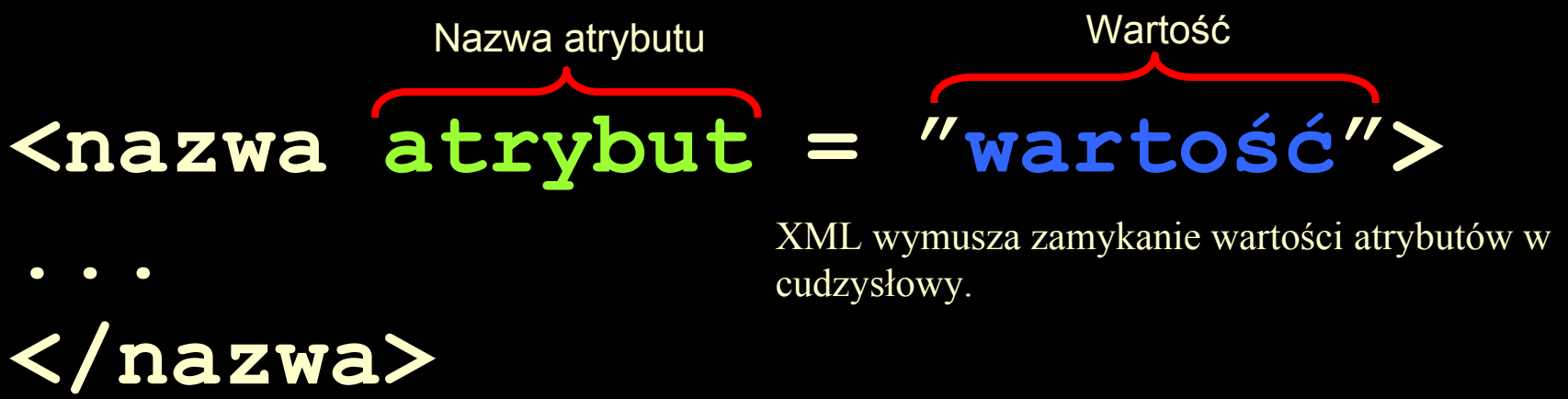
..ML - Markup Language



SGML (Standard Generalized Markup Language)
HTML (HyperText Markup Language)
XML (Extensible Markup Language)
XHTML (Extensible HTML)

Budowa znaczników

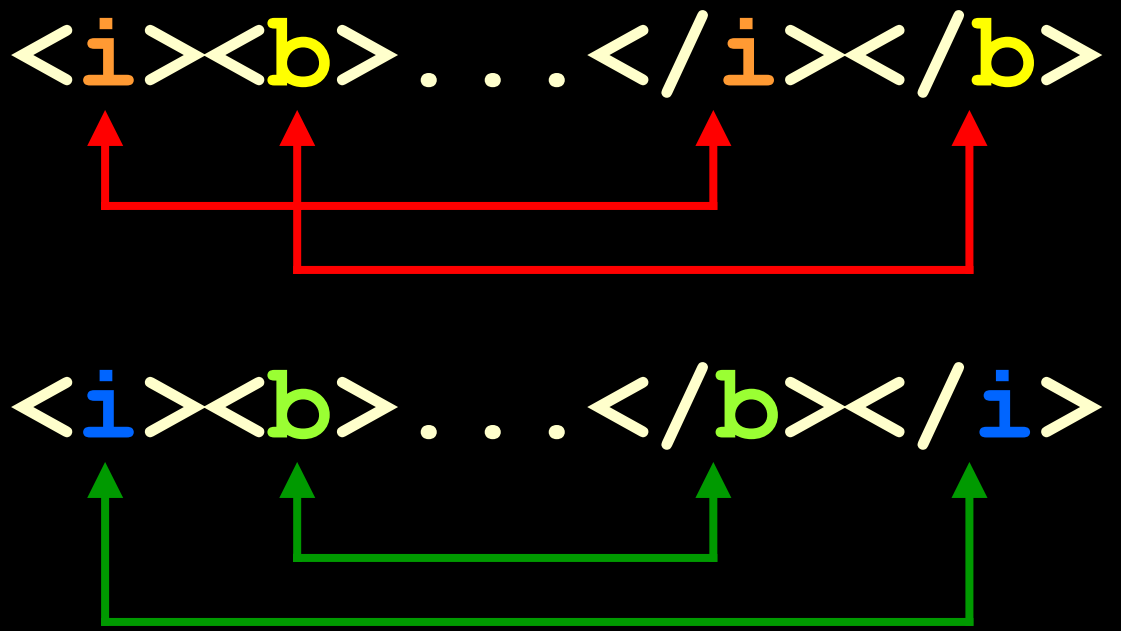




Reguły poprawnego formatowania dokumentów XML

- Istnienie jednego elementu głównego
- Stosowanie znaczników końcowych
- Prawidłowe zagnieżdżanie znaczników
- Znaczenie wielkości liter nazw znaczników
- Zamykanie wartości atrybutów w cudzysłowy
- Użycie dodatkowych encji (poza: &,<,>,”,’) wymaga definicji w DTD

Zagnieżdżanie znaczników



- Nie wolno zamykać elementu zewnętrznego, dopóki nie zostanie zamknięty element wewnętrzny

Wielkość liter nazw znaczników

```
<HTML>...</HTML>

<html>...</html>
```

- Znacznik zamykający powinien mieć taką samą nazwę jak znacznik otwierający (z uwzględnieniem wielkości liter)

Znacznik otwierający i zamykający

```
<znacznik>
```

Stosowanie tylko znaczników początkowych w XML'u zostało zabronione.

```
<znacznik></znacznik>
<znacznik/>
```

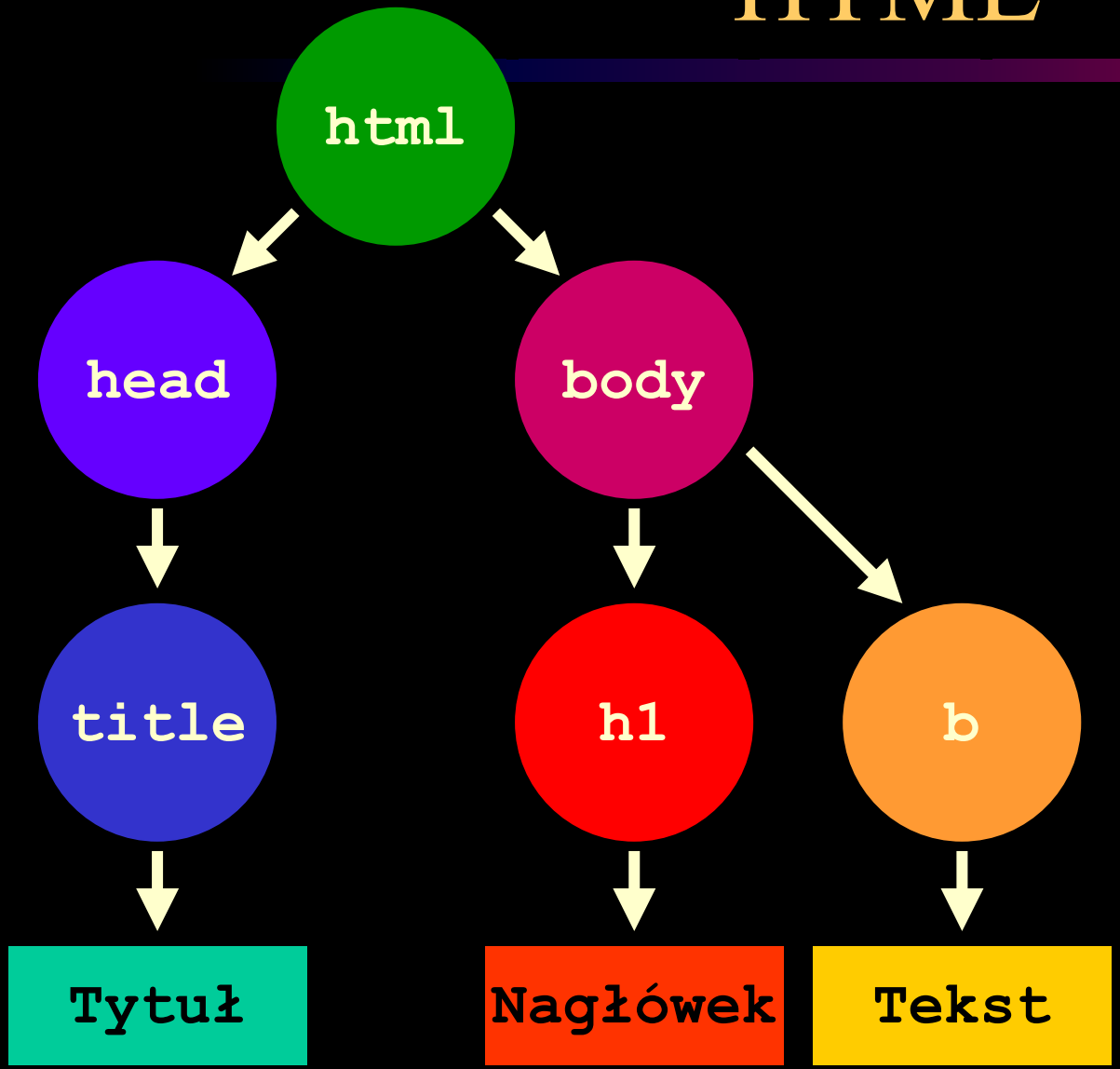
- Walidacja dokumentów
- Wzrost wydajności przetwarzania
- Mniejszy silnik (engine) przeglądarek do obsługi XML (w porównaniu z HTML)

Dodawanie komentarzy

```
<!--tekst komementarza-->
```

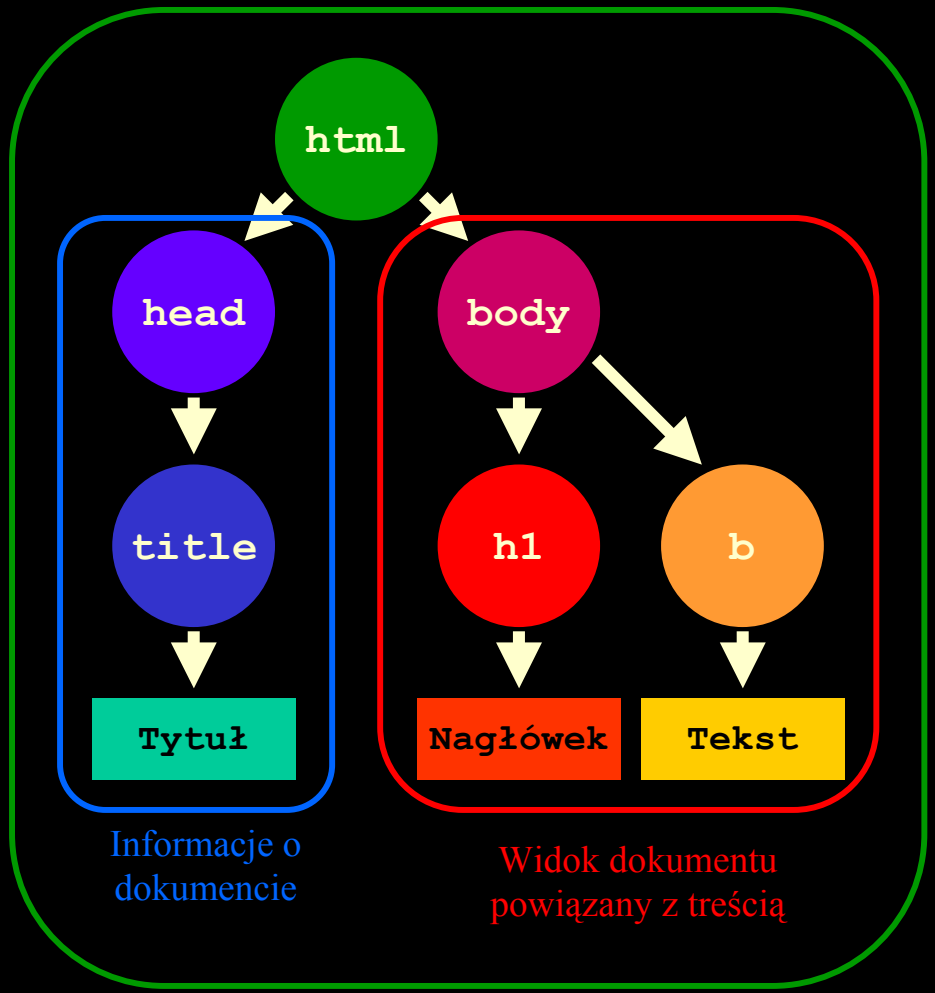
- Brak możliwości zagnieżdżania komentarzy

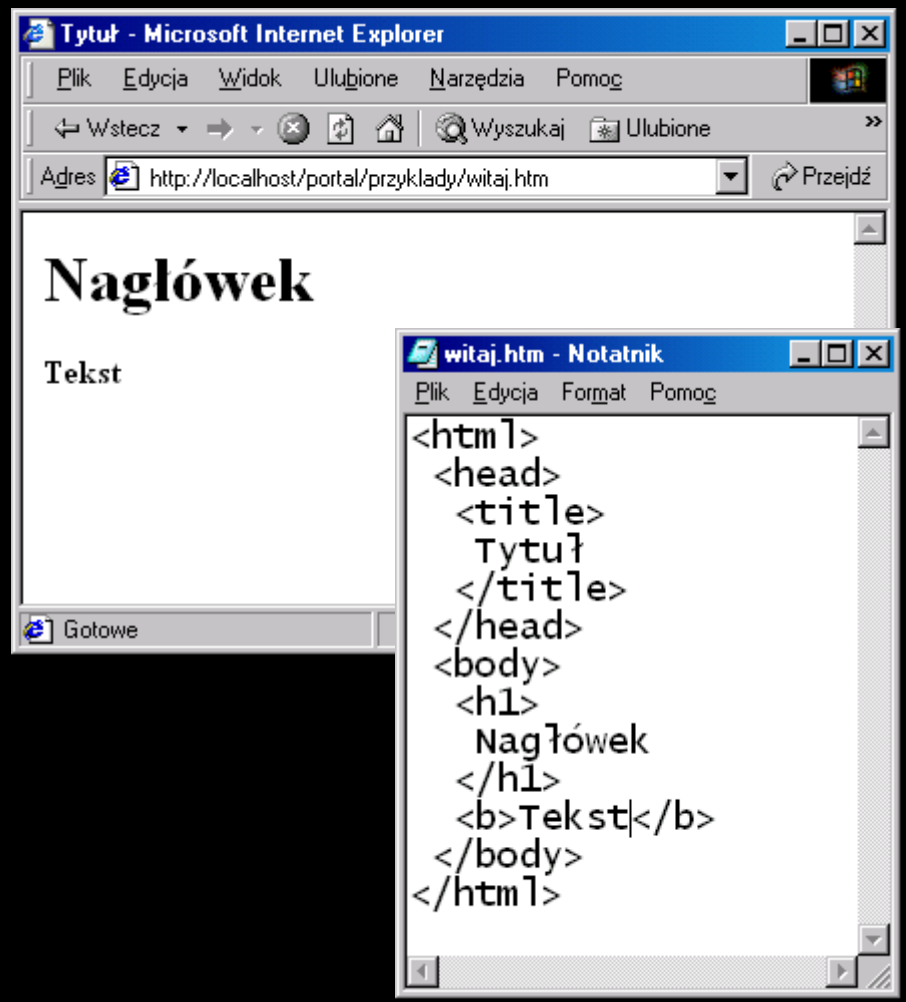
HTML



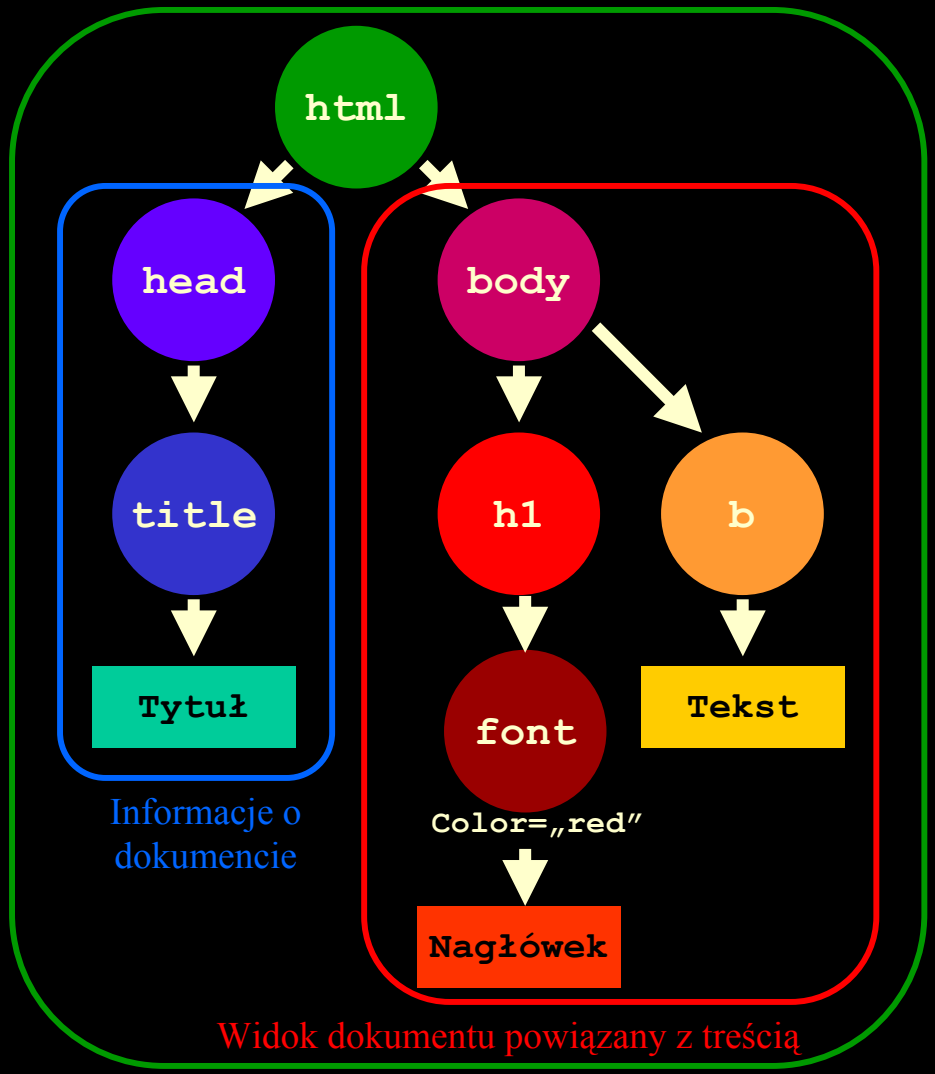

```
<html>
 <head>
  <title>
   Tytuł
  </title>
 </head>
 <body>
  <h1>
   Nagłówek
  </h1>
  <b>Tekst</b>
 </body>
</html>
```

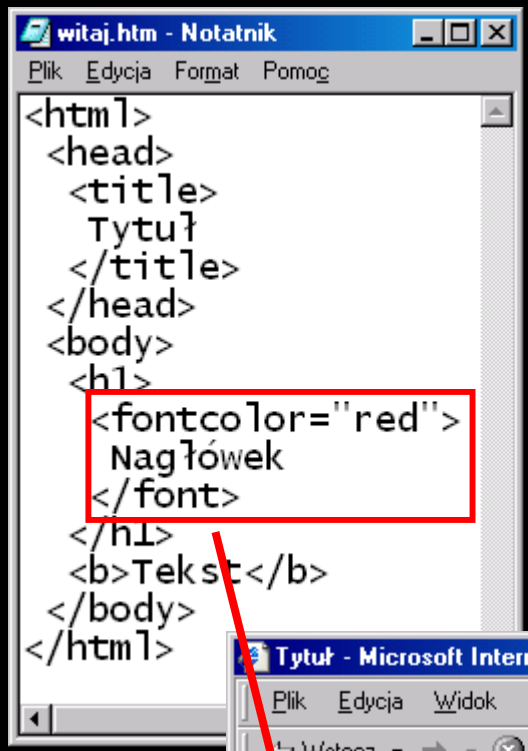
HTML

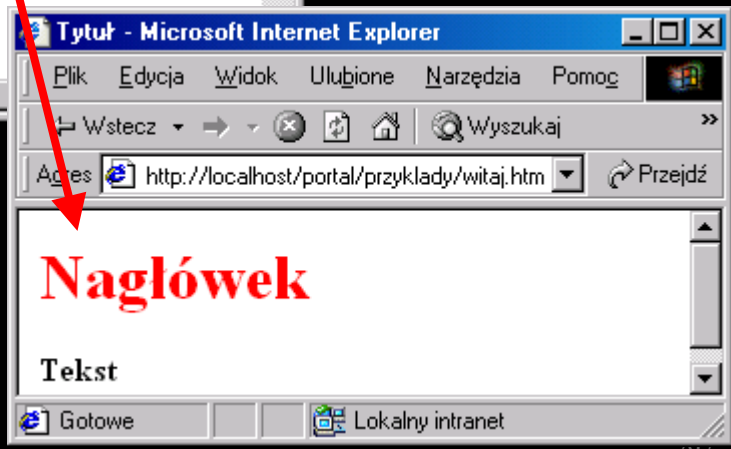

html
head
body
title
h1
b
Tytuł
Nagłówek
Tekst
Informacje o dokumencie
Widok dokumentu powiązany z treścią
Dokument HTML


Tytuł - Microsoft Internet Explorer
Plik Edycja Widok Ulubione Narzędzia Pomoc
Wstecz Wyszukaj Ulubione
Adres http://localhost/portal/przyklady/witaj.htm
Przejdź
Nagłówek
Tekst
Gotowe
witaj.htm - Notatnik
Plik Edycja Format Pomoc
<html>
<head>
<title>
Tytuł
</title>
</head>
<body>
<h1>
Nagłówek
</h1>
Tekst
</body>
</html>

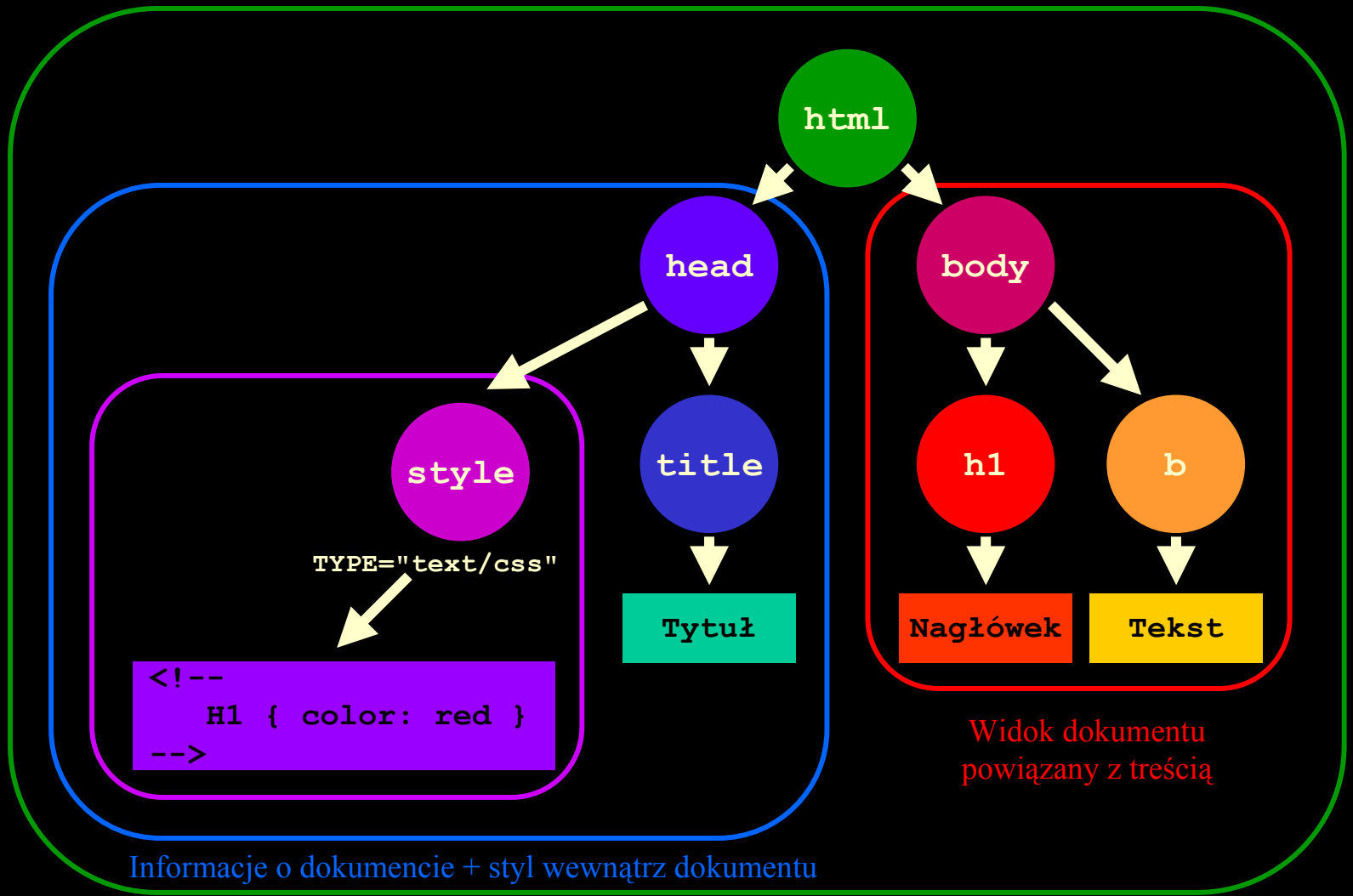
HTML - znacznik <font>



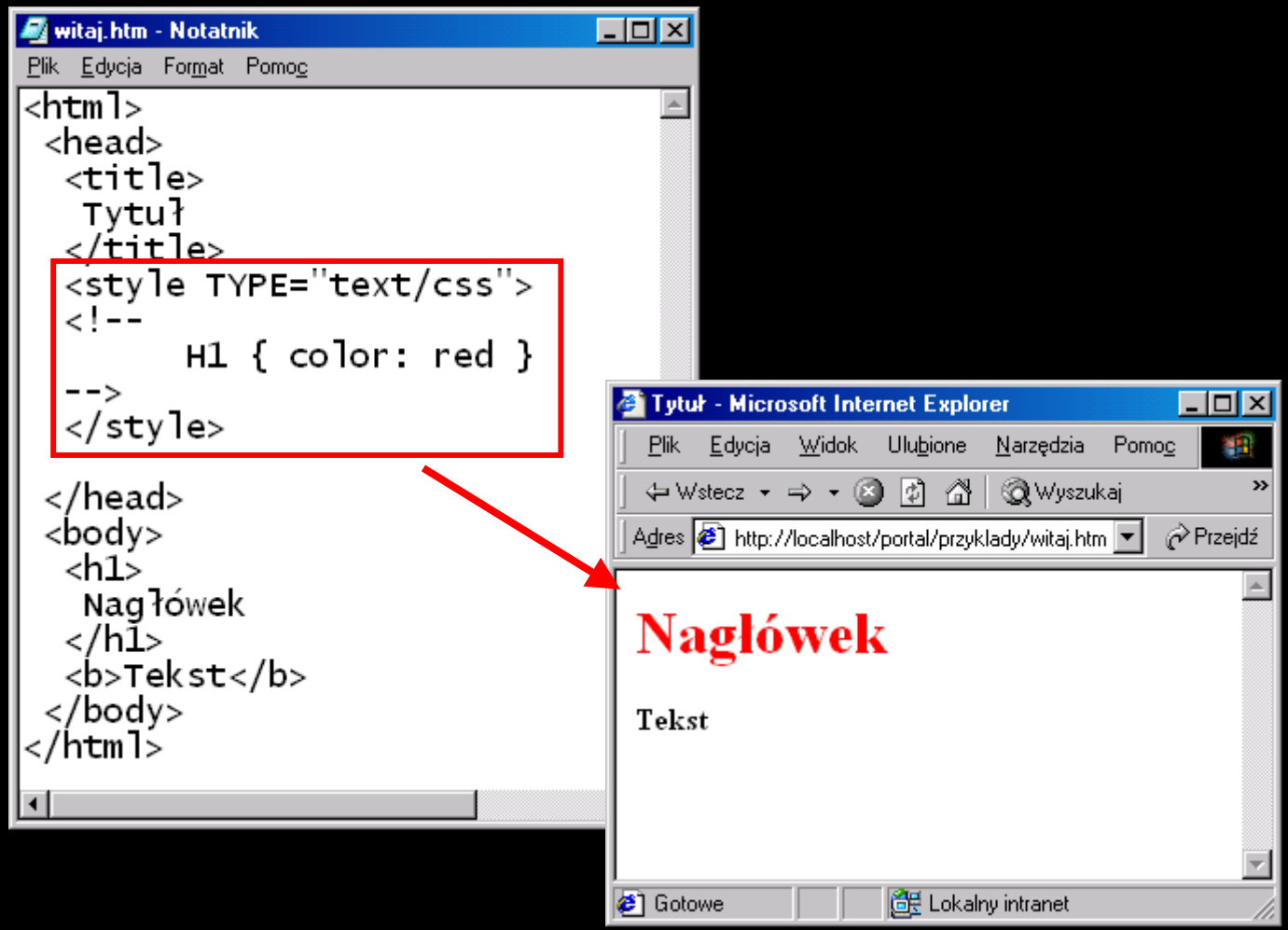




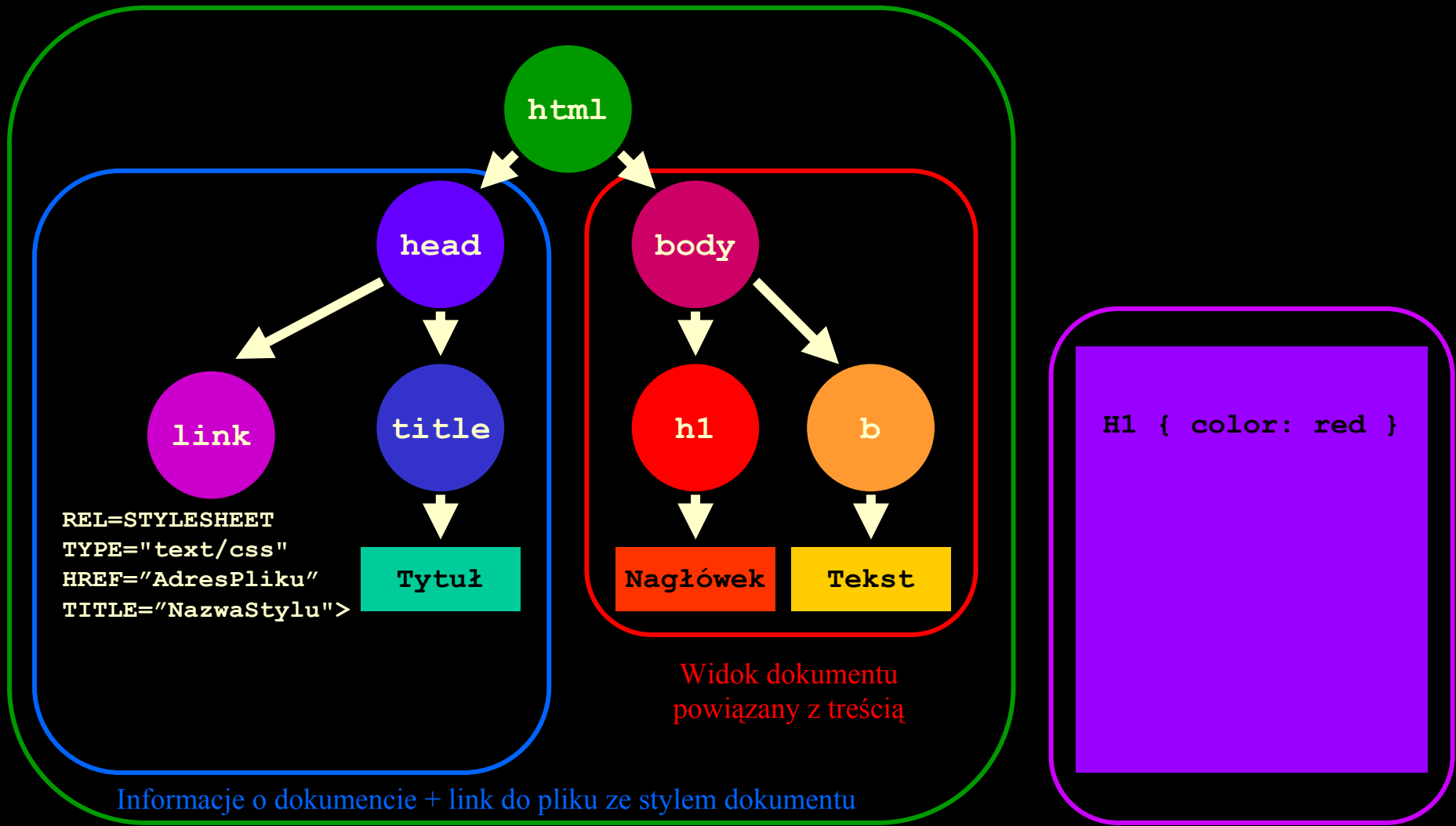
CSS - znacznik <style>



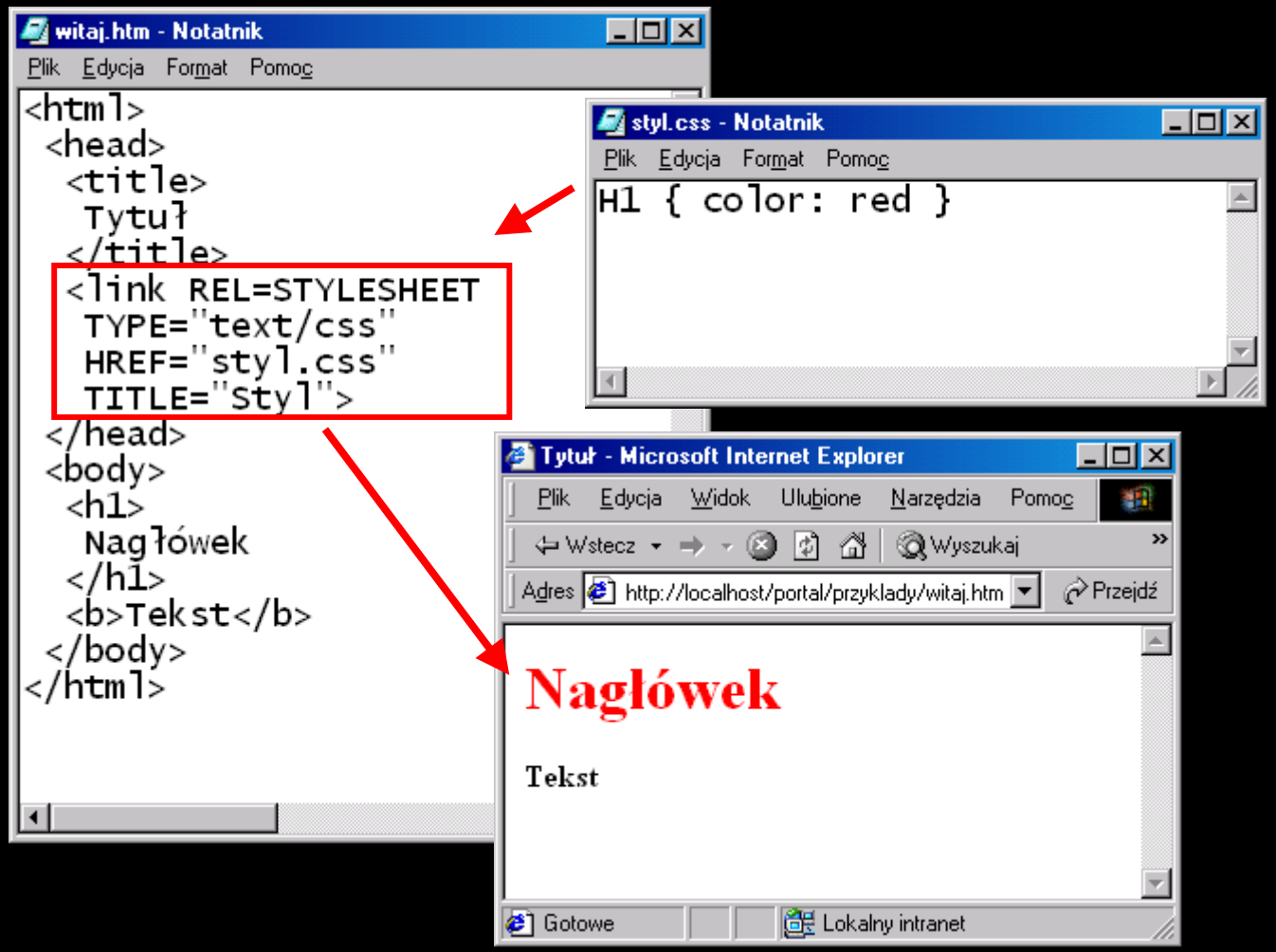
CSS - znacznik <style>



CSS - znacznik <link>

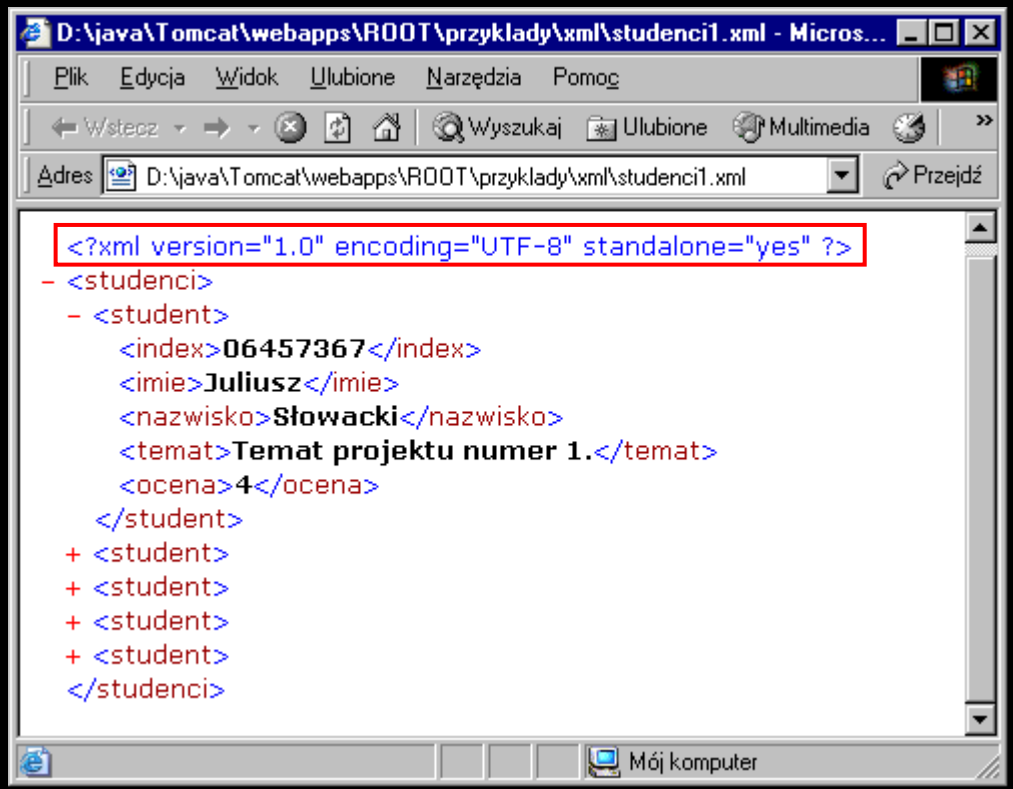


CSS - znacznik <link>



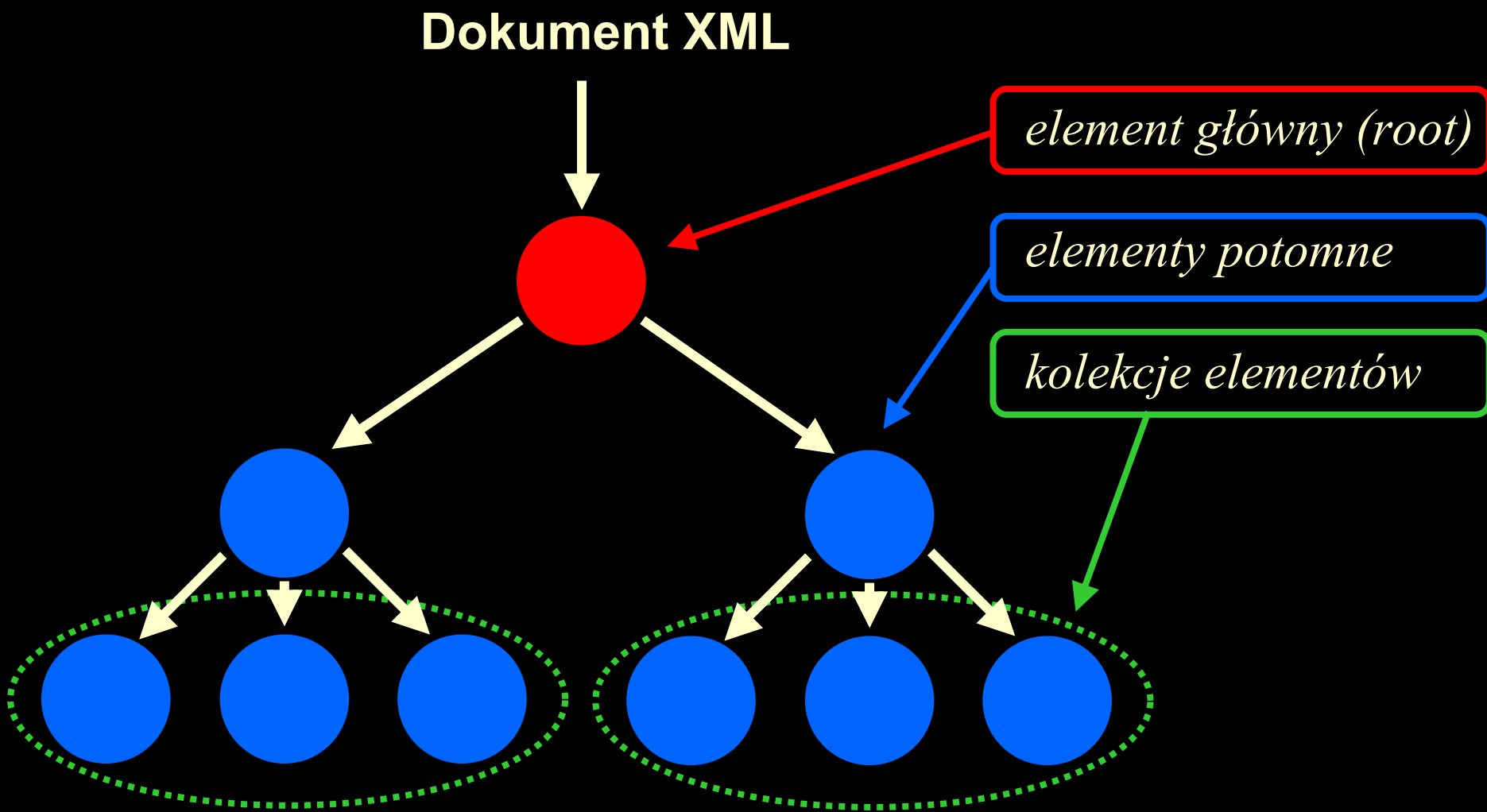
XML

Przeglądarki nie potrafią rozpoznać typów danych dokumentu XML i nie są w stanie zaproponować sposobu prezentacji tych danych.



Widok przykładowego dokumentu XML

Struktura dokumentu XML



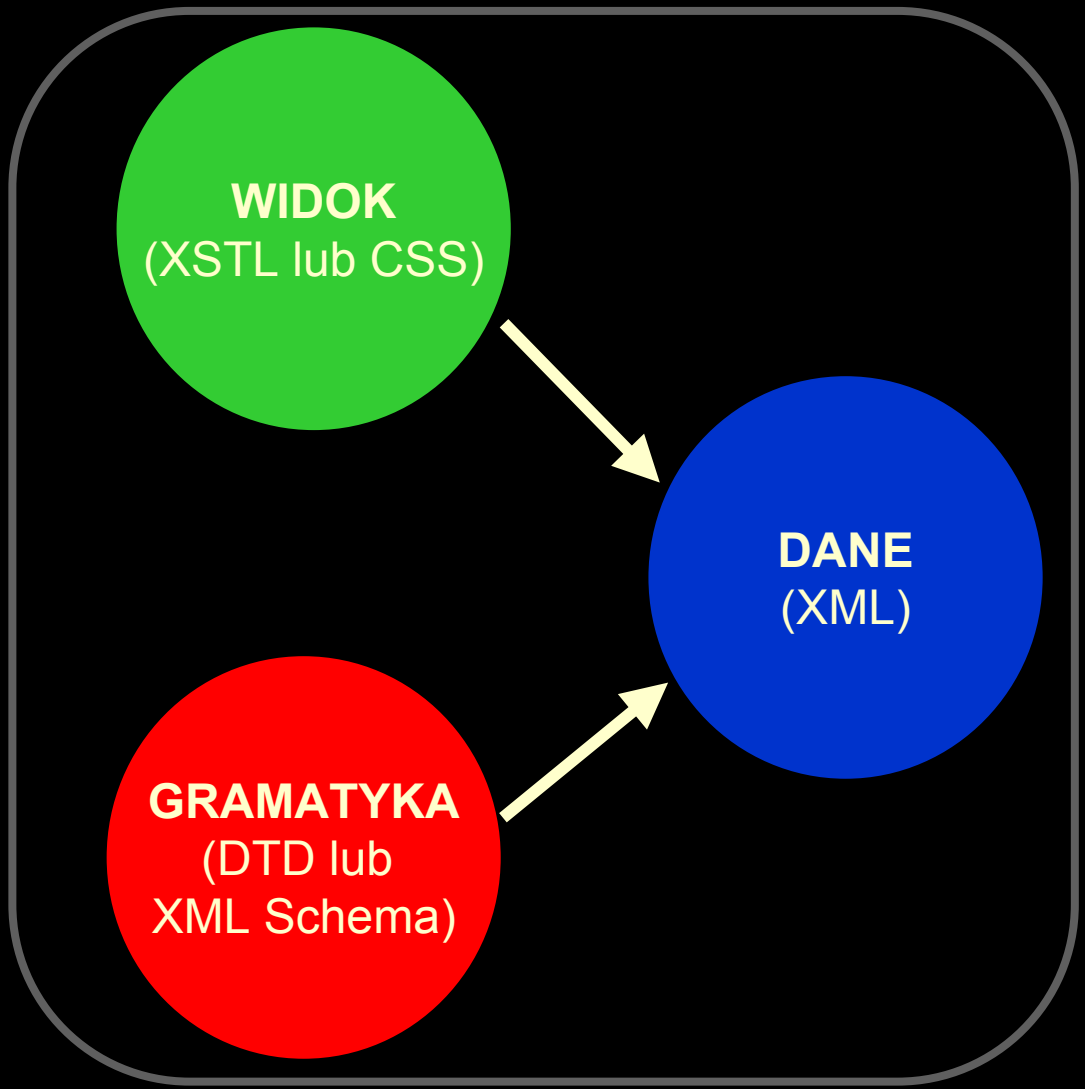
XML

Dokument XML przechowuje tylko dane

Do wykorzystania pełnych możliwości dokumentów XML potrzebne są dwa dodatkowe dokumenty:

- do wizualizacji (CSS lub XSTL)
- do zdefiniowania formatów i poprawności danych (DTD lub XML Schema)

XML



Publikowanie dokumentu XML z odpowiednim widokiem i o poprawnej strukturze

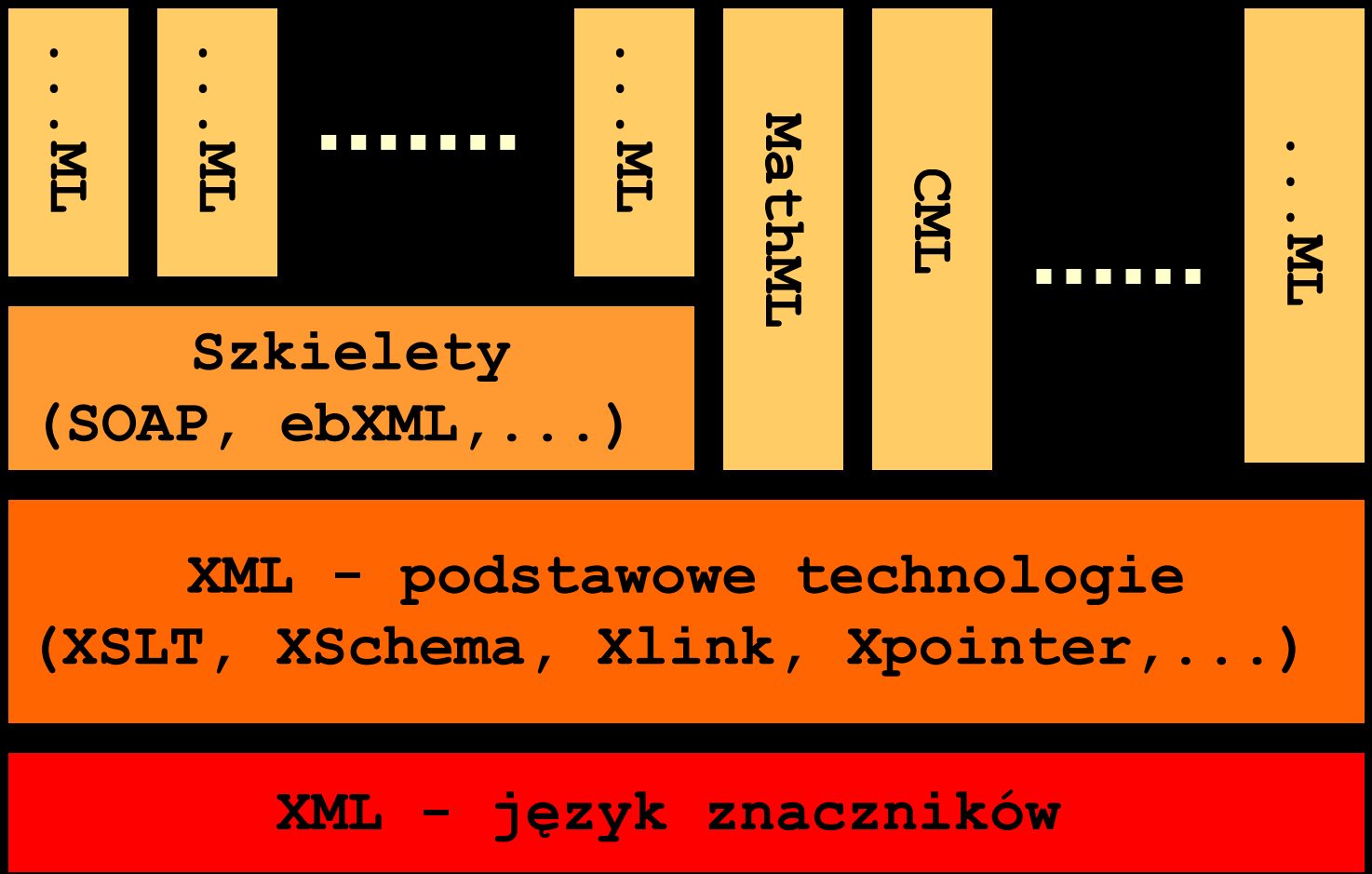
Cechy dokumentów XML

- opis danych zawartych w dokumencie XML poprzez własne nazewnictwo znaczników
- zdolność adaptacji do konkretnych potrzeb i zastosowań (z tego samego dokumentu XML można otrzymać stronę HTML, WML, czy też dokument PDF)
- łatwość zarządzania dokumentem poprzez oddzielenie struktury danych od prezentacji

Cechy dokumentów XML

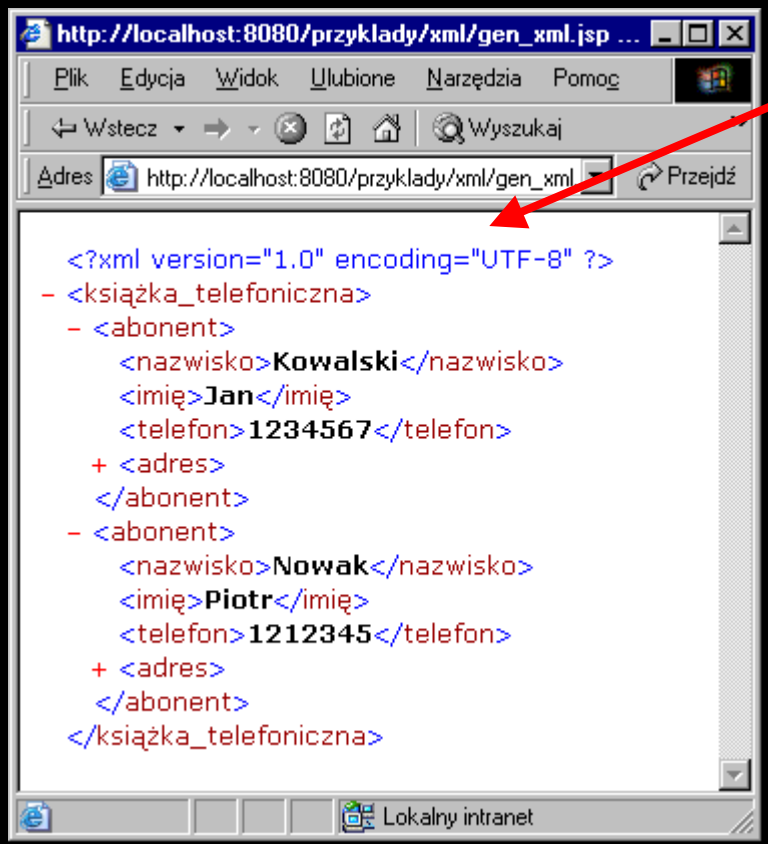
- możliwość bezpośredniej publikacji w internecie
- sprawdzenie poprawności danych zawartych w dokumencie XML (walidacja)
- łatwość realizacji systemów wyszukiwania informacji w oparciu o dane w XML'u
- tworzenie systemów baz danych zawierających drzewiaste struktury XML

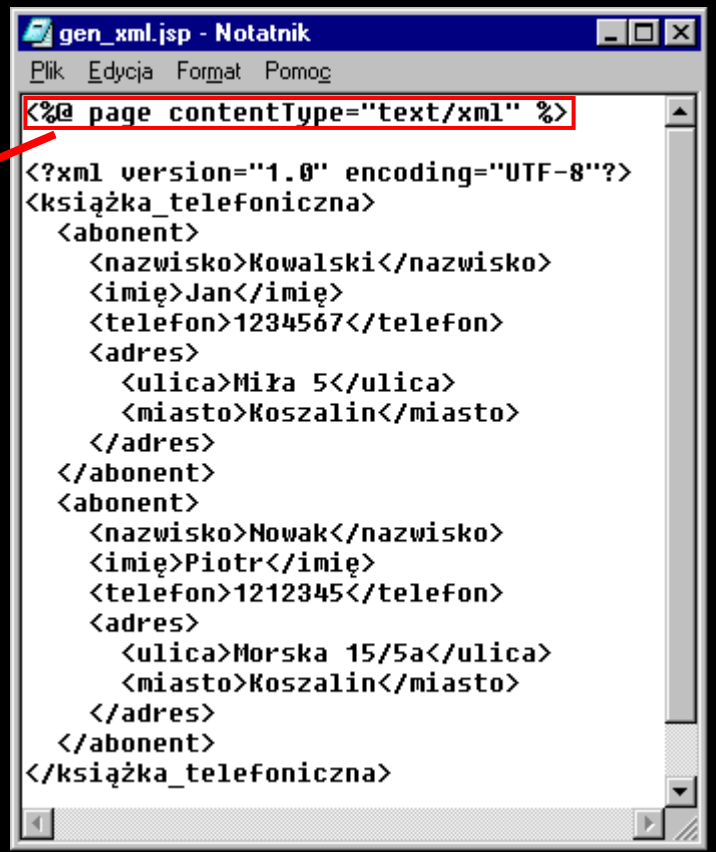
Meta-języki


...ML
...ML
.....
...ML
MathML
CML
.....
...ML
Szkielety
(SOAP, ebXML,...)
XML - podstawowe technologie
(XSLT, XSchema, Xlink, Xpointer,...)
XML - język znaczników

Generacja XML'a z JSP

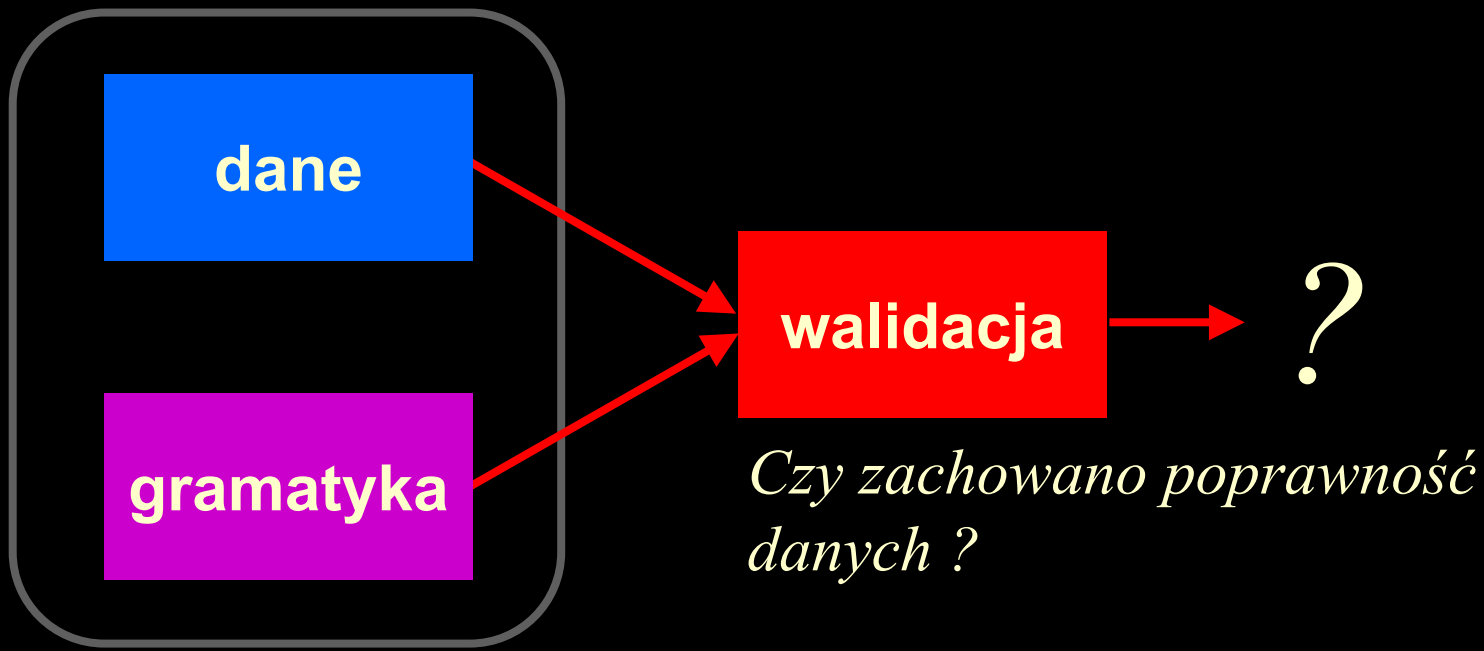
Generacja dokumentu XML w skrypcie JSP





Weryfikacja

Weryfikacja zgodności z założoną strukturą danych



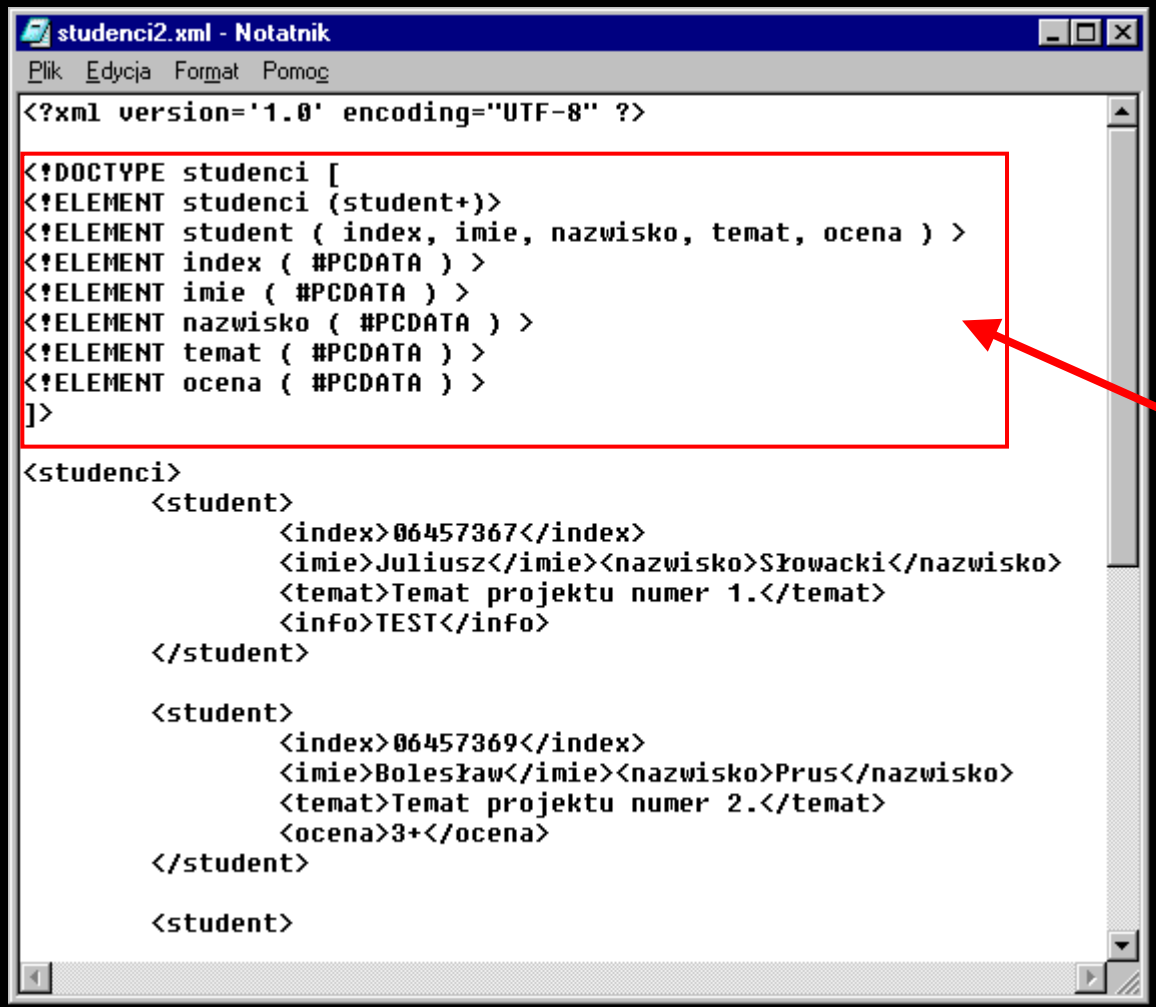
Definicja Typu Dokumentu

DTD (Document Type Definition) określa składnię (gramatykę) dokumentu XML:

- hierarchię elementów (znaczników)
- dopuszczalne atrybuty oraz ich wartości

DTD

Dokument XML + DTD



Gramatyka dokumentu (wewnątrz dokumentu XML)

DTD, a XML

Ograniczenia opisu XML za pomocą DTD

- brak możliwości parsowania (brak zgodności ze składnią XML, DTD nie jest XML'em)
- brak wsparcia dla przestrzeni nazw

XML Schema

XML Schema jest alternatywnym rozwiązaniem w stosunku do języka definiowania typów dokumentów (DTD)

- składnia zgodna z XML (możliwość walidacji dokumentów)
- określanie typu zawartości elementów, maksymalnej i minimalnej liczby powtórzeń elementów
- definiowanie typów złożonych

XML Schema

Lister - [D:\java\Tomcat\webapps\ROOT\przyklady\xml\studenci_schema.x...

File Edit Options Help 100 %

```
<?xml version='1.0' encoding="ISO-8859-2" standalone="yes" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">

<ElementType name="imie" content="textOnly"/>
<ElementType name="nazwisko" content="textOnly"/>

</Schema>
```

Dokument XML Schema (gramatyka)

Przykładowa defincja XML Schema

Dokument XML (dane)

studenci4.xml - Notatnik

Plik Edycja Format Pomoc

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<?xml:stylesheet type="text/xsl" href="studenci.xsl"?>

<studenci xmlns="x-schema:studenci_schema.xml">
        <student>
                <index>06457367</index>
                <imie>Juliusz</imie><nazwisko>Słowacki</nazwisko>
                <temat>Temat projektu numer 1.</temat>
                <ocena>4</ocena>
        </student>

        <student>
                <index>06457369</index>
                <imie>Bolesław</imie><nazwisko>Prus</nazwisko>
                <temat>Temat projektu numer 2.</temat>
                <ocena>3+</ocena>
```

Definiowanie schematów

- DCD
- DDML
- RDF
- RELAX
- Schematron
- SOX
- Trex
- XDR
- Xduce
- XML-Data
- Xscheam

XML Schema

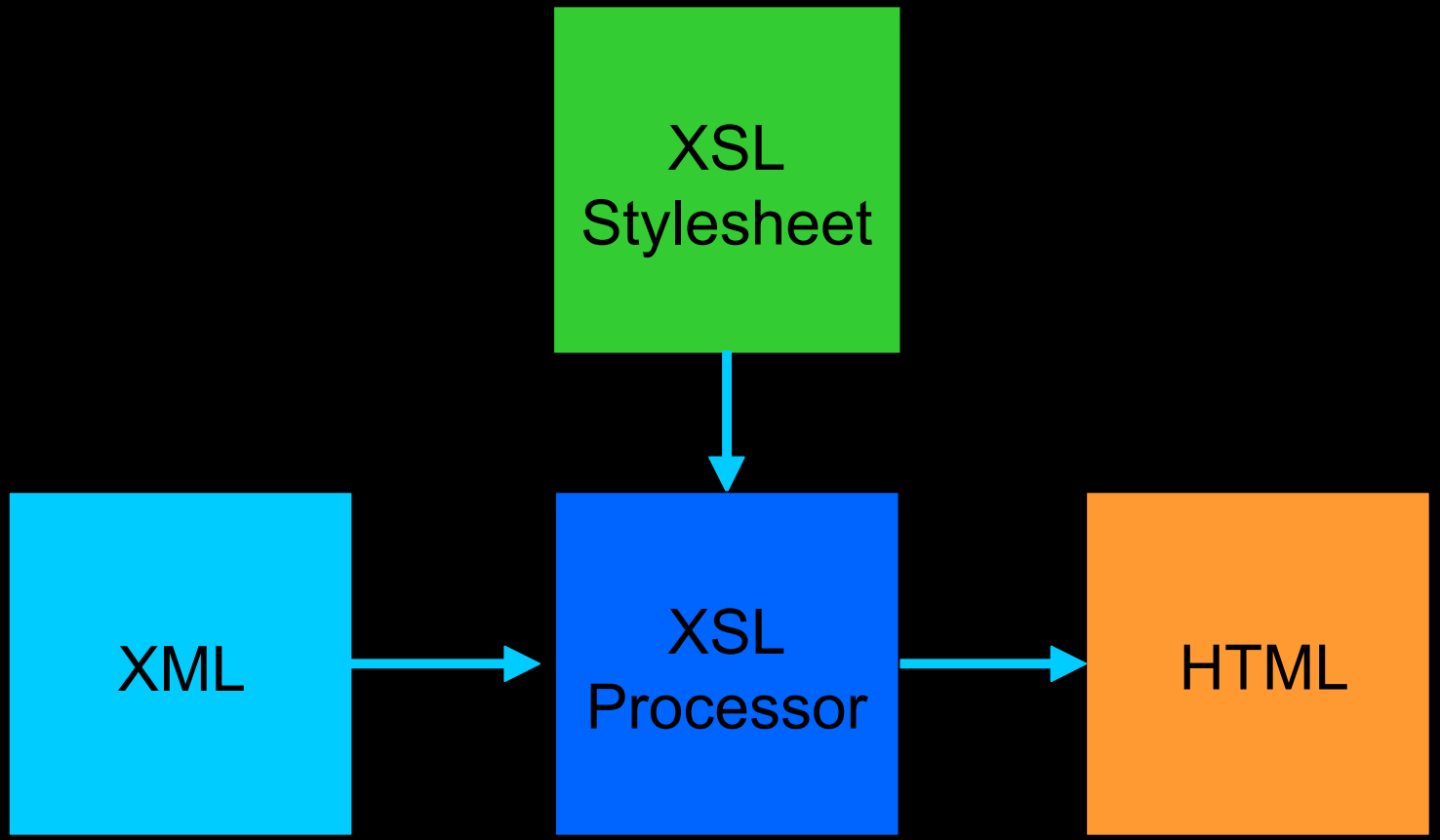
Formatowanie dokumentów XML

- po stronie serwera
  - kontrola nad sposobem wyświetlenia dokumentu
- po stronie klienta
  - przeglądarka z obsługą formatowania
  - dodatkowy narzut czasowy
  - brak kontroli nad prezentacją dokumentu (różnice w interpretacji danych przez przeglądarki)

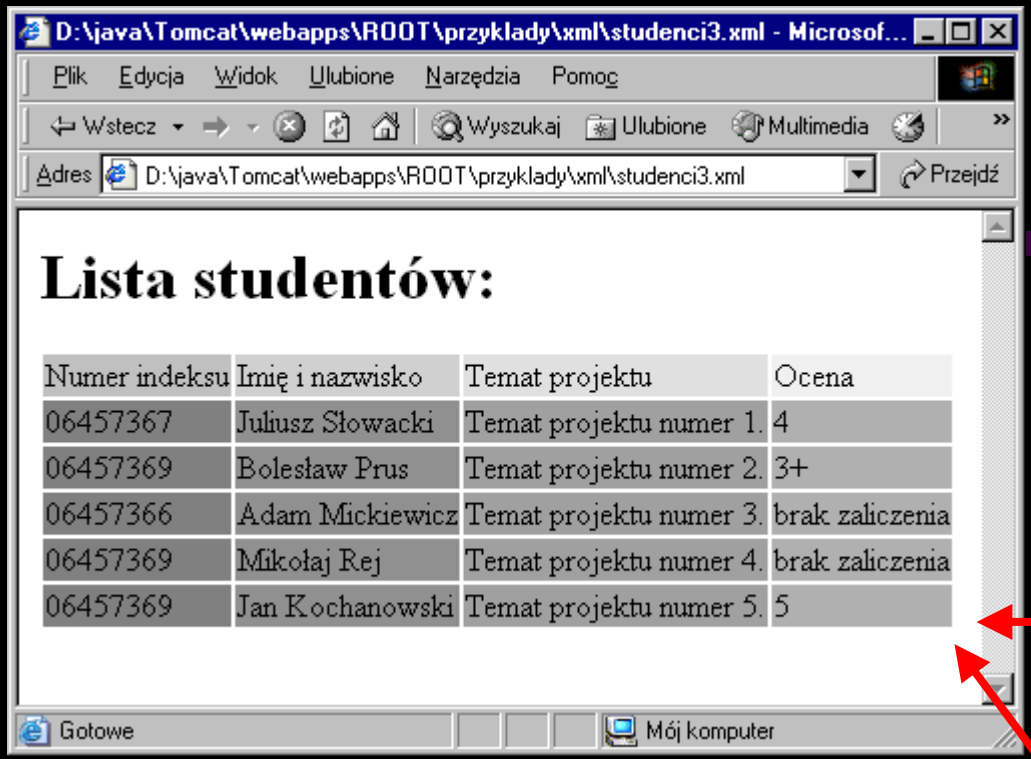
Przekształcenia XSLT

XSLT - język przekształceń dokumentów XML na inne formaty np. HTML (umożliwienie wyświetlania dokumentów XML w przeglądarkach internetowych)

Architektura XSL

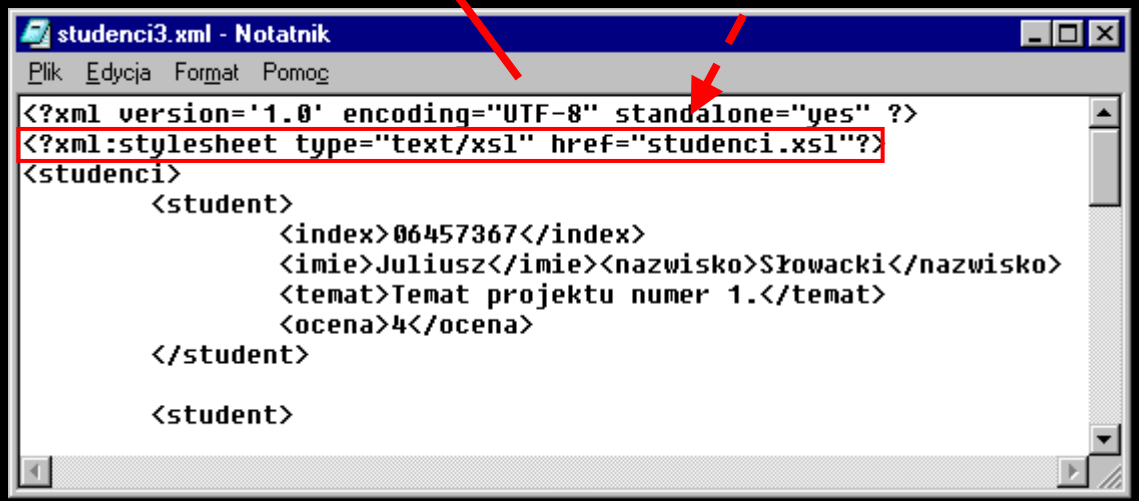


XSL



Przeglądarka (widok)

Dokument XSL (wygląd)



Dokument XML (dane)

Za proces tworzenia widoku odpowiada przeglądarka (klient)

Dokument XSL 1/4

```
<?xml version='1.0' encoding="UTF-8" standalone="yes" ?>
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/TR/WD-xsl"
        xmlns="http://www.w3.org/TR/REC-html40"
        result-ns="">
```

```
<xsl:template>
<xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="/">
<HTML>
        <HEAD>
        <TITLE></TITLE>
        </HEAD>
        <BODY>
                <H1>Lista studentów:</H1>
                <xsl:apply-templates select="/studenci"/>
        </BODY>
</HTML>
</xsl:template>
```

Dokument XSL 2/4

```
<xsl:template match="studenci">
        <TABLE BORDER="0">
                <TR>
                        <TD BGCOLOR="#C0C0C0">Numer indeksu</TD>
                        <TD BGCOLOR="#D0D0D0">Imię i nazwisko</TD>
                        <TD BGCOLOR="#E0E0E0">Temat projektu</TD>
                        <TD BGCOLOR="#F0F0F0">Ocena</TD>
                </TR>

                <xsl:apply-templates select="/studenci/student"/>

        </TABLE>
        <BR/>
</xsl:template>
```

Dokument XSL 3/4

```
<xsl:template match="student">
        <TR>
                <TD BGCOLOR="#808080">
                        <xsl:apply-templates select="index"/>
                </TD>
                <TD BGCOLOR="#909090">
                        <xsl:apply-templates select="imie"/>
                        <xsl:apply-templates select="nazwisko"/>
                </TD>
                <TD BGCOLOR="#A0A0A0">
                        <xsl:apply-templates select="temat"/>
                </TD>
                <TD BGCOLOR="#B0B0B0">
                        <xsl:apply-templates select="ocena"/>
                </TD>
        </TR>
</xsl:template>
```

Dokument XSL 4/4

```
<xsl:template match="index">
        <xsl:value-of/>
</xsl:template>
```

```
<xsl:template match="imie">
        <xsl:value-of/>
</xsl:template>
```

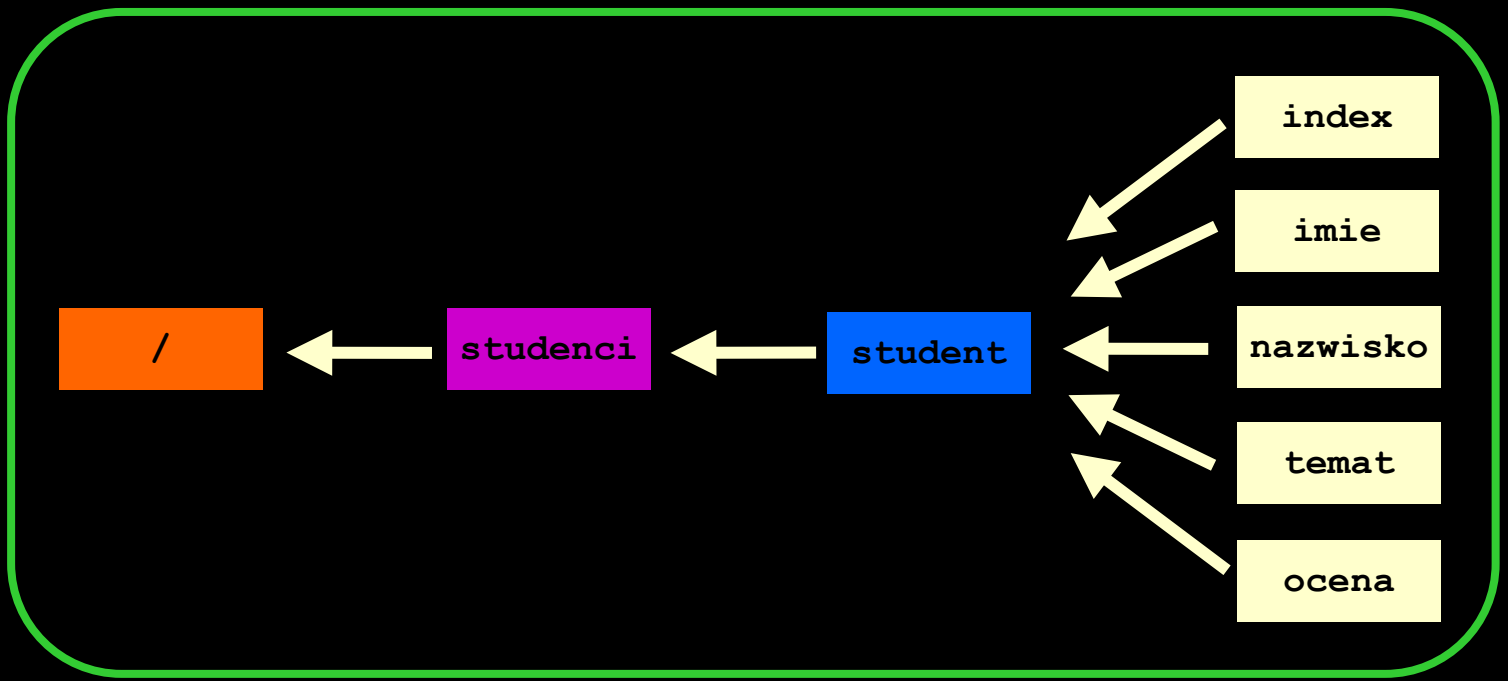
```
<xsl:template match="nazwisko">
        <xsl:value-of/>
</xsl:template>
```

```
<xsl:template match="temat">
        <xsl:value-of/>
</xsl:template>
```

```
<xsl:template match="ocena">
        <xsl:value-of/>
</xsl:template>
```

```
</xsl:stylesheet>
```

Dokument XSL




```
xmlns:xsl="http://www.w3.org/TR/WD-xsl"
xmlns="http://www.w3.org/TR/REC-html40"
```

Przetwarzanie XML

- publikacja tych samych danych w różnych postaciach (wml,pdf)
- publikowanie danych dla różnych klientów (personalizacja stron)
- dodawanie, usuwanie oraz modyfikacja dokumentów XML
- ładowanie danych do baz danych
- tworzenie raportów

Publikowanie HTML

HTML

- używanie znaczników obsługiwanych przez wszystkie przeglądarki
- tworzenie stron HTML dostosowanych do konkretnej przeglądarki
- dołączanie do stron skryptów, generujących kod specyficzny dla danej przeglądarki

Publikowanie XML

XML

- sprawdzenie przeglądarki klienta
- przekształcenie (w locie) dokumentu XML do odpowiedniej postaci akceptowanej przez przeglądarkę klienta

Obsługiwany jest tylko jeden plik źródłowy:

- formatowanie i przetwarzanie jest oddzielone od danych

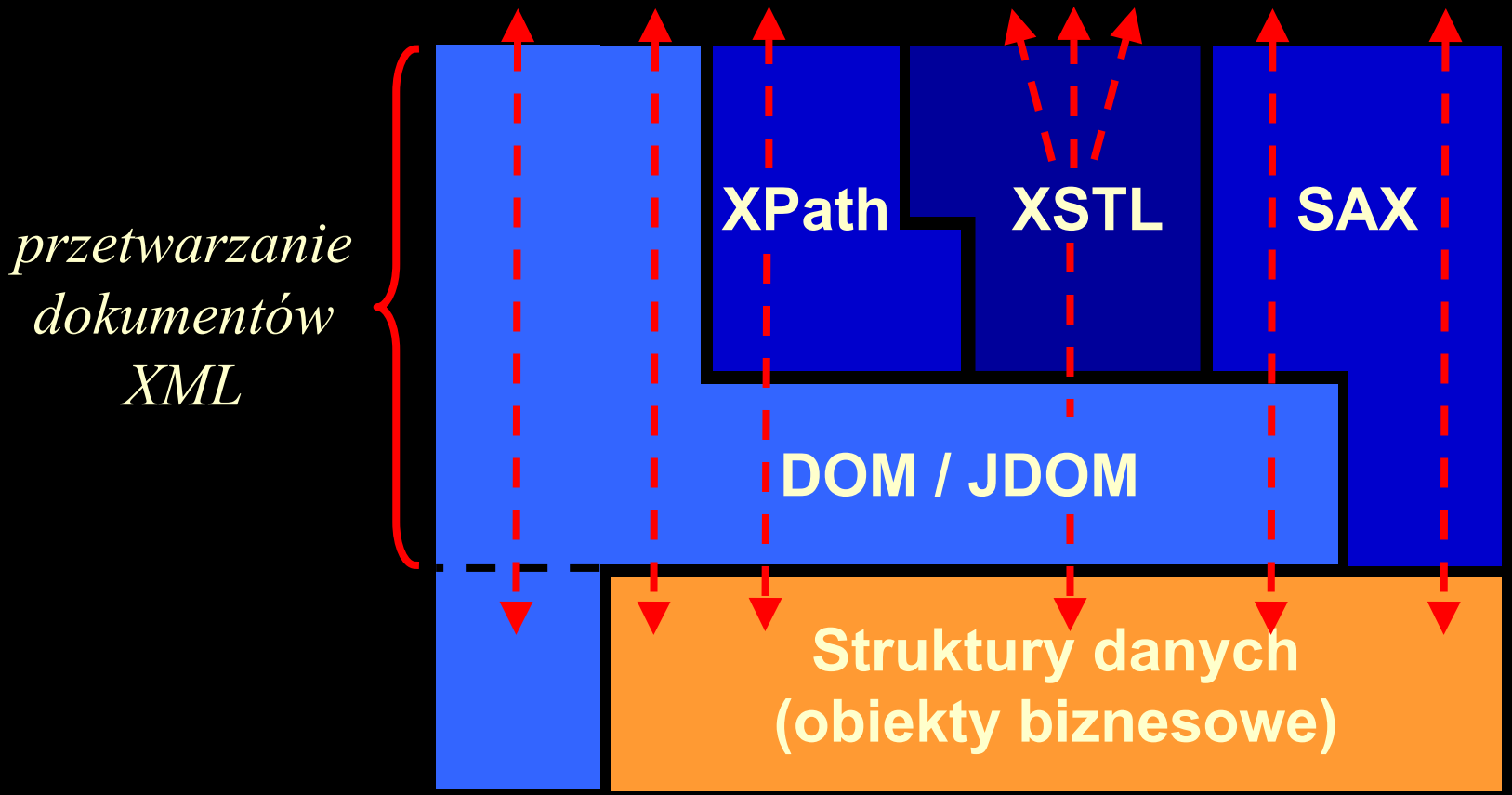
Personalizacja XML

Informacje z XML można publikować z adresacją dla różnych grup odbiorców np.

- początkujący
- zaawansowani
- specjaliści

Możliwość publikowania danych w zależności od zaawansowania użytkowników

SAX & DOM & Xpath & XSTL



SAX & DOM

SAX (Simple API for XML)

- model oparty o zdarzenia
- sekwencyjny dostęp do elementów XML
- niewielkie zużycia pamięci (tylko dla wygenerowanych zdarzeń)
- jednokrotny proces analizy dokumentu

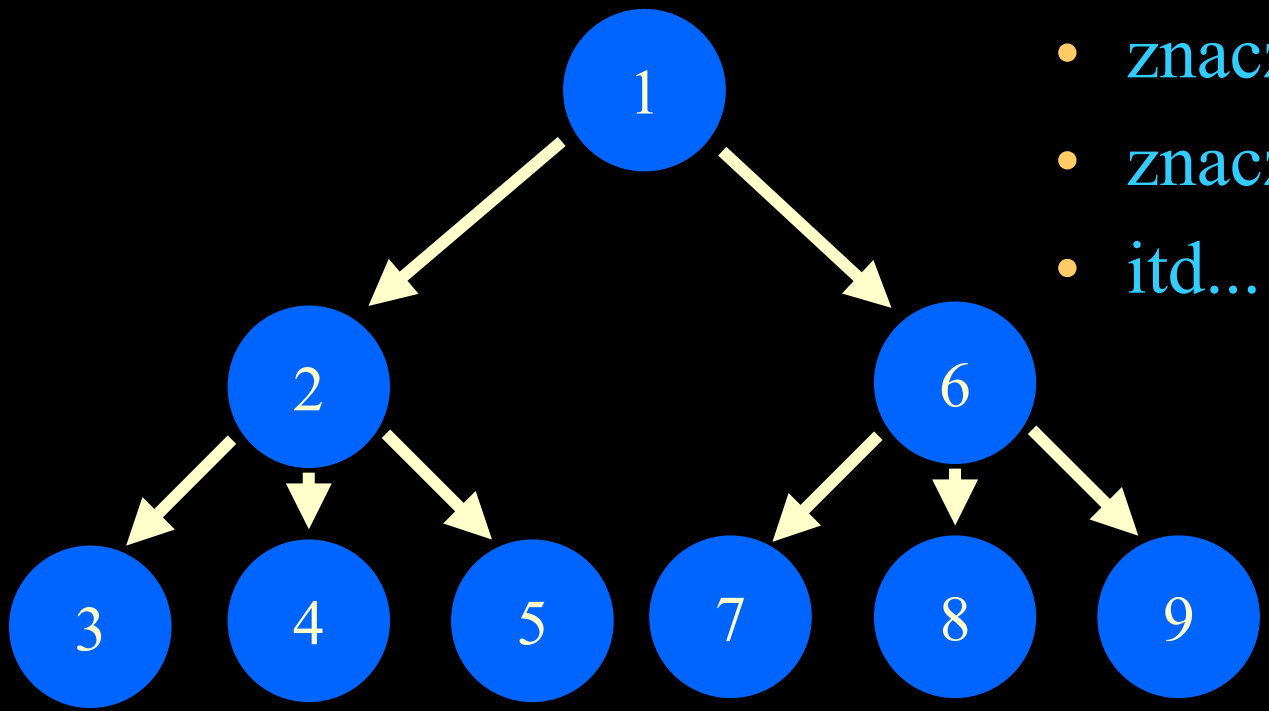
DOM (Document Object Model)

- model oparty o struktury drzewiaste
- swobodny dostęp do elementów XML
- duże zapotrzebowanie na pamięć
- możliwość wielokrotnej analizy dokumentu (całość znajduje się w pamięci)

SAX - zdarzenia

Podczas sekwencyjnego czytania pliku XML mają miejsce zdarzenia np.

- początek dokumentu
- znacznik początkowy
- znacznik końcowy
- itd...



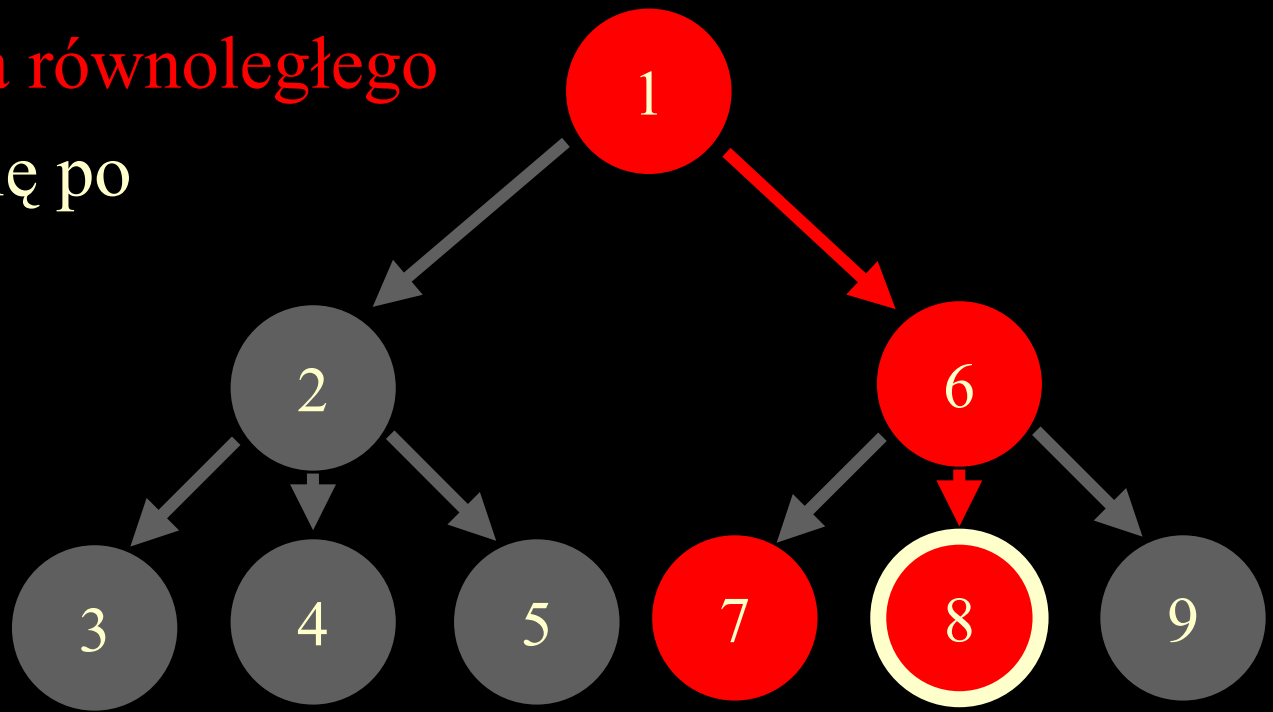
Numerkami oznaczono kolejność przetwarzania węzłów

Procedura Obsługi Zdarzenia - fragment kodu uruchamiany, kiedy dane zdarzenie ma miejsce

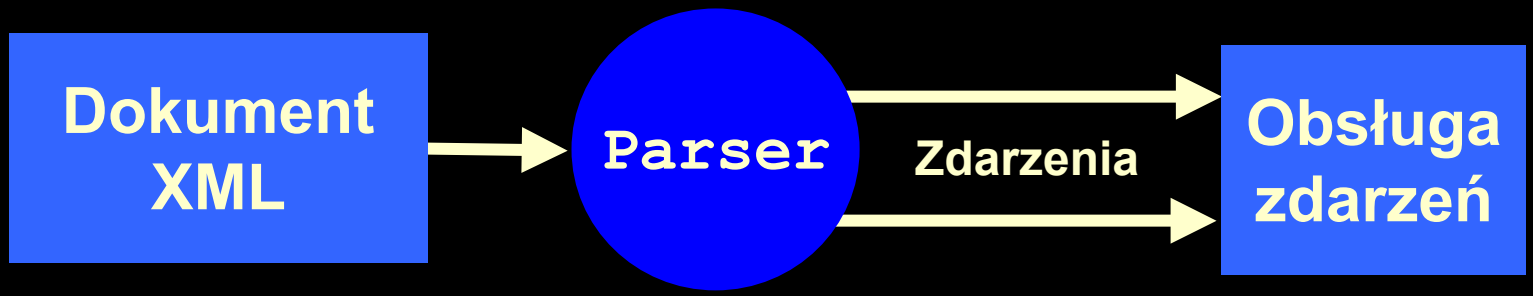
SAX - zdarzenia

Analizator pamięta rodzaje elementów i atrybuty:

- węzłów nadrzędnych (przodków), aż do elementu głównego
- elementu węzła równoległego znajdującego się po lewej stronie



SAX - zdarzenia



Zalety

- prostota, szybkość działania
- małe obciążenie pamięci

Wady

- brak możliwości „spojrzenia w przód” (wymagana jest wtedy powtórna analiza)

SAX - przykład

```
<%@ page import="java.io.*" %>
<%@ page import="org.xml.sax.*" %>
<%@ page import="org.xml.sax.helpers.*" %>

<%!
  class SAXHandler extends DefaultHandler {
  public void startDocument()    { ... }
  public void endDocument()      { ... }
  public void startElement (...) { ... }
  public void endElement (...)   { ... }
  public void characters (...)   { ... }
  public void warning (SAXParseException e)    { ... }
  public void error (SAXParseException e)      { ... }
  public void fatalError (SAXParseException e) { ... }
}
%>

<%
  String uri = application.getRealPath("/przyklady/xml/users.xml");
  XMLReader parser = new org.apache.xerces.parsers.SAXParser();
  SAXHandler handler = new SAXHandler();
  parser.setContentHandler(handler);
  parser.setErrorHandler(handler);
  parser.parse(uri);
%>
```

SAX - przykład

Metody obsługi zdarzeń klasy SAXHandler

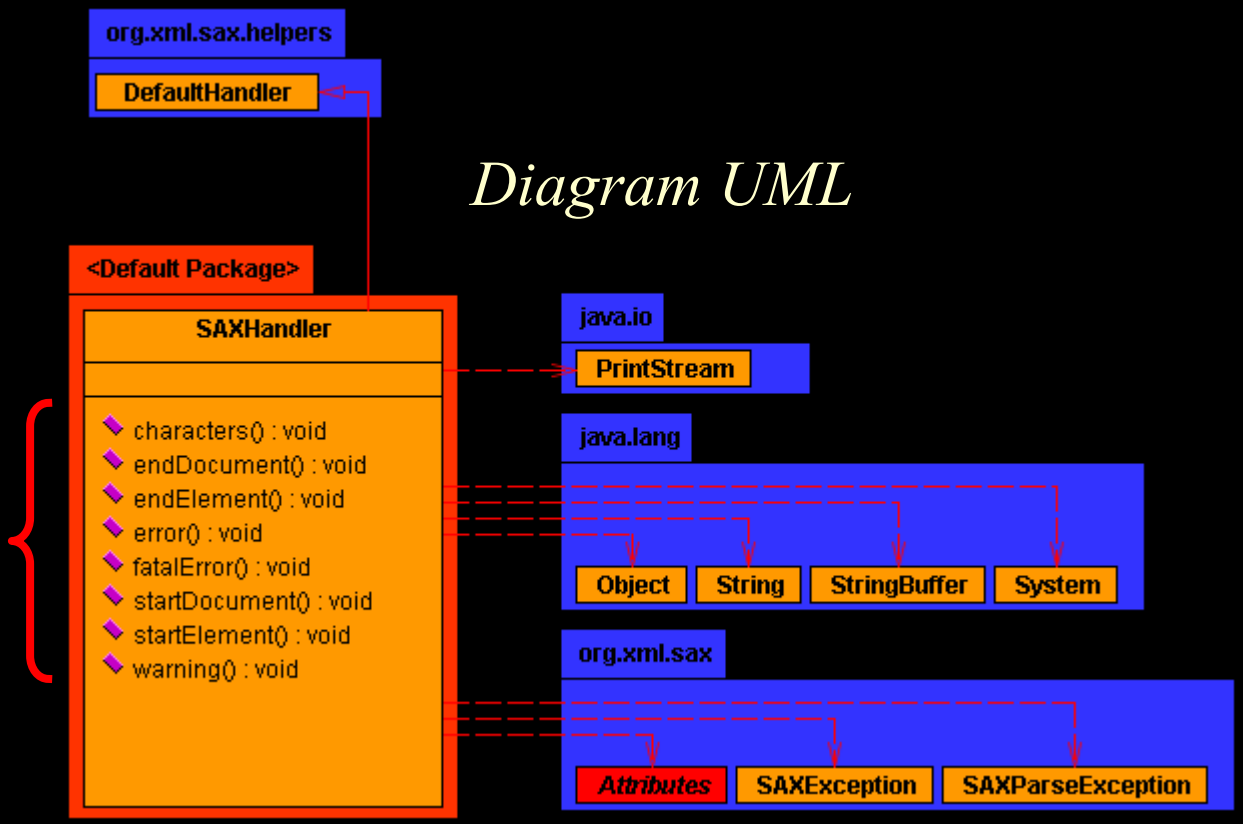
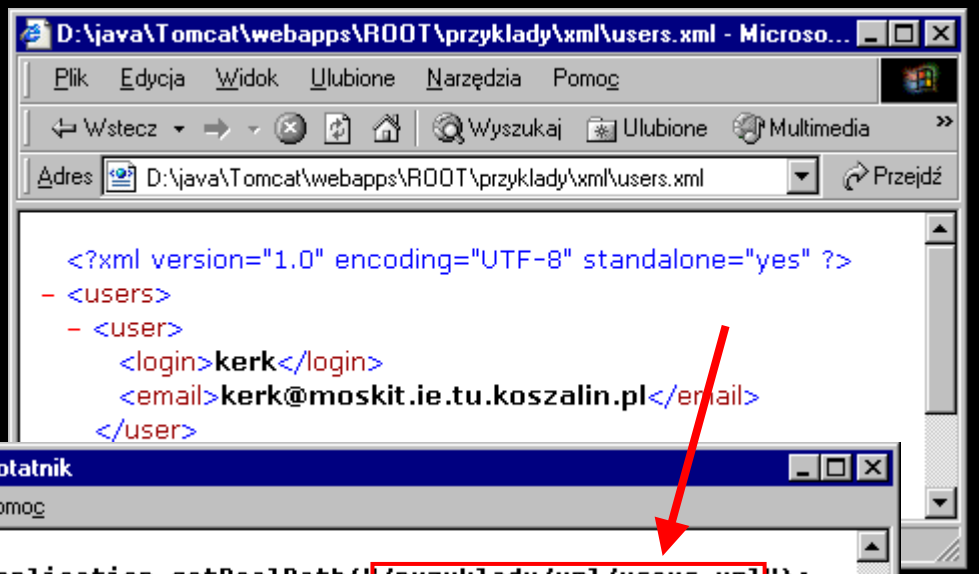


Diagram UML

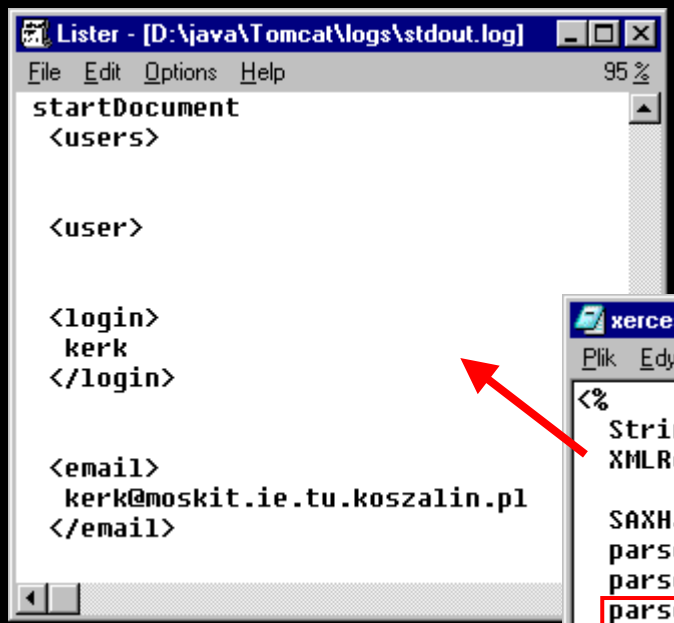
Biblioteka org.xml.sax

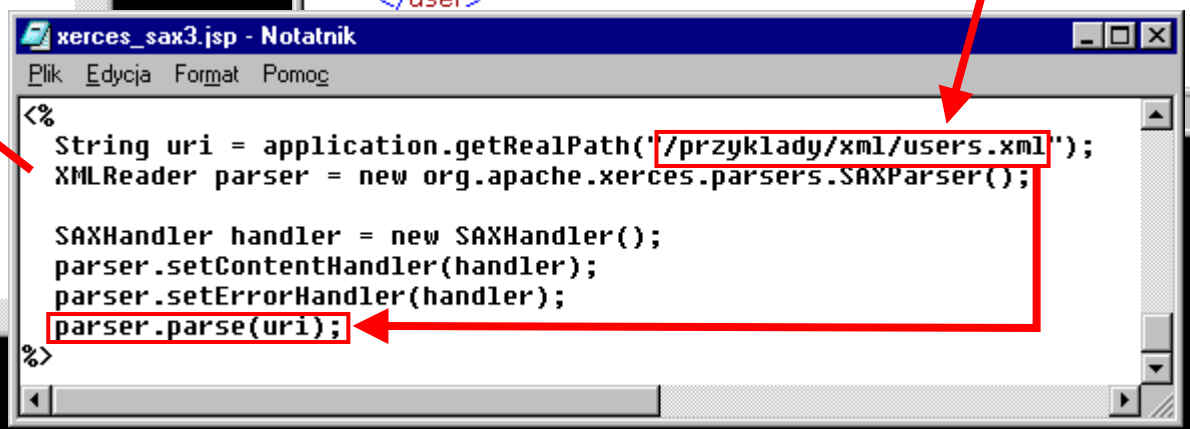
SAX - przykład

Dokument XML



Stdout.log (TOMCAT)





Strona JSP (serwer)

SAX - przykład 1/3

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

<%@ page import="org.xml.sax.*" %>
<%@ page import="org.xml.sax.helpers.*" %>

<%!
  class SAXHandler extends DefaultHandler {

  public void startDocument() {
    System.out.println("startDocument");
  }

  public void endDocument() {
    System.out.println("endDocument");
  }

  public void startElement (String uri, String localName, String qName,
                                                    Attributes atts) {
    System.out.println(" <"+localName+">");
  }

  public void endElement (String uri, String localName, String qName) {
    System.out.println(" </"+localName+">");
  }
```

SAX - przykład 2/3

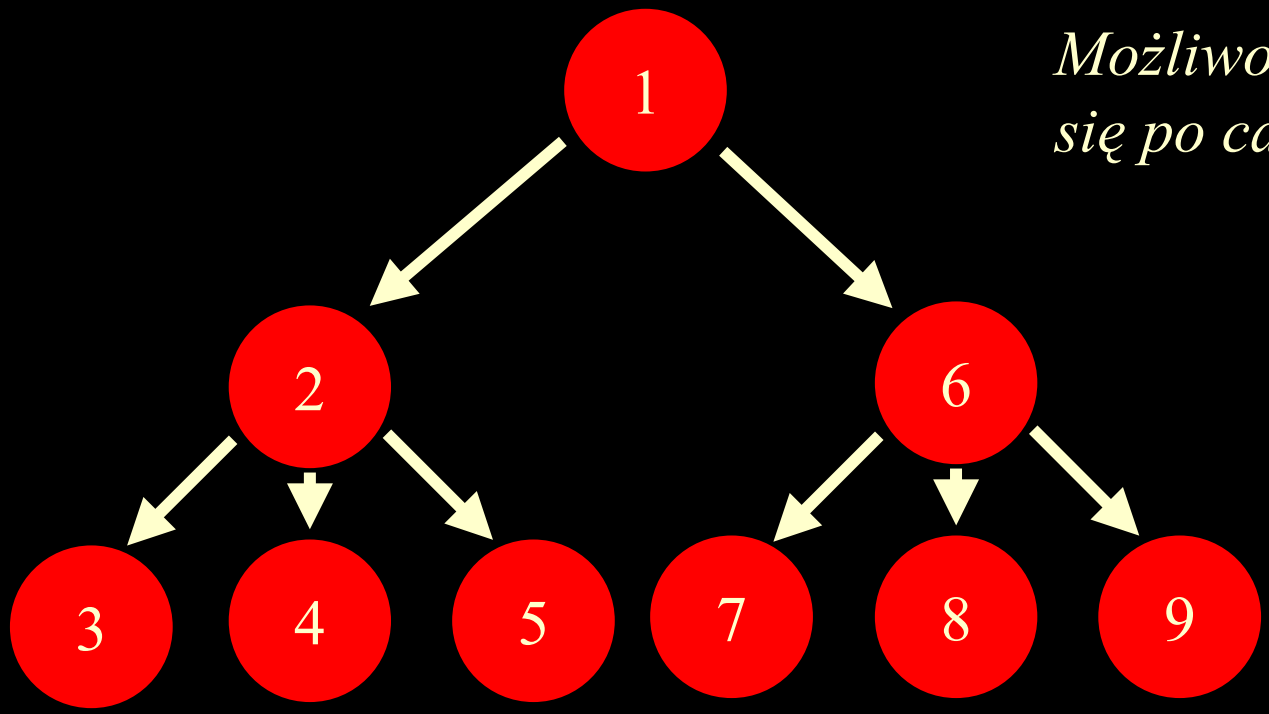
```
  public void characters (char ch[], int start, int length) {
    System.out.println("  "+ new String(ch, start, length));
  }

  public void warning (SAXParseException e) {
    System.err.println("warning ("+e.getSystemId()+":"+
        e.getLineNumber()+":"+e.getColumnNumber()+") "+e.getMessage());
  }

  public void error (SAXParseException e) {
    System.err.println("error ("+e.getSystemId()+":"+
        e.getLineNumber()+":"+e.getColumnNumber()+") "+e.getMessage());
  }

  public void fatalError (SAXParseException e) {
    System.err.println("fatalError ("+e.getSystemId()+":"+
        e.getLineNumber()+":"+e.getColumnNumber()+") "+e.getMessage());
  }
}
%>
```

SAX - przykład 3/3

```
<%
  String uri = application.getRealPath("/przyklady/xml/users.xml");
  XMLReader parser = new org.apache.xerces.parsers.SAXParser();

  SAXHandler handler = new SAXHandler();
  parser.setContentHandler(handler);
  parser.setErrorHandler(handler);
  parser.parse(uri);

%>
```

DOM - struktura hierarchiczna/drzewiasta

Dokument XML zostaje przekształcony na wewnętrzną reprezentację drzewiastą



Możliwość poruszania się po całym drzewie

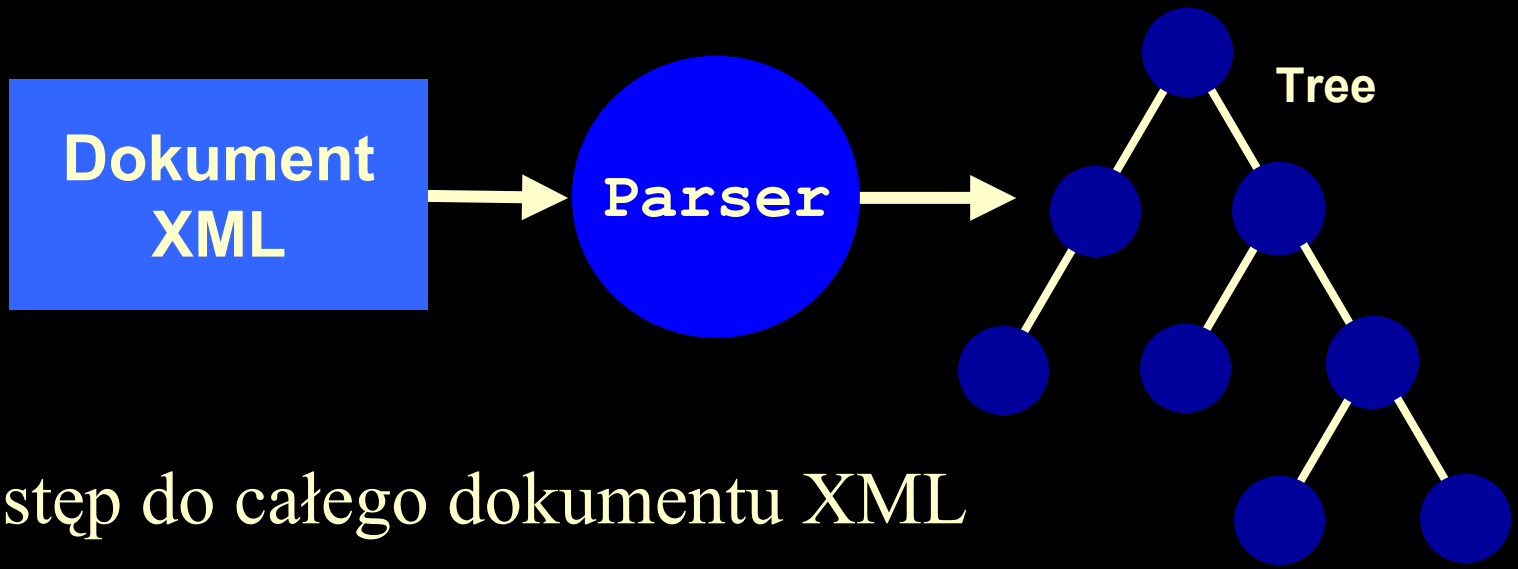
DOM - struktura hierarchiczna/drzewiasta

Analiza dokumentu XML przeprowadzona zostaje w dwóch przebiegach:

- przebieg pierwszy: analiza dokumentu i budowa struktury drzewiastej
- przebieg drugi: przetwarzanie samych danych

Możliwość „spojrzenia w przód” (np. czy element jest ostatnim dzieckiem swego rodzica)

DOM - struktura hierarchiczna/drzewiasta



Zalety

– dostęp do całego dokumentu XML

Wady

– budowa całego drzewa i poruszania się po nim wymaga większego nakładu pamięci
– wolniejsze przetwarzanie
– dwa przebiegi analizy dokumentu XML

DOM - przykład

```
<%@ page import="java.io.*" %>
<%@ page import="org.w3c.dom.*" %>
<%@ page import="org.apache.xerces.parsers.DOMParser" %>

<%!
  private void traverseTree(Node node, JspWriter out) throws Exception {
    ...
    switch (node.getNodeType()) {
      case Node.DOCUMENT_NODE: { ... }
      case Node.ELEMENT_NODE:  { ... }
      case Node.TEXT_NODE:     { ... }
    }
  }
%>

<%
  String uri = application.getRealPath("/przyklady/xml/users.xml");
  DOMParser parser = new DOMParser();
  parser.parse(uri);
  Document doc = parser.getDocument();
  ...
  traverseTree(doc,out);
  ...
%>
```

DOM - przykład

Document - specjalny rodzaj węzła reprezentujący całe drzewo dokumentu

Element - węzeł reprezentujący element dokumentu XML

Attr - obiekt reprezentujący atrybut elementu

Text - węzeł tekstowy, reprezentujący ciąg znaków nie zawierający znacznków ani innych konstrukcji składniowych XML

DOM

Funkcja traverseTree() w sposób rekurencyjny wykonuje analizę wszystkich węzłów drzewa

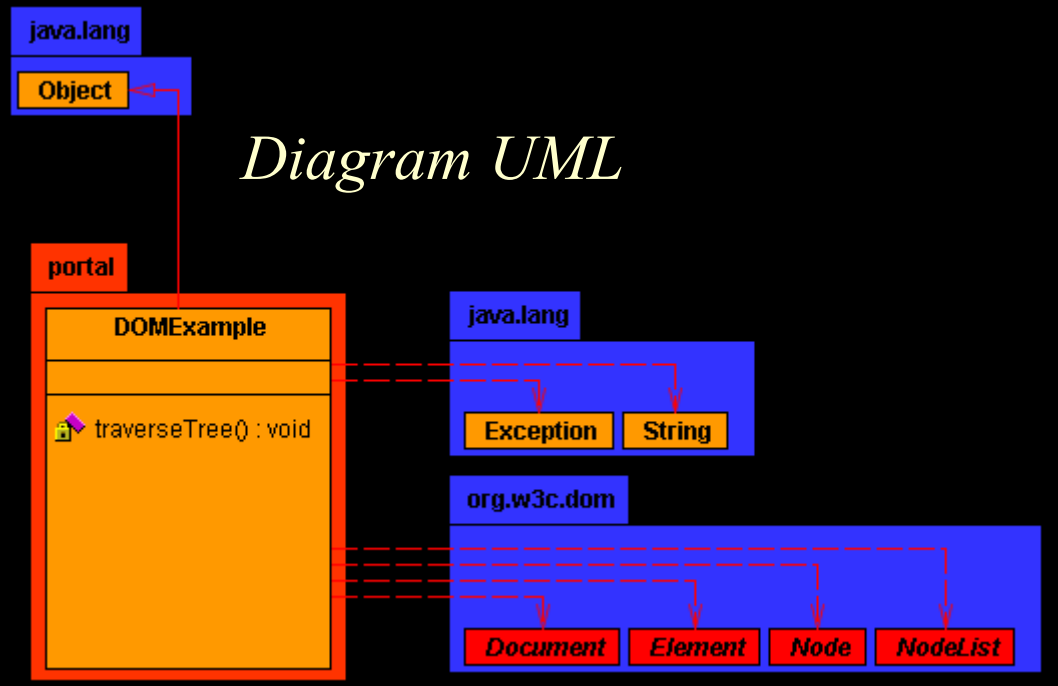
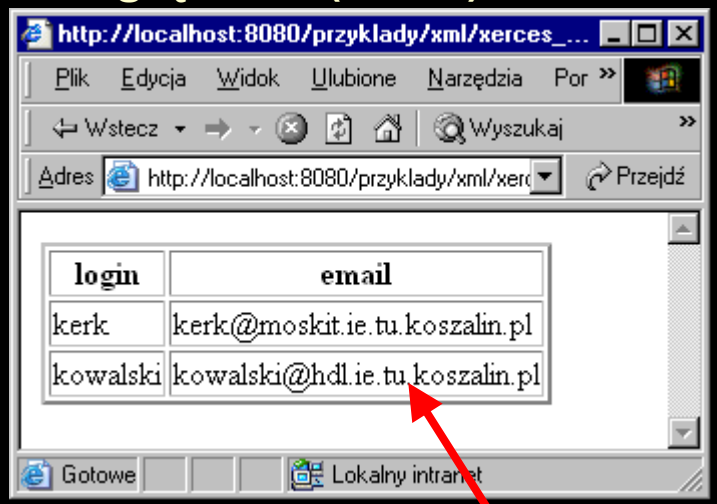


Diagram UML

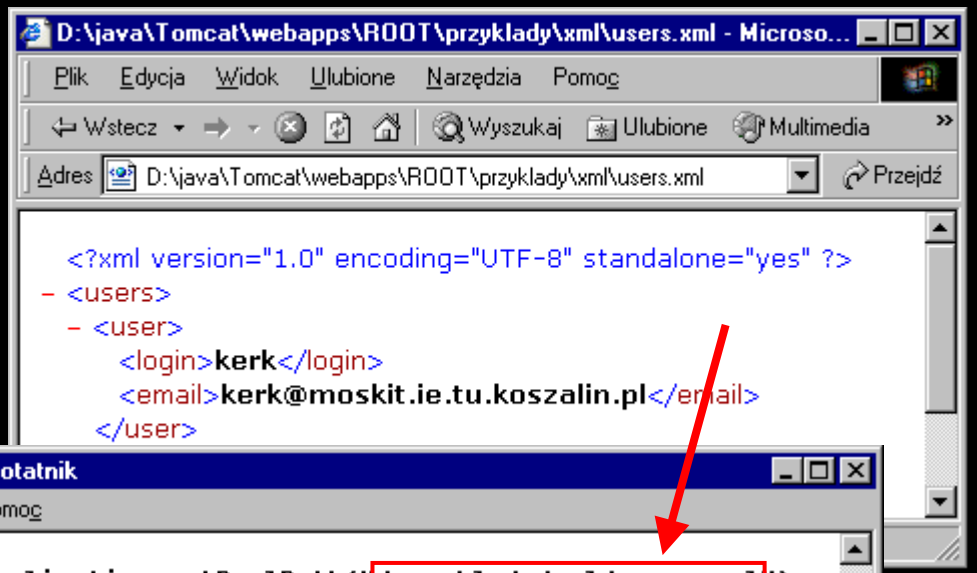
Biblioteka org.w3c.dom

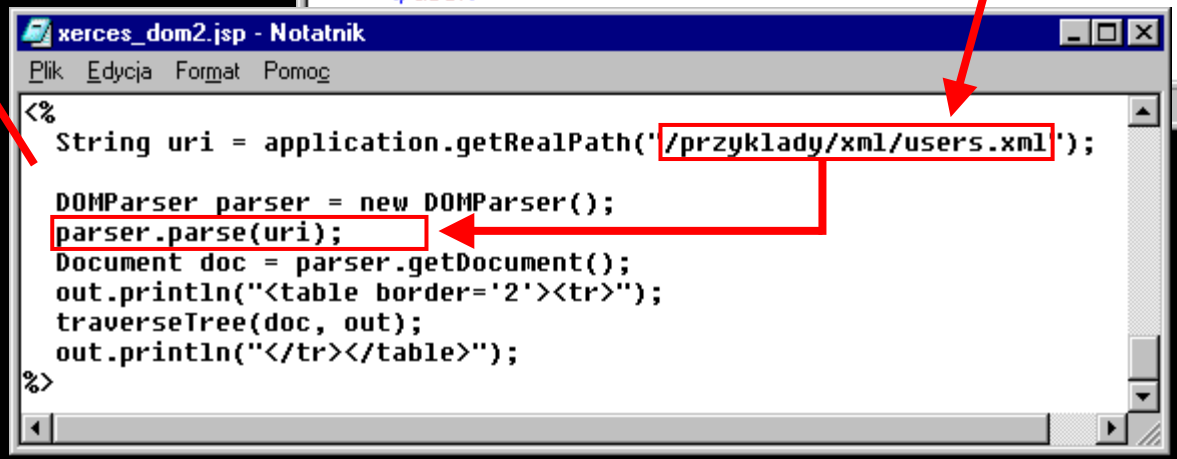
DOM - przykład

Przeglądarka (klient)



Dokument XML





Strona JSP (serwer)

DOM - przykład 1/3

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

<%@ page import="org.w3c.dom.*" %>
<%@ page import="org.apache.xerces.parsers.DOMParser" %>

<%!
  private void traverseTree(Node node, JspWriter out) throws Exception {
    if(node == null) { return; }

    switch (node.getNodeType()) {

      case Node.DOCUMENT_NODE: {
        out.println("<th>login</th><th>email</th>");
        traverseTree(((Document)node).getDocumentElement(),out);
        break;
      }

      case Node.ELEMENT_NODE: {
        if(node.getNodeName().equals("user")) {
          out.println("</tr><tr>");
        }
```

DOM - przykład 2/3

```
        NodeList childNodes = node.getChildNodes();

        if(childNodes != null) {
          for (int i= 0; i<childNodes.getLength(); i++)
          {
            traverseTree(childNodes.item(i),out);
          }
        }
        break;
      }

      case Node.TEXT_NODE: {
        String value = node.getNodeValue().trim();
        if((value.indexOf("\n")<0) && (value.length()> 0)) {
          out.println("<td>"+value+"</td>");
        }
      }
    }
  }
%>
```

DOM - przykład 3/3

```
<%
  String uri = application.getRealPath("/przyklady/xml/users.xml");

  DOMParser parser = new DOMParser();
  parser.parse(uri);
  Document doc = parser.getDocument();
  out.println("<table border='2'><tr>");
  traverseTree(doc, out);
  out.println("</tr></table>");
%>
```

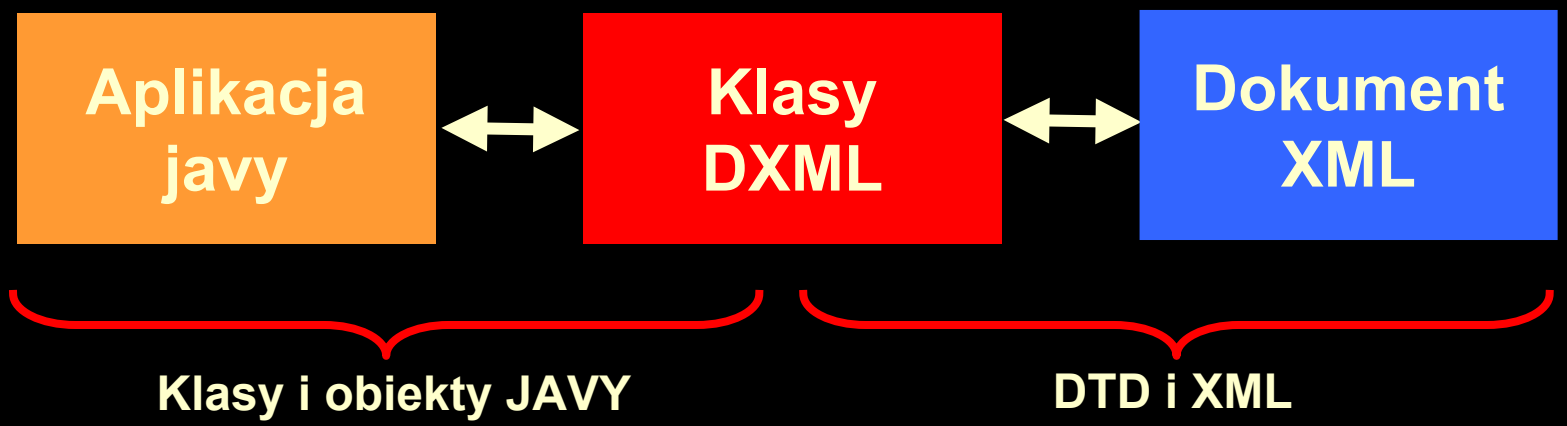
DXML (Dynamic XML)

Biblioteka DXML (ObjectSpace) umożliwia stworzenie wyższego poziomu abstrakcji w pracy z dokumentami XML

Na podstawie gramatyki dokumentu (DTD) następuje generacja zestawu klas javy pozwlająca pracować z danymi XML reprezentowanymi przez struktury obiektów

DXML (Dynamic XML)

Dane za pomocą klas javy mogą być modyfikowane i w prosty sposób ponownie zapisywane do dokumentu XML (przezroczystość)



Projekt COCOON

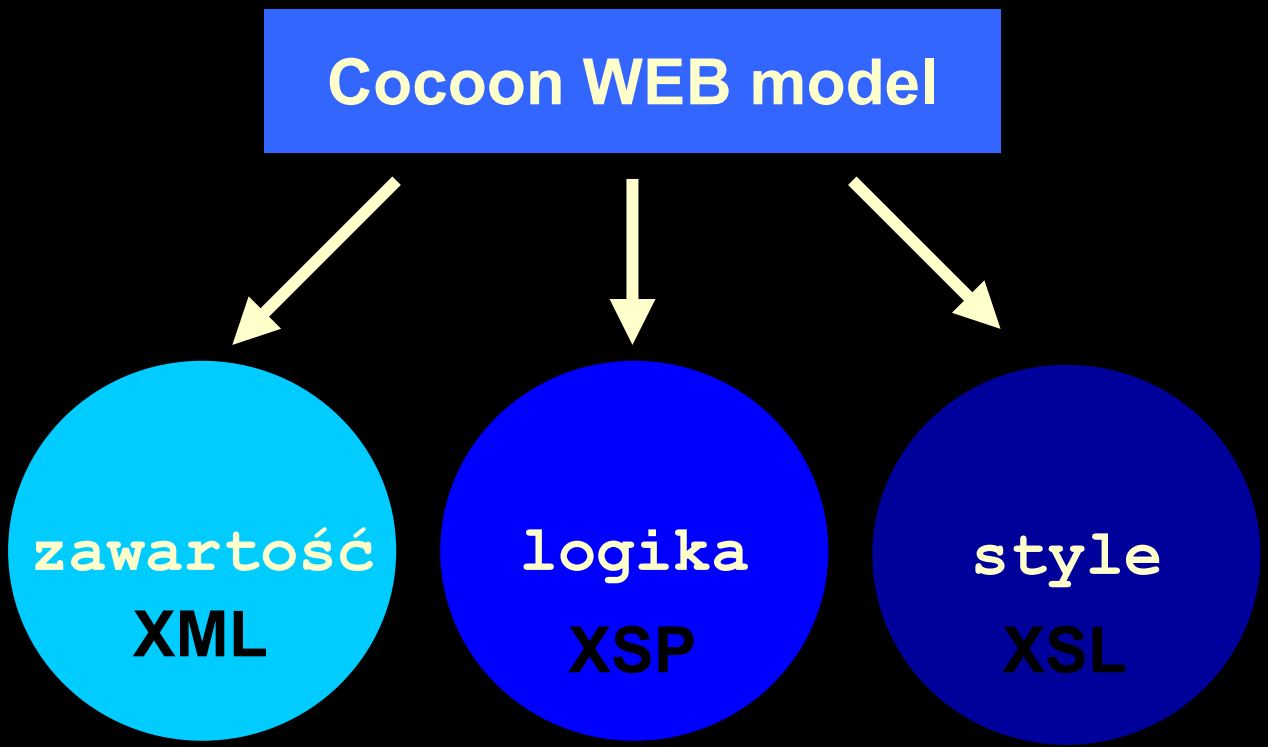
Cocoon - bezpłatne środowisko publikacyjne napisane w postaci serwletu javy

- dynamiczne przekształcanie dokumentów XML na inne formaty
- trójwarstwowy model publikowania (treść, logika, style)
- wielokrotne przekształcanie dokumentów

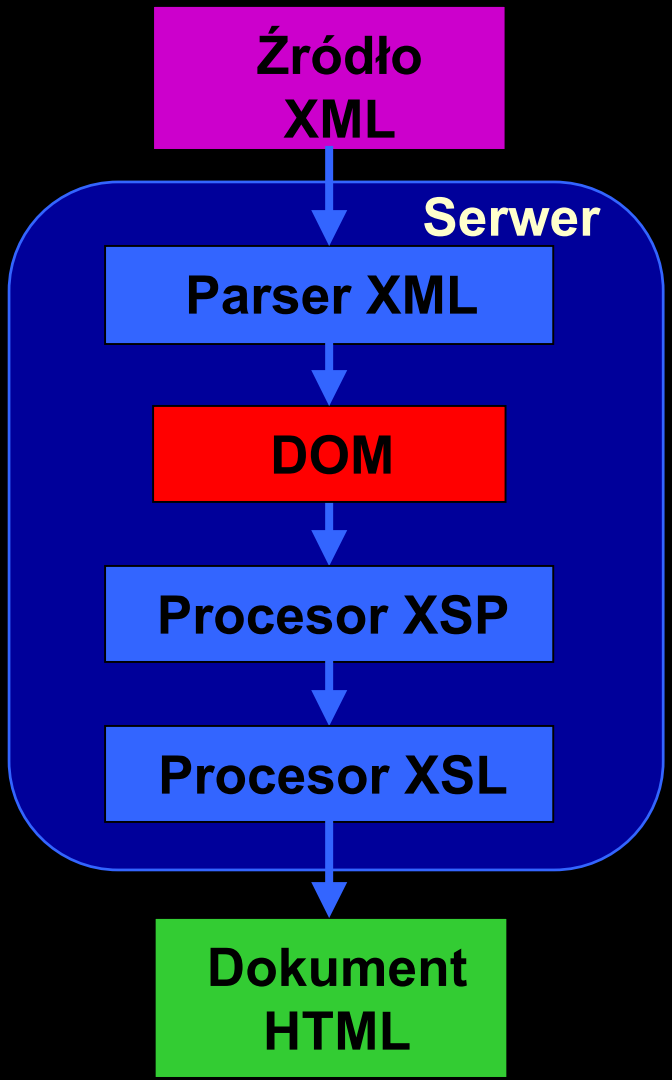
Trójwarstwowość modelu

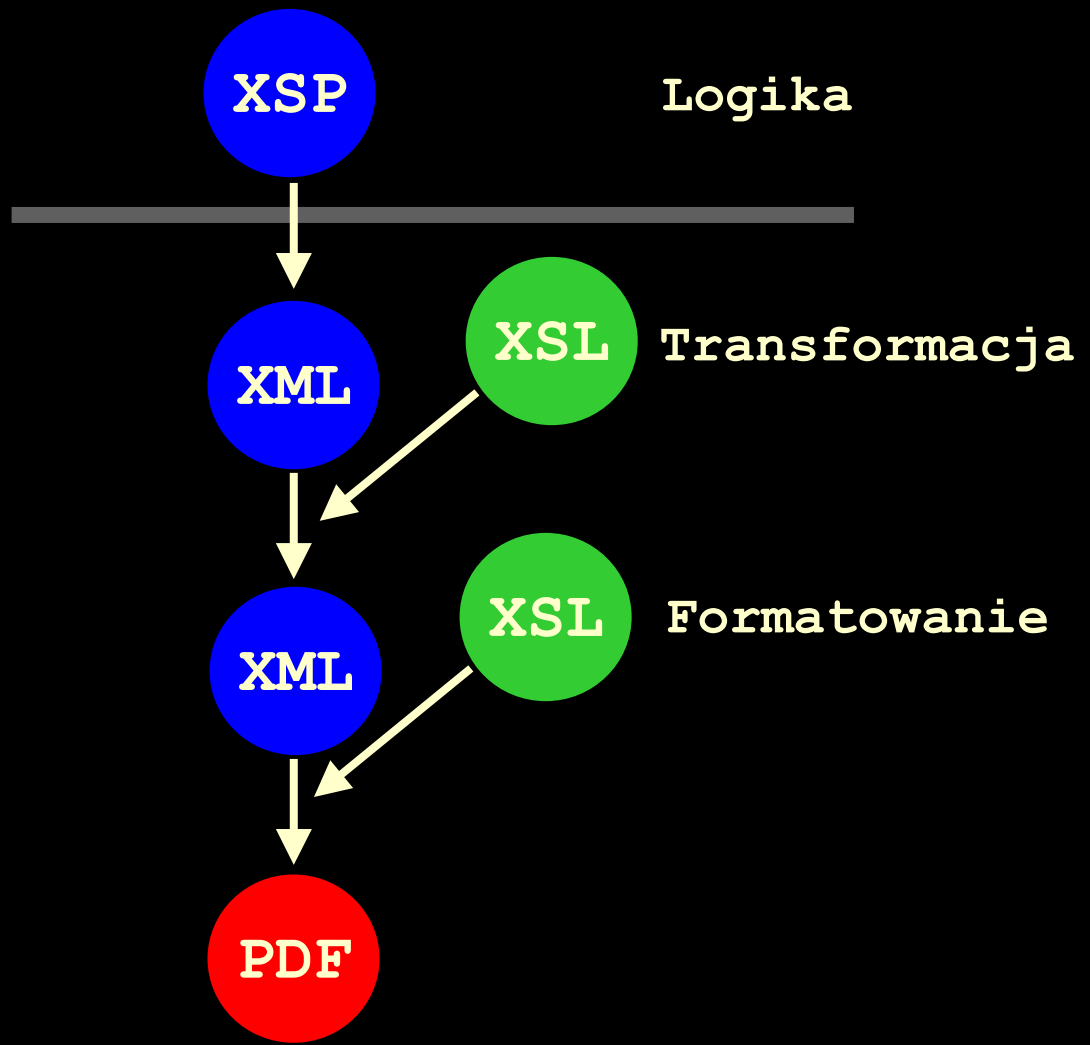
Transformacje dokumentów XML:

- HTML
- WML
- VoxML
- SVG
- VRML
- PDF



Projekt COCOON





Pakiet COCOON

Instalacja aplikacji cocoon:

- rozpakowanie pliku **cocoon.war** do katalogu **webapps** serwera Tomcat
- deklaracja aplikacji w pliku **server.xml** serwera Tomcat

```
<Context path="/cocoon" docBase="cocoon"
debug="0"      reloadable="true" trusted="false"/>
```

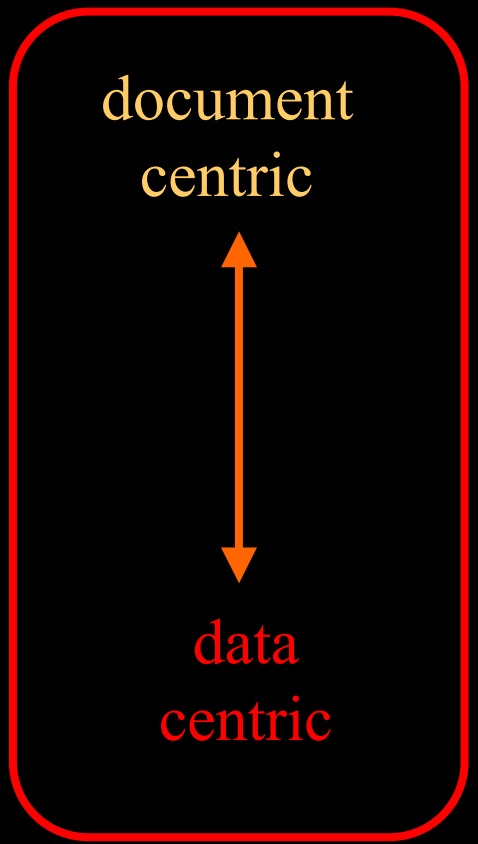
XML - bazy danych

Przechowywanie dokumentów:

- eksport i import dokumentów XML z baz przechowujących dane w wewnętrznej postaci innej niż XML (XML-enabled)
- przechowywanie danych w „naturalnej” postaci XML, bez ingerencji w strukturę - zarządzanie blokami tekstu (native XML)
- rozwiązania mieszane

Przechowywanie dokumentów XML

- Pure Relational - normalizacja danych XML do postaci rzędów i kolumn
- Post Relational - przechowywanie jako typ LOB (Large OBject) z możliwością przetwarzania i przeszukiwania na poziomie jednej komórki
- Native XML - przechowywanie danych w DBMS



XML - bazy danych

Rozwiązania bazodanowe dla XML:

- XDK (Oracle)
- XML Extender (IBM DB2)
- Microsoft SQL Server 2000
- Tamino (format natywny) (Software AG)
- Xml-xindice (projekt Apache)
- dbXML (Open Source)

Tendencje rozwojowe serwerów baz danych XML

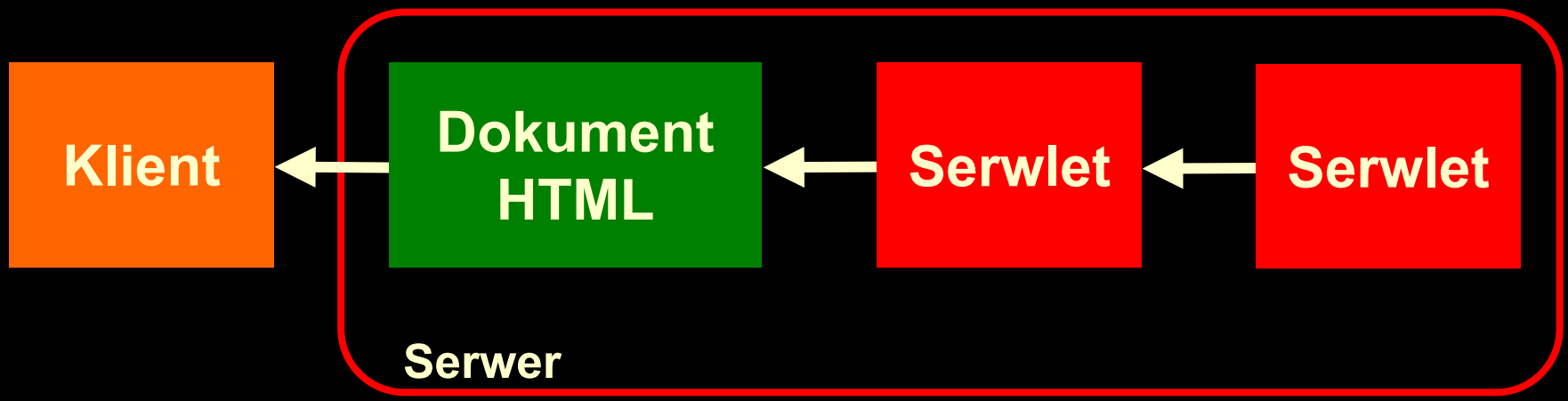
- graficzne edytory schematów
- centralizacja zarządzania
- narzędzia do masowej obróbki danych
- systemy pełnotekstowego wyszukiwania danych
- otrzymywanie wyników zapytań SQL w modelach DOM, SAX i JDOM (translacja, walidacja)

Serwlety

Serwlety

Serwlet - kod klasy java (komponentu) umożliwiający dynamiczną generację zawartości

- generowania stron HTML
- strumieniowania danych pomiędzy innymi serwletami



Serwlety

Zalety

- większa szybkość w porównaniu ze skryptami CGI (odmienny model przetwarzania)
- wszelkie zalety środowiska Java (niezależność sprzętowa oraz łatwość programowania)
- dostęp do wielu interfejsów API środowiska JAVA
- uniwersalność i automatyczna generacja serwletów z kodu Java Server Pages

JSP => Servlet => class

witaj$jsp.java

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class witaj$jsp extends HttpJspBase {

    static {
    }
    public witaj$jsp( ) {
    }

    private static boolean _jspx_inited = false;

    public final void _jspx_init() throws org.apache.jasper.runtime.JspException {
    }
...
...
```

witaj.jsp

```
<html>
  <head>
    <title>Witaj</title>
  </head>
<body>
            <% out.println("Witaj"); %>
</body>
</html>
```

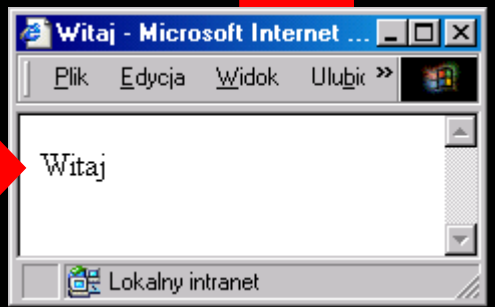
2

1

3

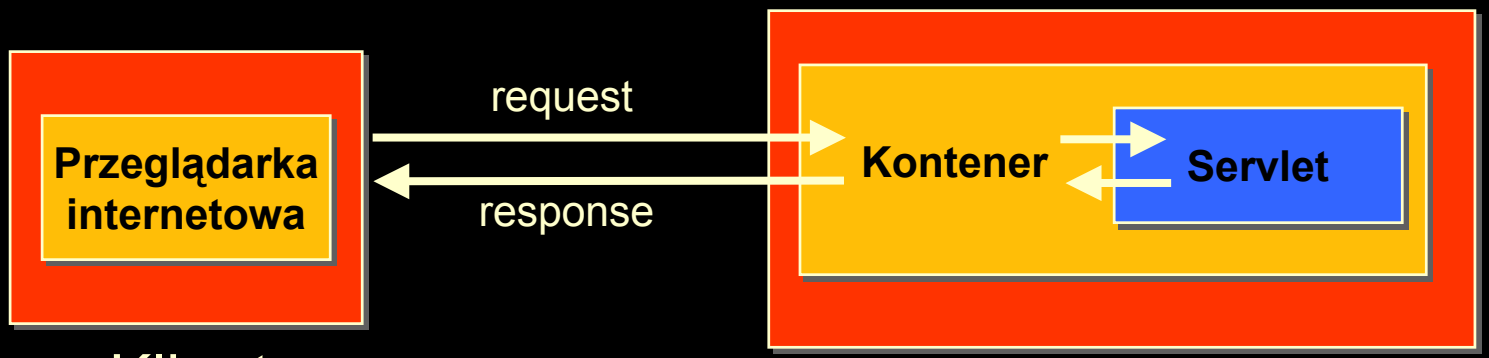
Witaj$jsp.class

4



Serwlety

Klasy serwletów implementują jeden z interfejsów zdefiniowanych w pakiecie `javax.servlet.*` lub `javax.servlet.http.*` i służą do obsługi żądań klienta

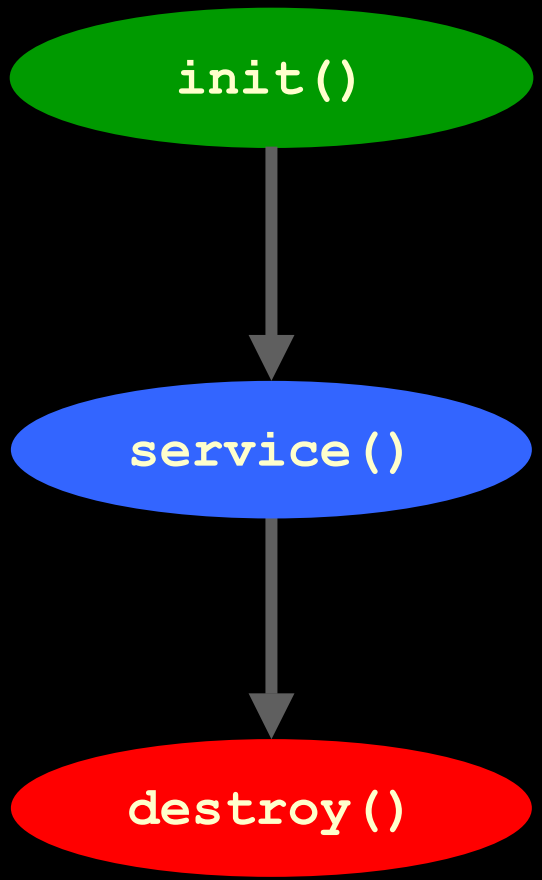


Kontener wywołuje odpowiednie metody serwletu

Serwlety - metody

Kontener wywołuje odpowiednie metody serwletu w zależności od

- etapu życia serwletu
  - `init()`
  - `destroy()`
- rodzaju żądania klienta
  - `services()`
  - `doGet()`, `doPost()`
  - `doPut()`, `doDelete()`
  - `doHead()`, `doOption()`
  - `doTrace()`

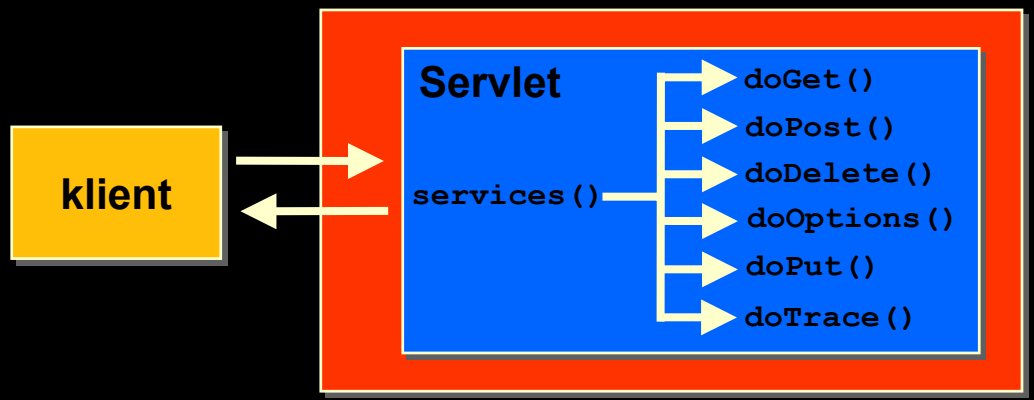


Serwlety - metody

Wybór metody `services()` lub metod `doXXX()` zależny jest od rodzaju klasy (dziedziczenie) na podstawie której budujemy własny serwlet

*Klasa **GenericServlet***
- metoda `services()`

*Klasa **HttpServlet***
- metody `doXXX()`
np. `doGet()`, `doPost()` itd...



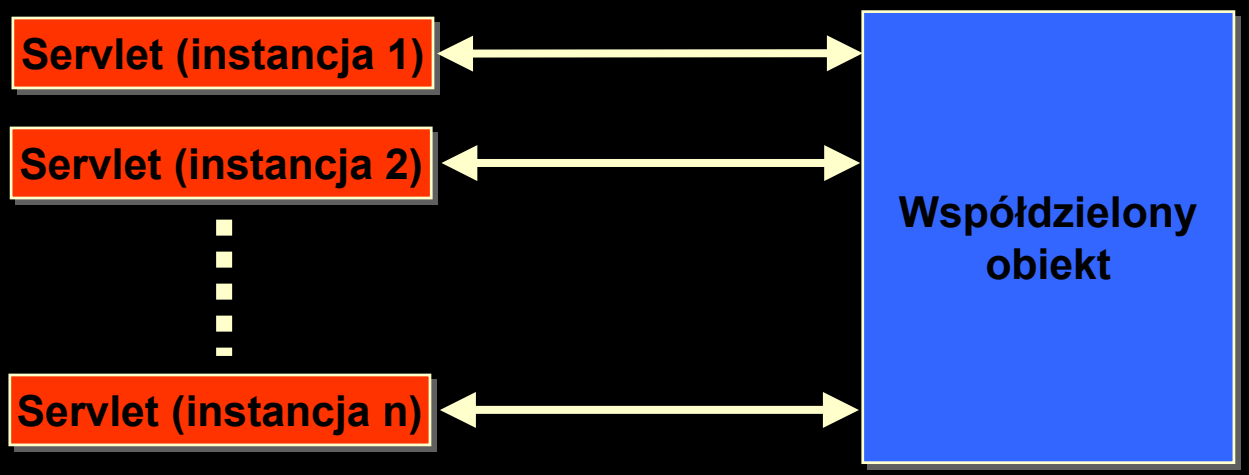
Nazwy metod dla klasy HttpServlet ściśle odpowiadają metodom żądań protokołu HTTP

Serwlety - metody

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

public class TestServlet extends HttpServlet {

  public void init() throws ServletException {...}

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

  public void doPost(HttpServletRequest request, HttpServletResponse response)
                                  throws ServletException, IOException {...}

  public void doPut(HttpServletRequest request, HttpServletResponse response)
                                  throws ServletException, IOException {...}

  public void doDelete(HttpServletRequest request, HttpServletResponse response)
                                  throws ServletException, IOException {...}

  public void destroy() {...}
}
```

Synchronizacja

Synchronizacja wymusza wykonywanie kodu serwletu w ten sposób, że będzie tylko jedno żądanie klienta dostępu do metod `service()` lub `doXXX()`



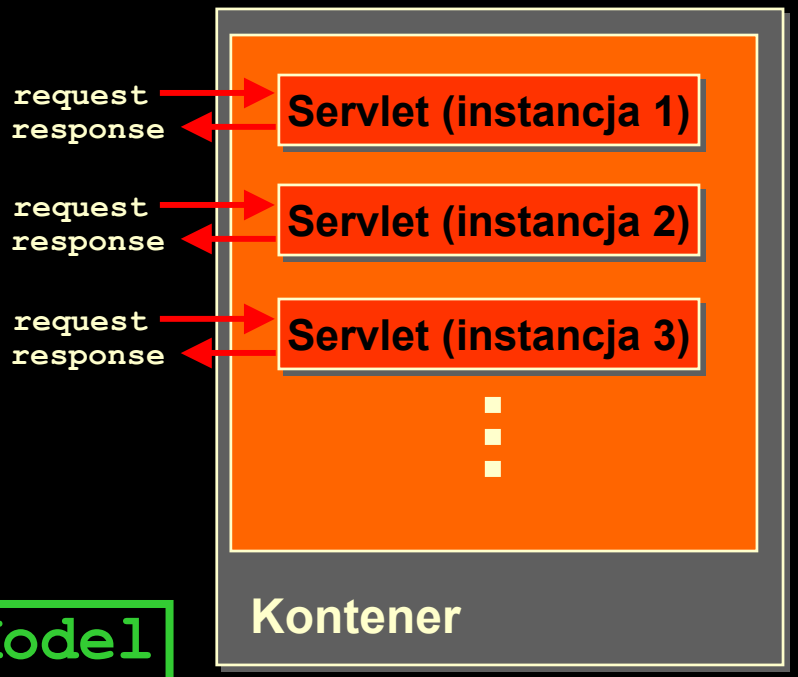
Wielowątkowość umożliwia jednoczesne wykonywanie się tego samego kodu przez różne wątki

Synchronizacja

Synchronizacja:

– blok kodu

```
synchronized(this) {
}
```



– pula instancji servleta

```
implements SingleThreadModel
```

– synchronizacja metod

```
public synchronized void NazwaMetody() {
}
```

Kompilacja serwletu

Kompilacja


```
javac -classpath ".;%TOMCAT_HOME%\common\lib\servlet.jar" TestServlet.java
```

Klasa serwletu (TestServlet.java)

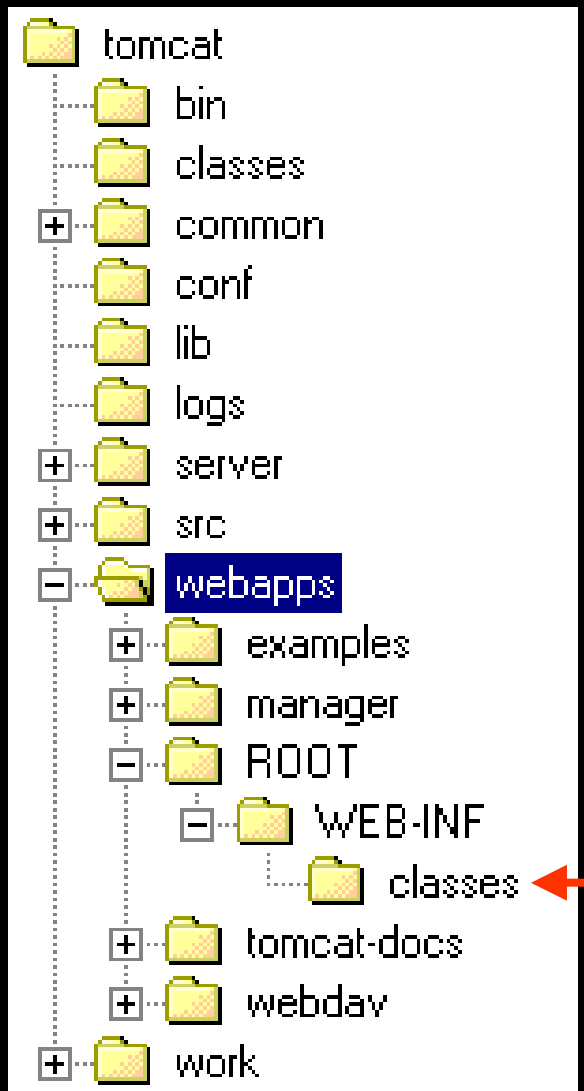
TestServlet.java - Notatnik

Plik Edycja Format Pomoc

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

public class TestServlet extends HttpServlet {

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

    PrintWriter out = response.getWriter();
    out.println("Witaj!");
  }
}
```

Nazwa klasy musi mieć taką samą nazwę jak plik

Lokalizacja serwletu

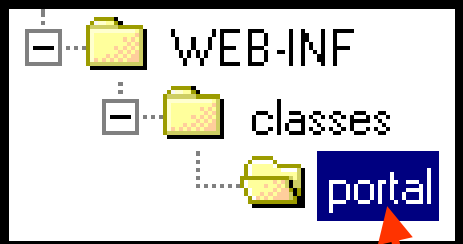


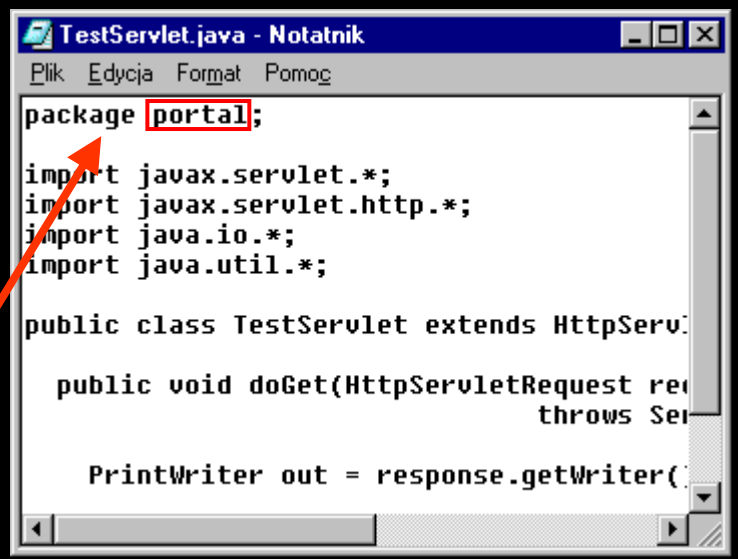
Skompilowany kod klasy serwletu `*.class` należy umieścić w katalogu `Web-inf\classes` projektowanej aplikacji webowej

Katalog w którym należy umieszczać klasy serwletów

Pakiety

W przypadku tworzenia pakietów należy dokonać zagnieżdżenia poszczególnych klas serwletów do określonych katalogów o nazwach odpowiadających nazwie pakietu

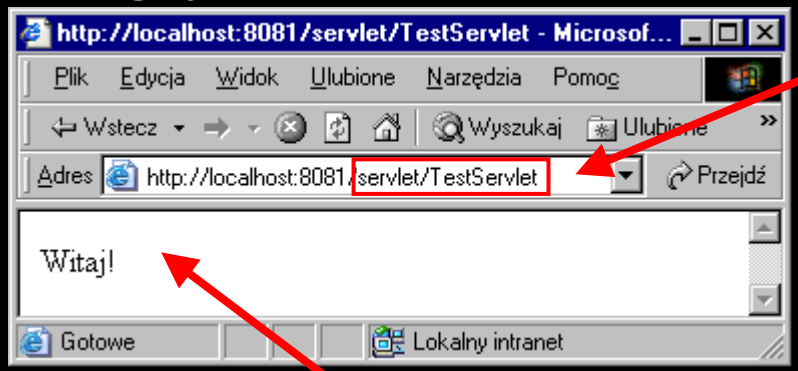




Katalog (zawiera klasę serwletu) odpowiada nazwie pakietu

Wywołanie serwletu

Przeglądarka (klient)



Wywołanie za pomocą metody GET (wprowadzenie bezpośredniego adresu do przeglądarki)

Klasa serwletu (serwer)

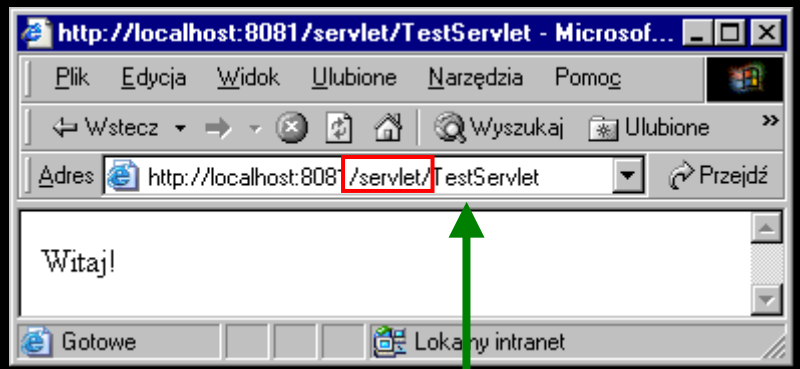
TestServlet.java - Notatnik

Plik Edycja Format Pomoc

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

public class TestServlet extends HttpServlet {

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

    PrintWriter out = response.getWriter();
    out.println("Witaj!");
  }
}
```

Mapowanie przedrostka servlet/*

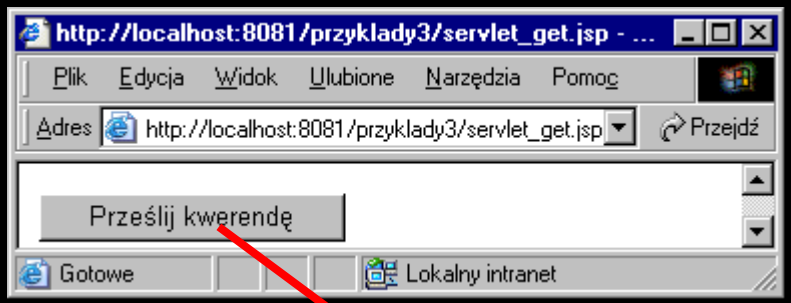
```
<servlet-mapping>
  <servlet-name>invoker</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```



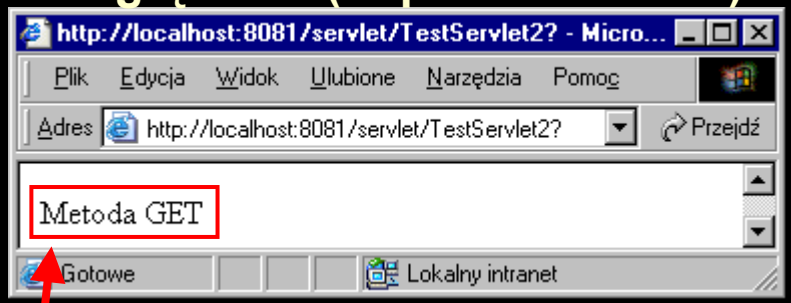
http://localhost:8080/servlet/nazwa_klasy_servletu

Metod GET

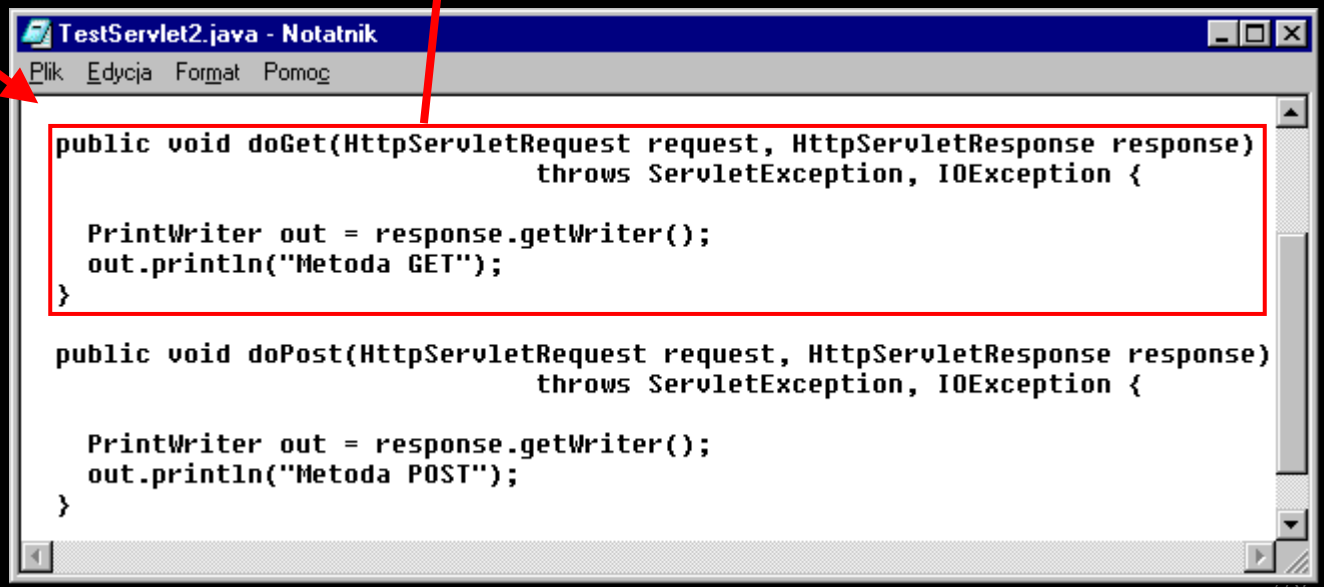
Przeglądarka (żądaie klienta - GET)



Przeglądarka (odpowiedź klient)

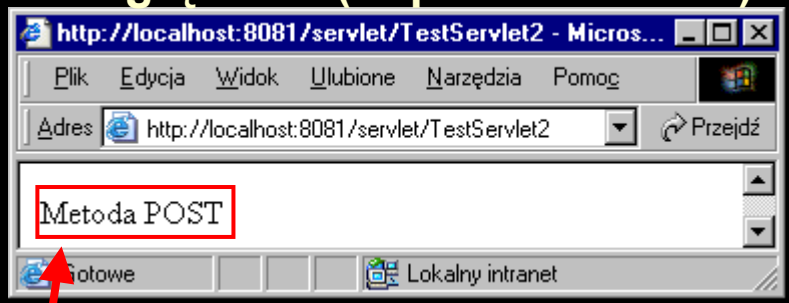


Klasa serwletu (serwer)

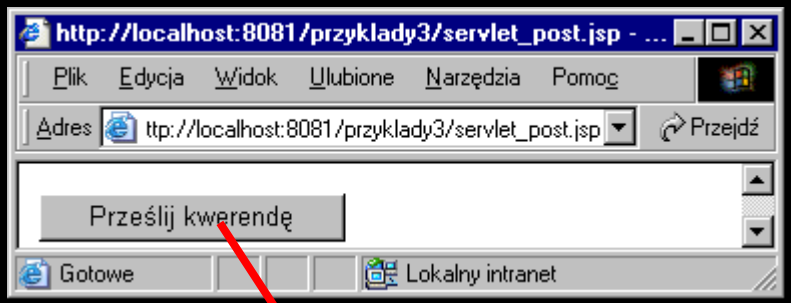


Metod POST

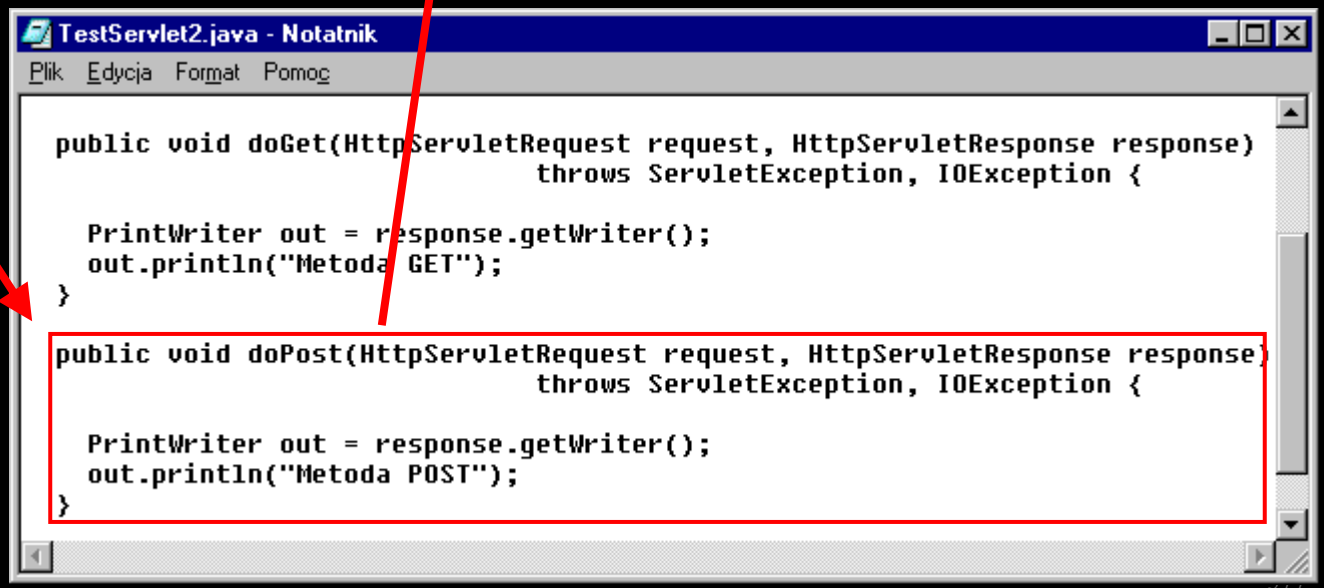
Przeglądarka (odpowiedź klient)



Przeglądarka (żądaie klienta - POST)

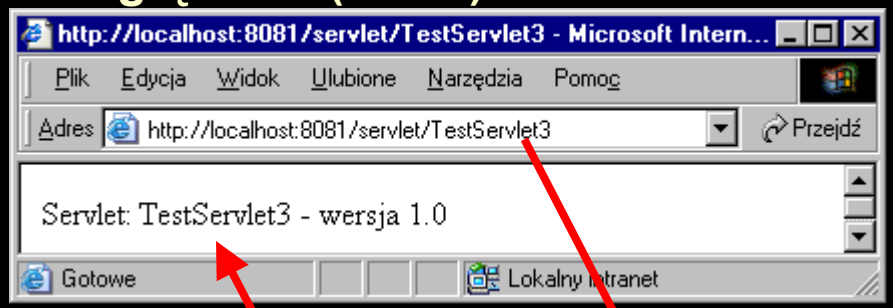


Klasa serwletu (serwer)



getServletInfo

Przeglądarka (klient)



Metoda getServletInfo() służy do umieszczania informacji o servlecie

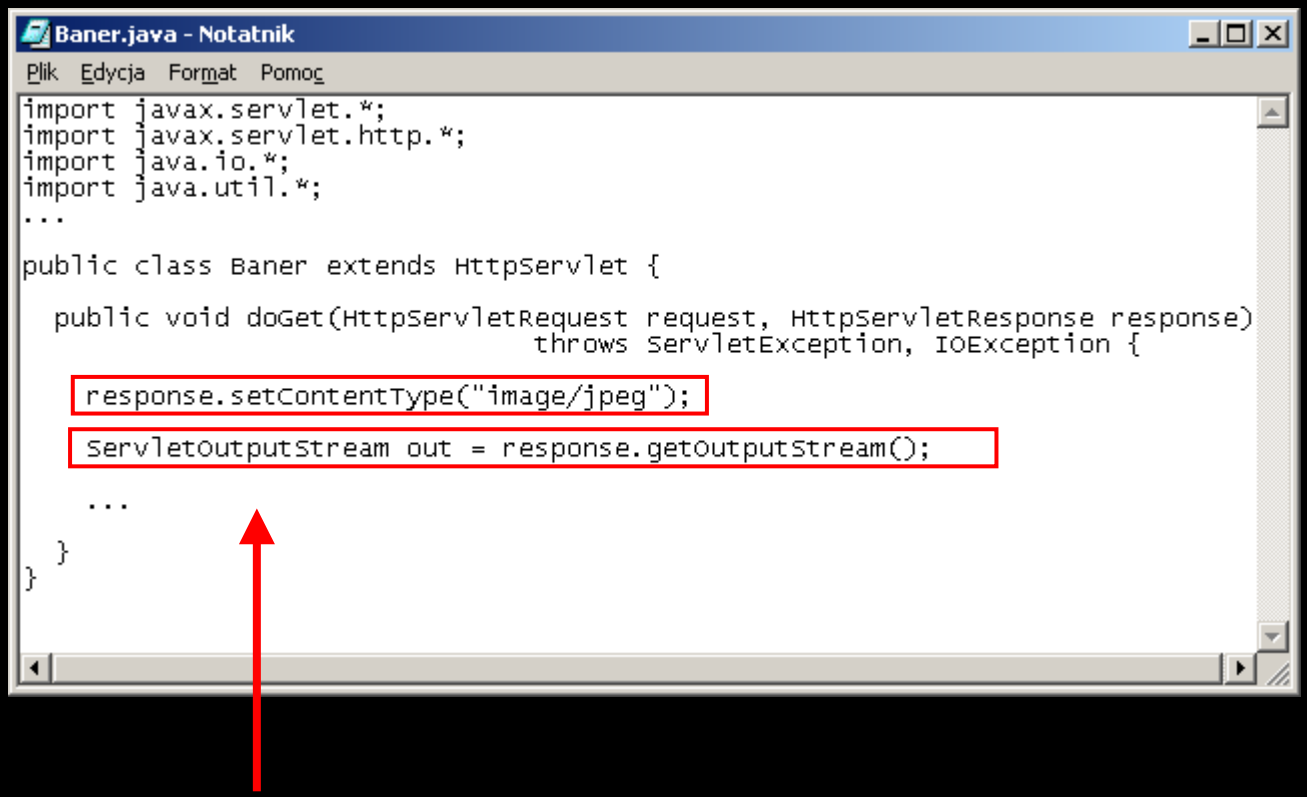
Klasa serwletu (serwer)

TestServlet3.java - Notatnik

Plik Edycja Format Pomoc

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

public class TestServlet3 extends HttpServlet {

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

    PrintWriter out = response.getWriter();
    out.println(getServletInfo());
  }

  public java.lang.String getServletInfo() {
    return "Servlet: TestServlet3 - wersja 1.0";
  }
}
```

Generowanie plików binarnych



Ustawienie typu MIME generowanego dokumentu oraz utworzenie strumienia do przesyłania danych binarnych

Baner.java - Notatnik

Plik Edycja Format Pomoc

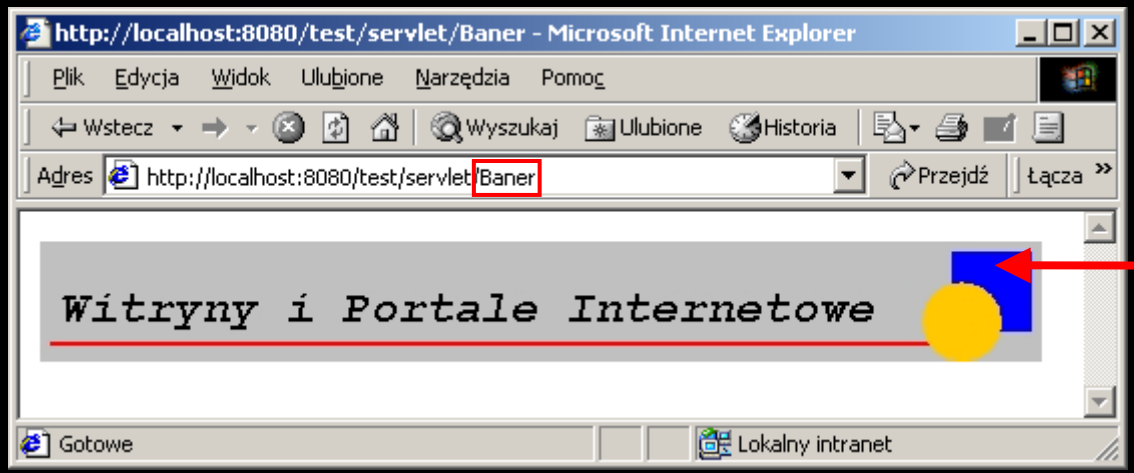
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;
import com.sun.image.codec.jpeg.*;

public class Baner extends HttpServlet {

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

    response.setContentType("image/jpeg");

    ServletOutputStream out = response.getOutputStream();

    BufferedImage image = null;
    Graphics g = null;

    try {
      image =new BufferedImage(500,60,BufferedImage.TYPE_INT_RGB);
      g = image.getGraphics();
      g.setColor(Color.lightGray);
      g.fillRect(0,0,500,60);
      g.setColor(Color.red);
      g.fillRect(5,50,450,2);
      g.setColor(Color.blue);
      g.fillRect(455,5,40,40);
      g.setColor(Color.orange);
      g.fillOval(440,20,40,40);

      g.setColor(Color.black);
      g.setFont(new Font("Monospaced",Font.BOLD|Font.ITALIC,23));
      g.drawString("Witryny i Portale Internetowe",10,40);

      JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
      JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(image);
      param.setQuality(1.0f,false);
      encoder.setJPEGEncodeParam(param);
      encoder.encode(image);
    }
    finally {
      if(g!=null) g.dispose();
    }
  }
}
```

Biblioteka do obsługi plików JPEG (standard w JDK)

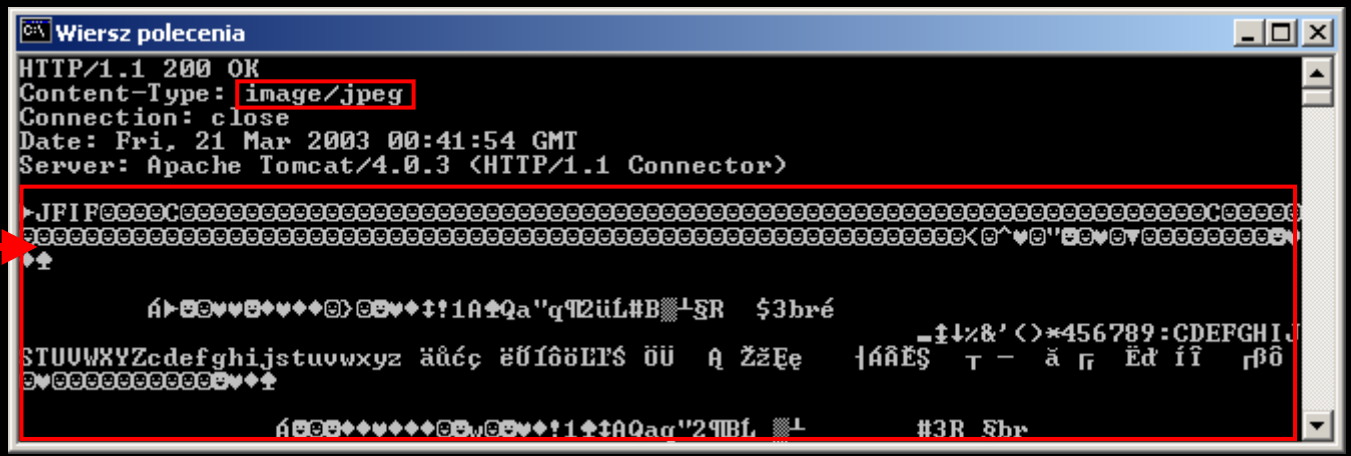
Rysowanie

Kompresja

Baner.java → image/jpeg



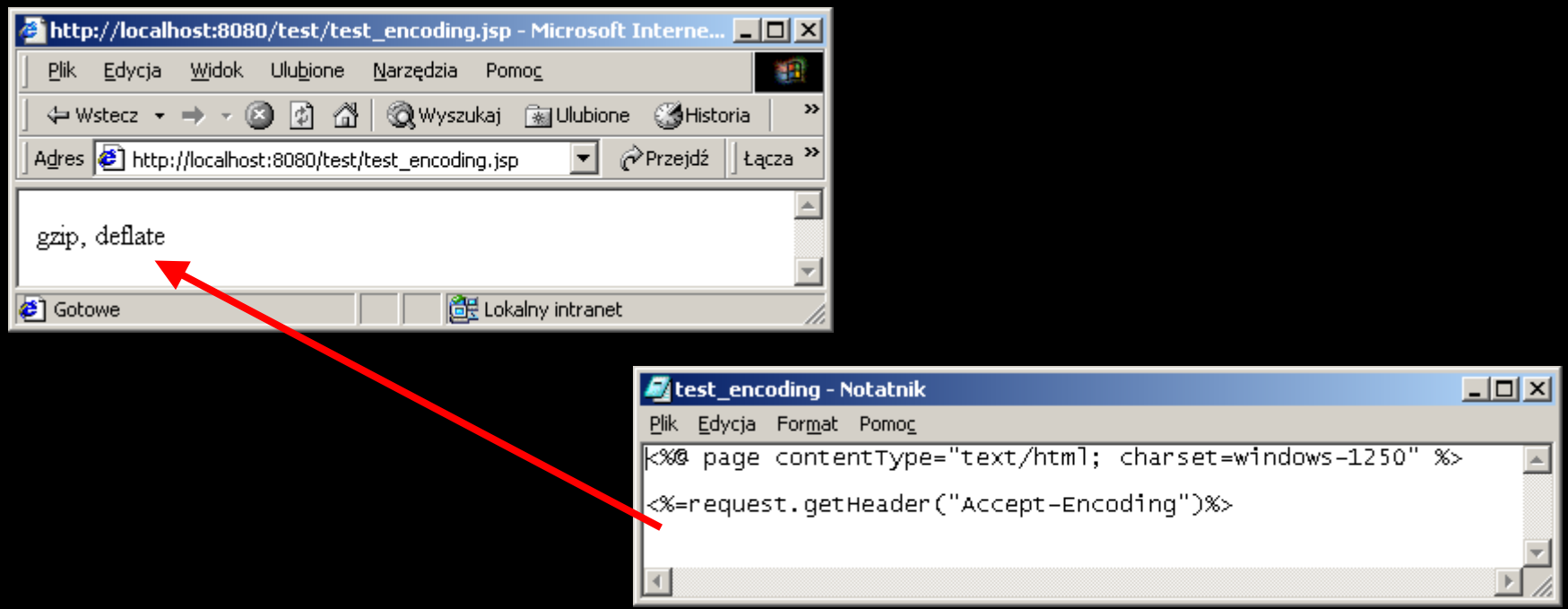
Wynik kodu servletu Baner.class



Binarne dane obrazka

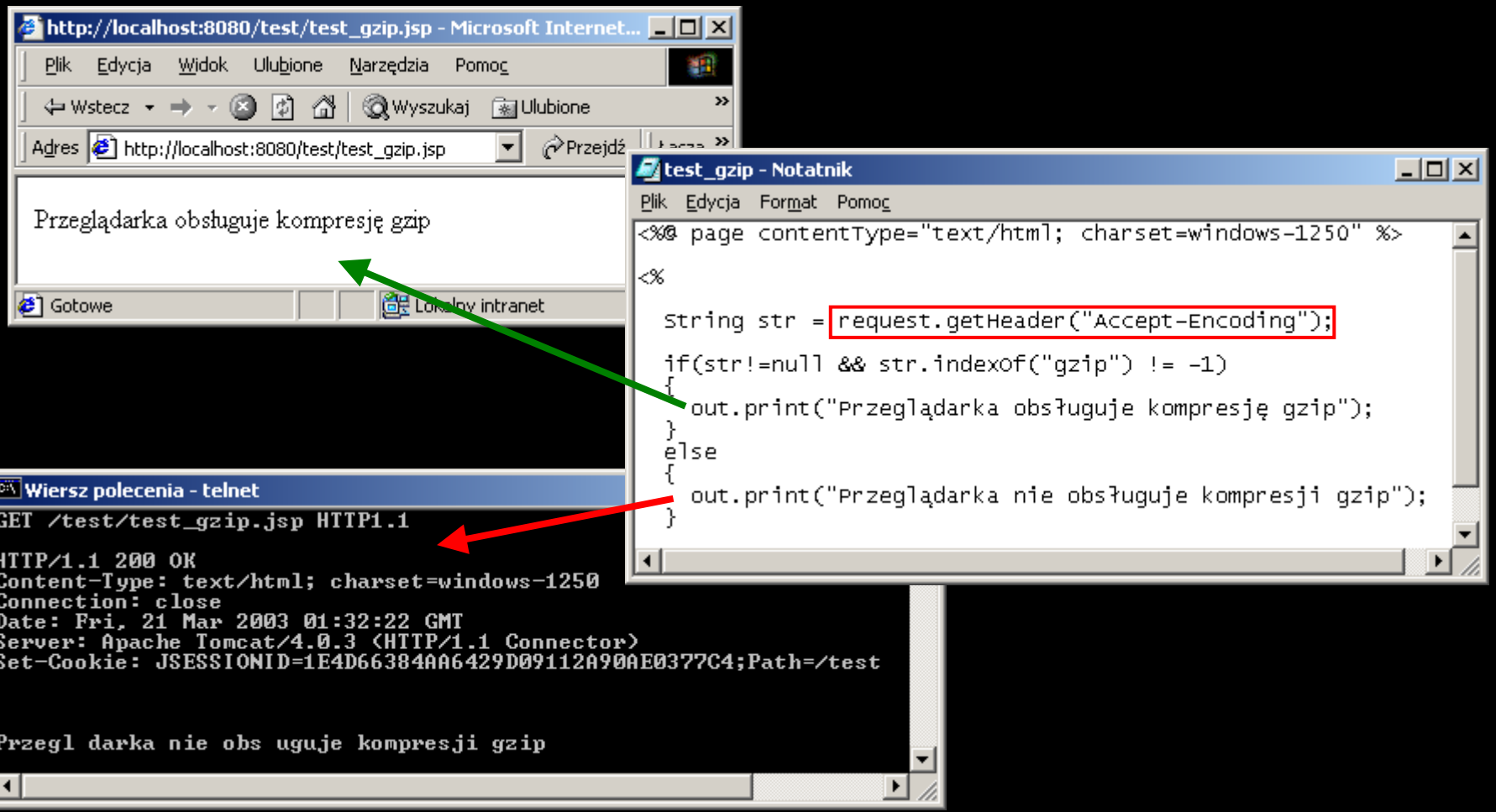
Accept-Encoding

Zbieranie informacji o możliwościach klienta do obsługi kodowania przesyłanych informacji



Schematy kodowania specyfikacji HTTP

Test kompresji gzip

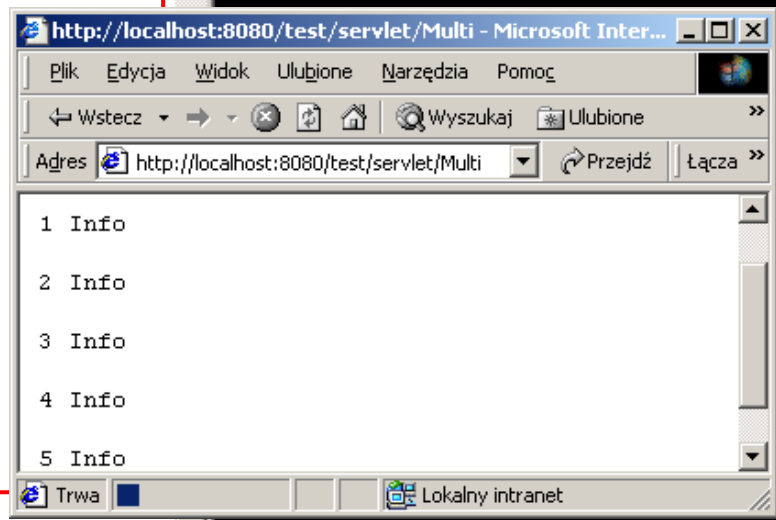


Servlet cykliczny

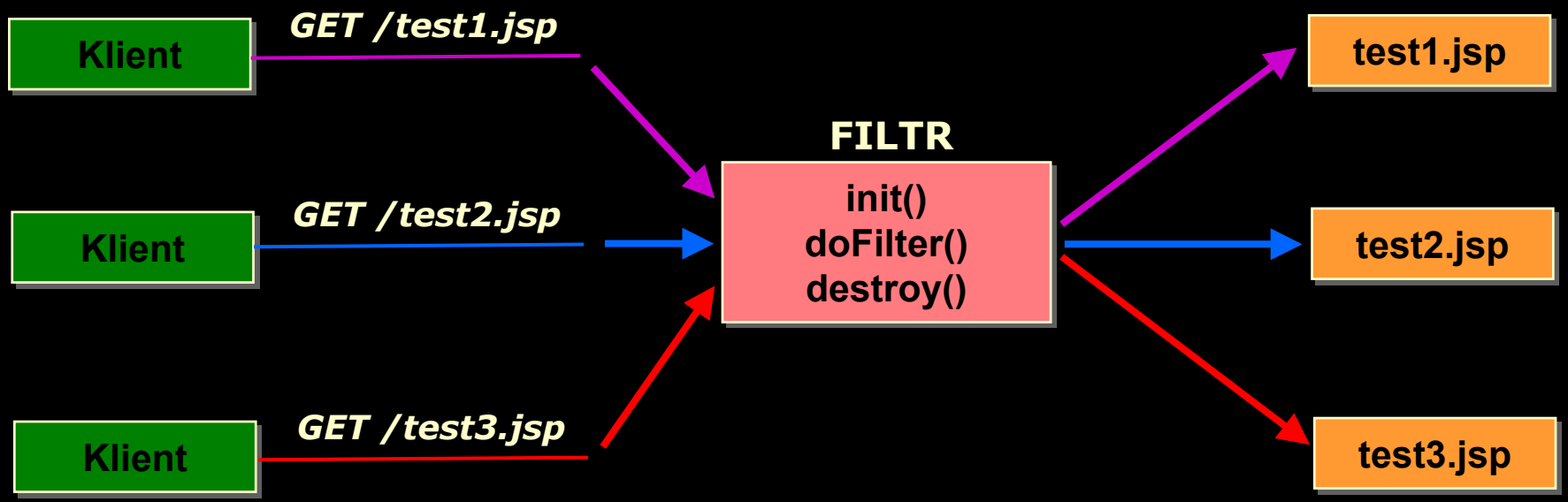
Lister - [c:\java\tomcat\webapps\multi\Web-inf\src\Multi.java]

File Edit Options Help 100 %

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

public class Multi extends HttpServlet {

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

    response.setContentType("multipart/x-mixed-replace; boundary=End");
    //ServletOutputStream out = response.getOutputStream();

    PrintWriter out = response.getWriter();

    out.println();

    for(int i=0;i<1000;i++) {

      out.println(i+" <<====");
      out.println();
      out.flush();
      try { Thread.sleep(2000); }catch (InterruptedException e){}
    }



    out.println("Koniec");
    out.flush();
  }
}
```



Filtrowanie



Filtry – obiekty umożliwiające zmianę postaci żądania i odpowiedzi

Przykład - TimeFilter

Powiązanie klasy filtru TimeFilter z nazwą timeFilter

web.xml

web - Notatnik

Plik Edycja Format Pomoc

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <filter>
    <filter-name>timerFilter</filter-name>
    <filter-class>TimerFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>timerFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

</web-app>
```

Do filtru o nazwie timeFilter będą przekazywane żądania spełniające kryterium adresu URL (/*) – *wszystkie żądania*

Przykład - TimeFilter

%TOMCAT_HOME%/logs/...log

localhost_log.2003-03-22 - Notatnik

Plik Edycja Format Pomoc

```
2003-03-12 12:53:13 StandardWrapper[/filtr:default]:
2003-03-12 12:53:13 StandardWrapper[/filtr:invoker]:
2003-03-12 12:54:23 /filtr/: 60ms
2003-03-12 12:54:25 /filtr/test1.jsp: 220ms
2003-03-12 12:54:27 /filtr/test2.jsp: 40ms
2003-03-12 12:54:29 /filtr/test3.jsp: 30ms
```

TimerFilter.java - Notatnik

Plik Edycja Format Pomoc

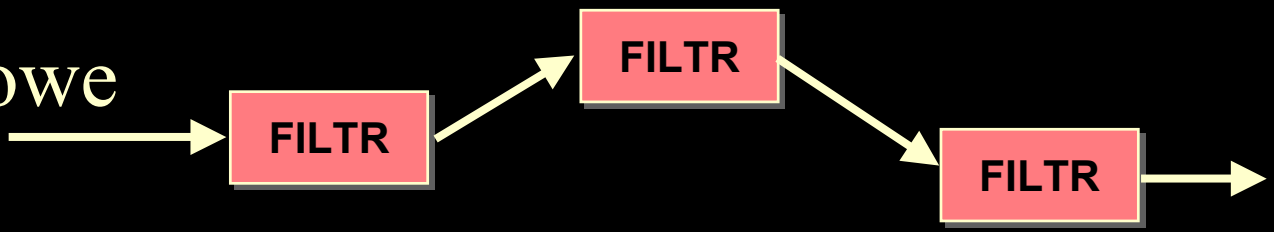
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class TimerFilter implements Filter {

  private FilterConfig config = null;

  public void init(FilterConfig config) throws ServletException {
    this.config = config;
  }

  public void destroy() {
    config = null;
  }

  public void doFilter(ServletRequest request, ServletResponse response,
                       FilterChain chain) throws IOException, ServletException {
    long before = System.currentTimeMillis();
    chain.doFilter(request, response);
    long after = System.currentTimeMillis();

    String name = "";
    if (request instanceof HttpServletRequest) {
      name = ((HttpServletRequest)request).getRequestURI();
    }
    config.getServletContext().log(name + ": " + (after - before) + "ms");
  }
}
```

Filtrowanie

- Wstępne przetwarzanie żądania oraz przetwarzanie odpowiedzi servlet'a
  - przechwytywanie wywołania servletu przed jego wywołaniem
  - sprawdzanie żądania przed wywołaniem servletu
  - przechwytywanie wywołania servletu po jego wykonaniu
  - możliwość modyfikacji nagłówka i danych odpowiedzi

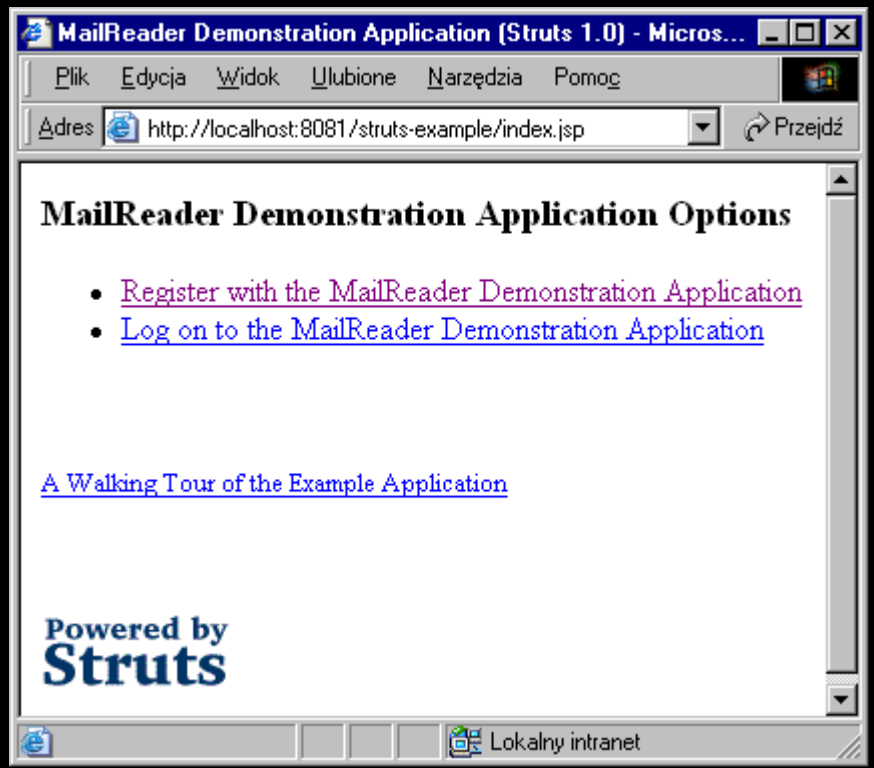
Rodzaje filtrów

- filtry uwierzytelniania i logowania
- filtry konwersji plików graficznych
- filtry kompresji danych
- filtry kodowania danych
- filtry przetwarzające pliki XML (XSLT)
- filtry dzielenia zasobów
- filtry łańcuchowe



Serwlet szkieletowy Struts

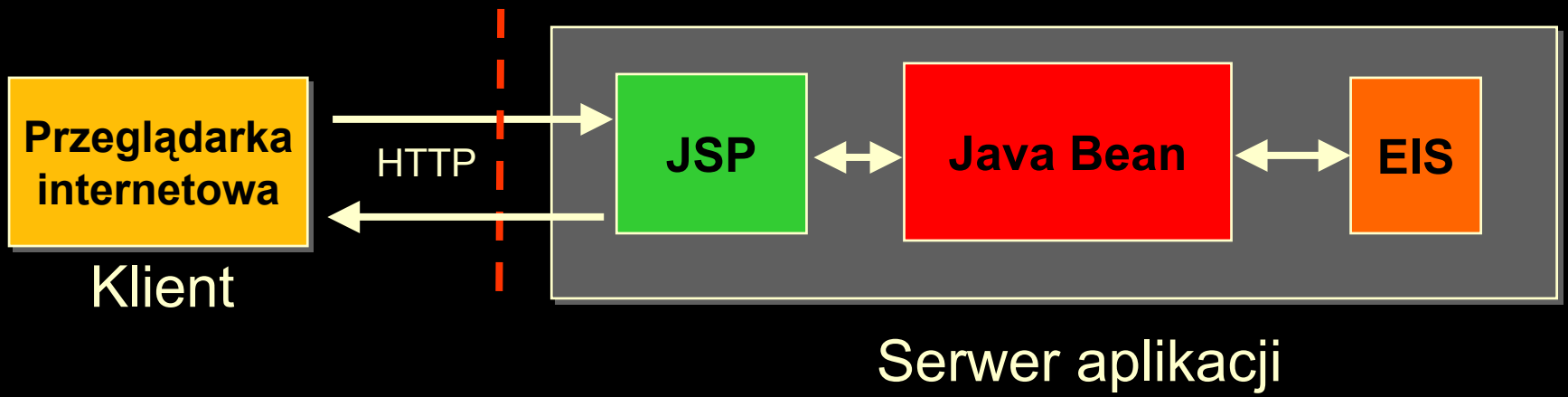
Struts - serwlet szkieletowy (frameWORK) umożliwiający budowanie aplikacji webowych z wykorzystaniem wzorca projektowego MVC (model-widok-kontroler)



Model 1

Rozdzielenie widoku od danych aplikacji

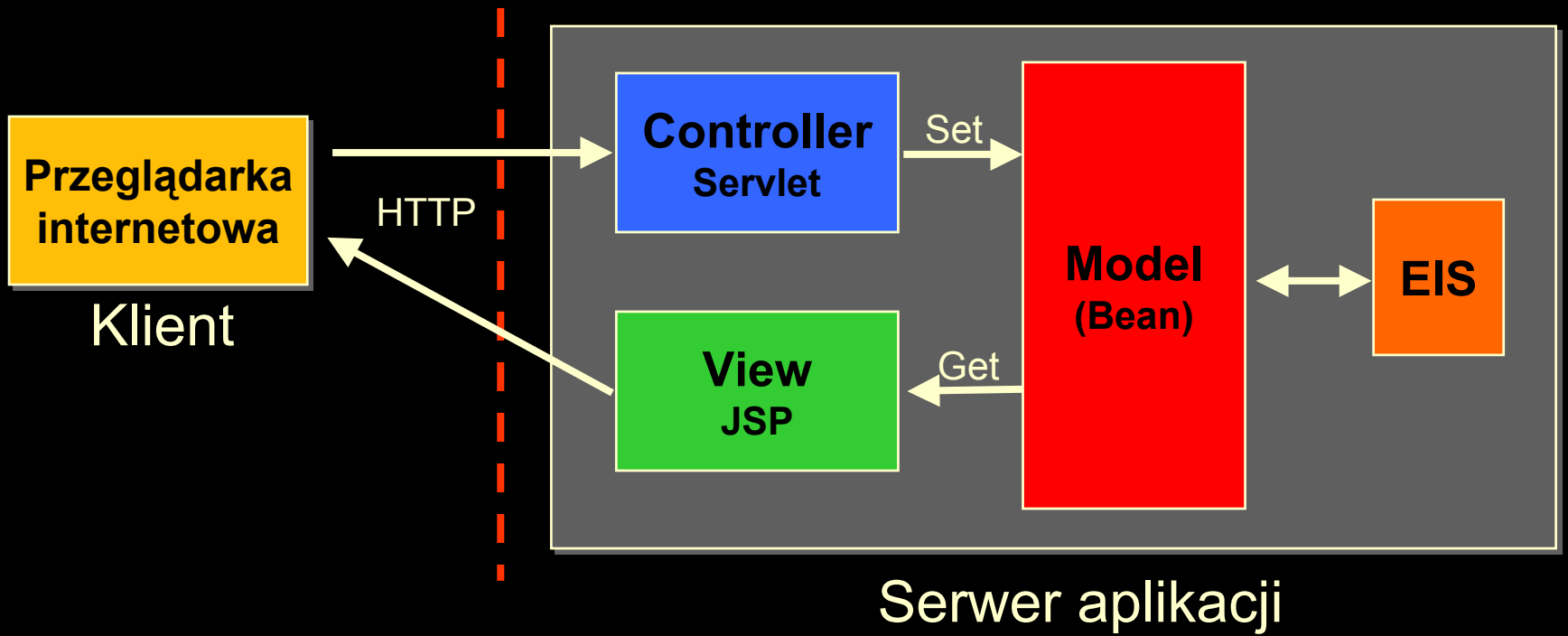
- widok (jsp)
- logika (java Bean)
- dane (EIS)



Model 2 (MVC)

Wzorzec projektowy

MVC - Model View Controller



Servlet szkieletowy Struts

Struts - zbiór klas javy, komponentów java beanów i stron jsp sprzężonych z servletem sterującym

Biblioteka wspomaga rozwój aplikacji webowych

- szablony stron JSP
- oddzielenie widoku od komponentów biznesowych (MVC)
- biblioteka custom tags rozszerzająca funkcjonalność stron JSP
- serwlet sterujący zarządzający akcjami

Struts - szablony

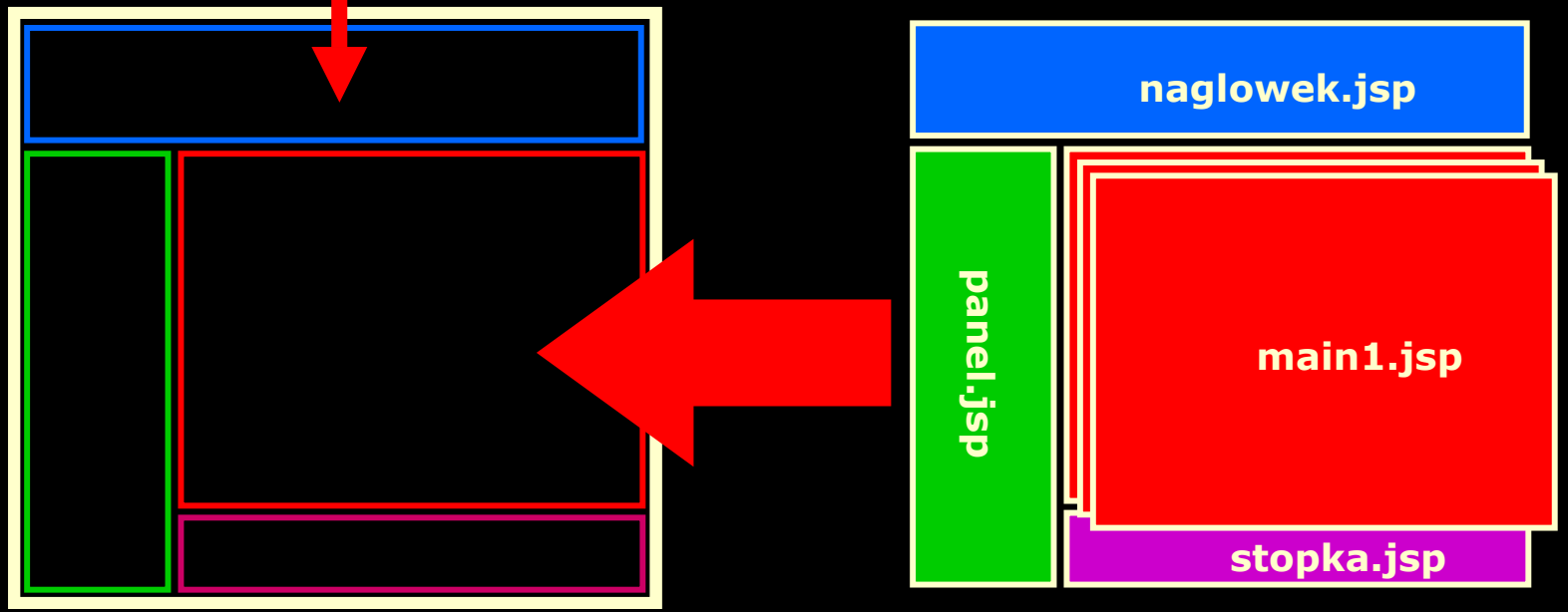
Strona JSP (serwer)

test1.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; windows-1250" %>
<%@ taglib uri="/WEB-INF/struts-template.tld" prefix="template" %>

<template:insert template="template/template.jsp">
  <template:put name="naglowek" content="naglowek.jsp"/>
  <template:put name="panel" content="panel.jsp"/>
  <template:put name="main" content="main1.jsp"/>
  <template:put name="stopka" content="stopka.jsp"/>
</template:insert>
```

test1.jsp



Web Archive

WAR - Web Archive

Aplikacje webowe mogą zostać spakowane do pliku Web Archive (WAR)

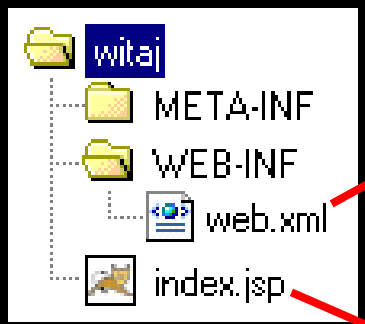
Pakowanie aplikacji:

- przejście do katalogu aplikacji np. witaj
- uruchomienie archiwizatora jar

```
jar -cvf witaj.war *
```

- w bieżącym katalogu powstaje plik witaj.war

WAR - Web Archive



web.xml - Notatnik

Plik Edycja Format Pomoc

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
</web-app>
```

index.jsp - Notatnik

Plik Edycja Format Pomoc

```
<html>
  <head>
    <title>Witaj</title>
  </head>
<body>
        <% out.println("Witaj"); %>
</body>
```

witaj.war

*Katalog **META-INF** zawiera plik **Manifest.mf** automatycznie tworzony przez archiwizator jar (Java Archive)*

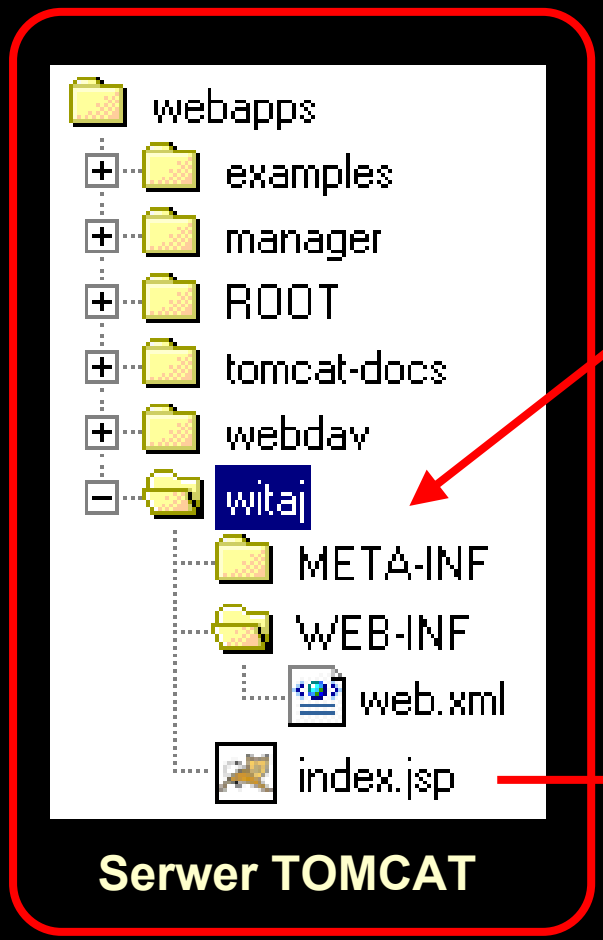
WAR - Web Archive

Instalacja archiwum na serwerze:

- umieszczenie pliku archiwum WAR do katalogu webapps serwera Tomcat
- restart serwera

Ponowne uruchomienie serwera powoduje, że serwer automatycznie rozpakowuje archiwa WAR (tworzony jest katalog z zawartością aplikacji archiwum webowego)

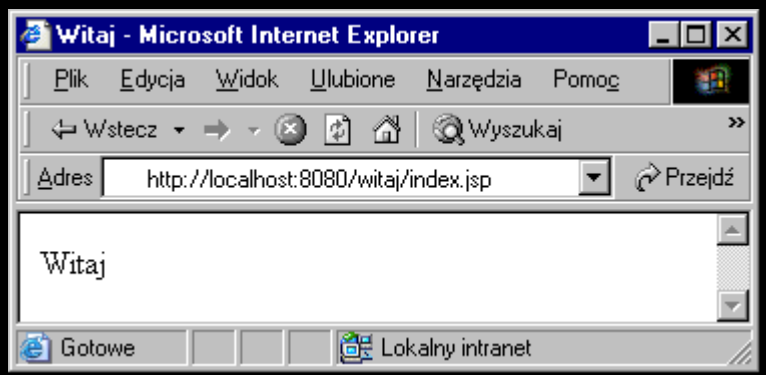
WAR - Web Archive



Dekompresja archiwum webowego

witaj.war

Przeglądarka (klient)

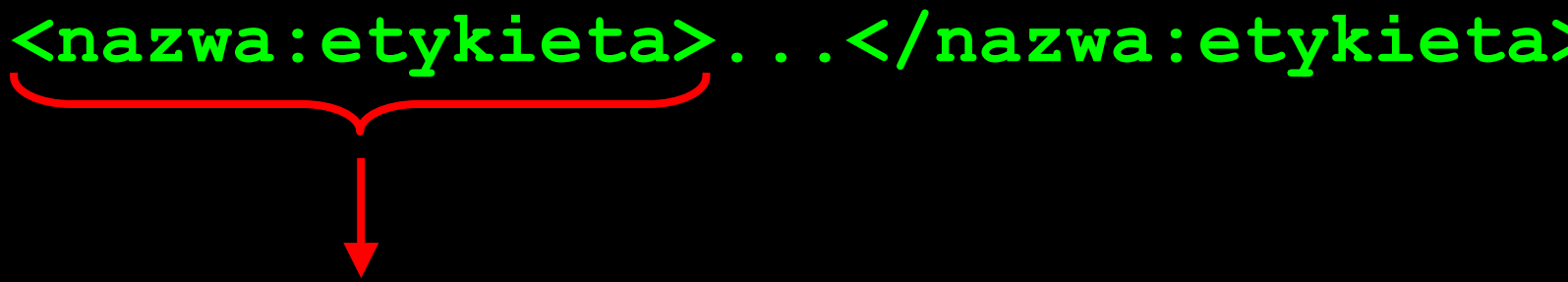


TagLibs

TagLibs

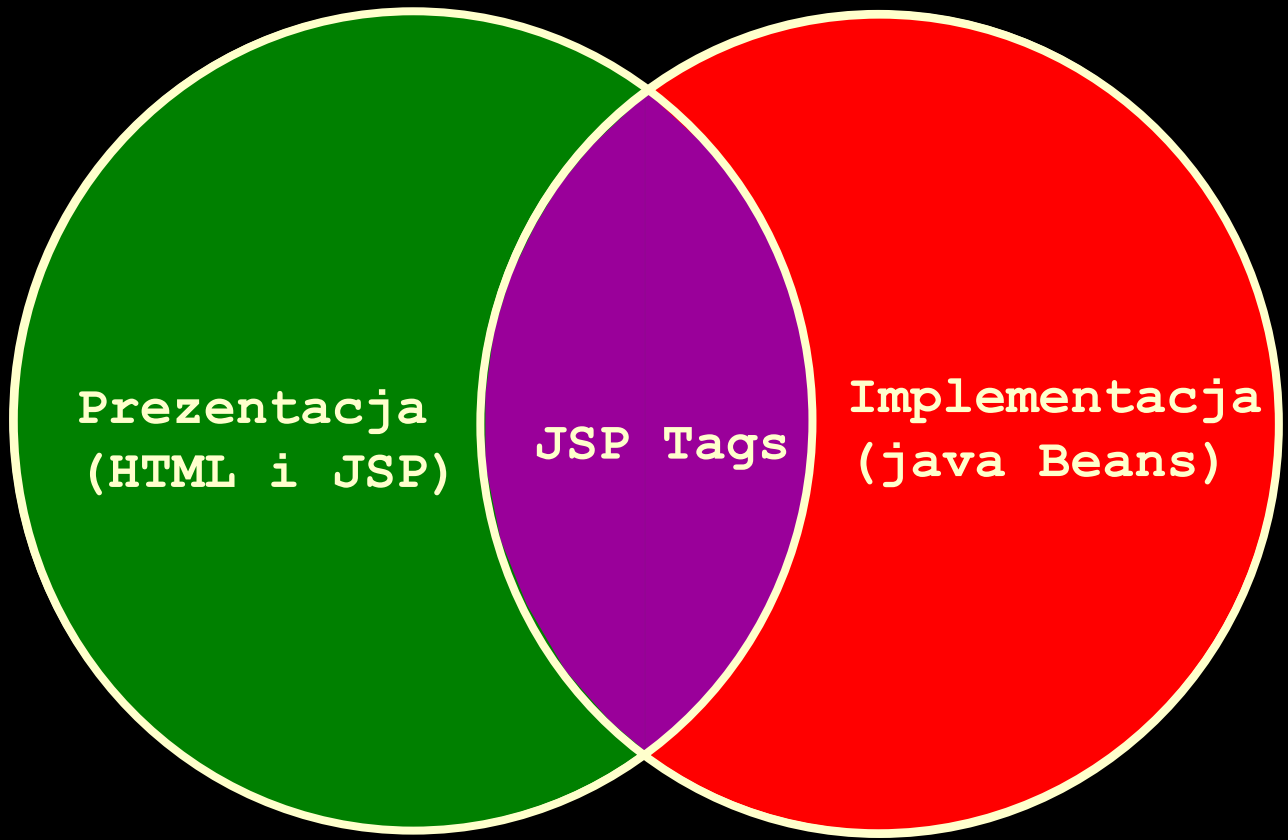
Mechanizm tworzenia własnych bibliotek TagLibs

- projektowanie własnych znaczników (znaczniki w budowie przypominają standardowe znaczniki HTML)



Po napotkaniu znacznika w kodzie strony JSP następuje wywołanie kodu Javy określonego znacznika

TagLibs



JSP Tags *- połączenie wywoływania metod klas javy z funkcjonalnością wywoływania znaczników w JSP*

TagLibs

Zalety

- umieszczanie w kodzie strony JSP prostych znaczników (funkcjonalność, zarządzanie, przenośność)
- możliwość wielokrotnego wykorzystania kodu klas znaczników
- podział pracy zespołu projektowego na projektantów bibliotek oraz integratorów korzystających z zaprojektowanych bibliotek

Deklaracja biblioteki znaczników

Dyrektywa taglib deklaruje wykorzystanie biblioteki znaczników zdefiniowanych przez użytkownika

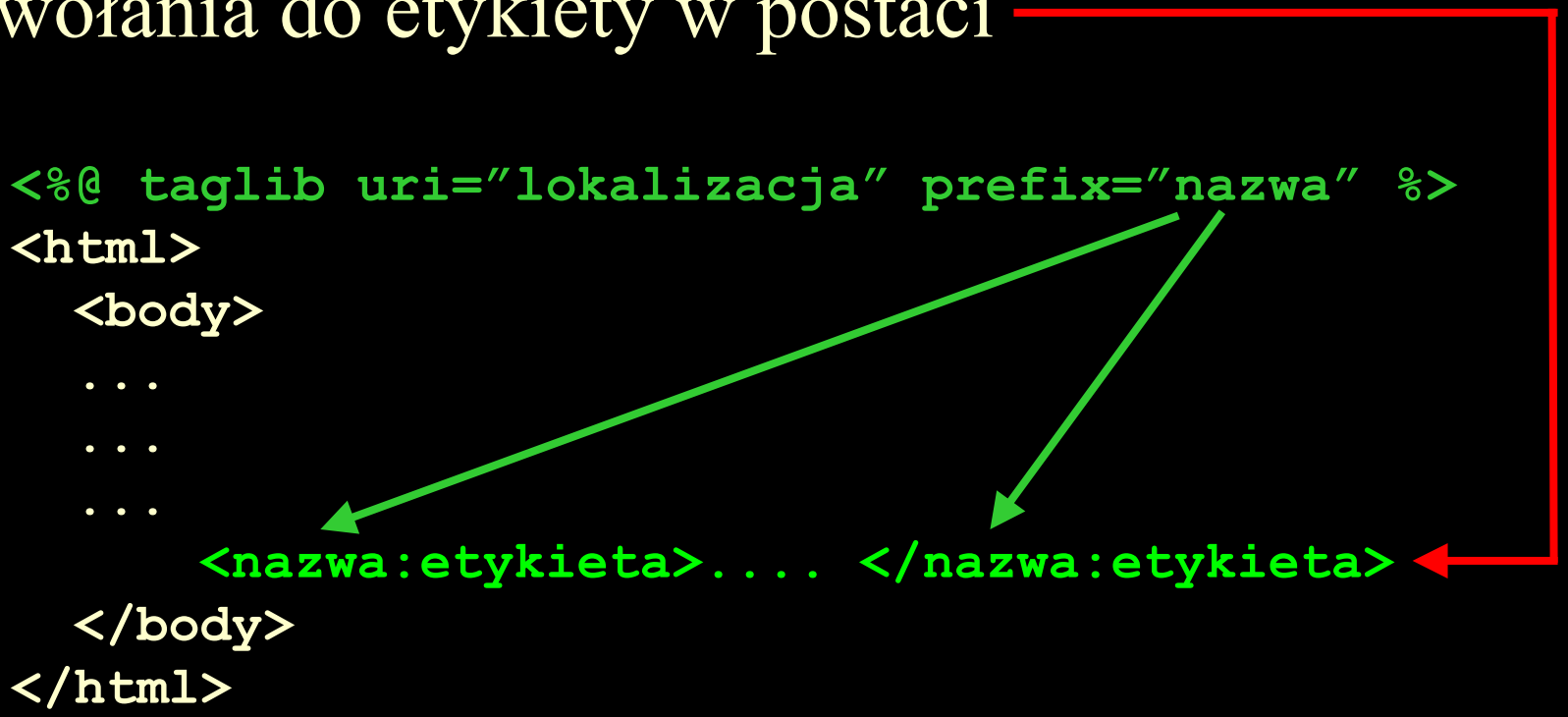
```
<%@ taglib uri="lokalizacja" prefix="nazwa" %>
```

uri - określenie lokalizacji deskryptora biblioteki, definiuje klasy, które zawierają kod obsługi etykiet

prefix - identyfikator klasy etykiety - nazwa za pomocą której można odwołać się do danej biblioteki (mogą istnieć dwie etykiety o takiej samej nazwie pochodzące z różnych bibliotek)

Wywołanie etykiety

Wywołanie etykiety realizuje się przez umieszczenie odwołania do etykiety w postaci



Upodobnienie technologii JSP do HTML

Wywołanie etykiety - przykład

Biblioteka TagLib (serwer)

portal.tld - Notatnik

Plik Edycja Format Pomoc

```
<?xml version="1.0" encoding="ISO-8859-1"
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.
        "http://java.sun.com/j2ee/dtds/we

<taglib>

  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>portal</shortname>
  <uri>hd1.ie.tu.koszalin.pl</uri>
  <info>Witryny i Portale Internetowe</in

  <tag>
    <name>data</name>
    <tagclass>portal.data</tagclass>
    <bodycontent>empty</bodycontent>
    <info>portal</info>
  </tag>
```

Strona JSP (serwer)

tag_data.jsp - Notatnik

Plik Edycja Format Pomoc

```
<%@ page contentType="text/html; charset=Windows-1250" %>
<%@ taglib uri="/WEB-inf/portal.tld" prefix="portal" %>
<html>
 <head>
  <title>Pobieranie bieżącej daty</title>
 </head>
 <body>
        Dzisiaj jest <portal:data/>
 </body>
</html>
```

Przeglądarka (klient)

Pobieranie bieżącej daty - Microsoft Internet Explorer

Plik Edycja Widok Ulubione Narzędzia Pomoc

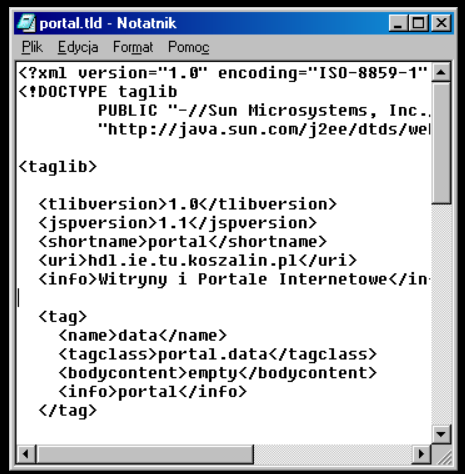
Wstecz Wyszukaj Ulubione

Adres http://localhost/portal/przyklady/tag_data.jsp Przejdź

Dzisiaj jest: 5 marzec 2002

Gotowe Lokalny intranet

Deskryptor biblioteki znaczników

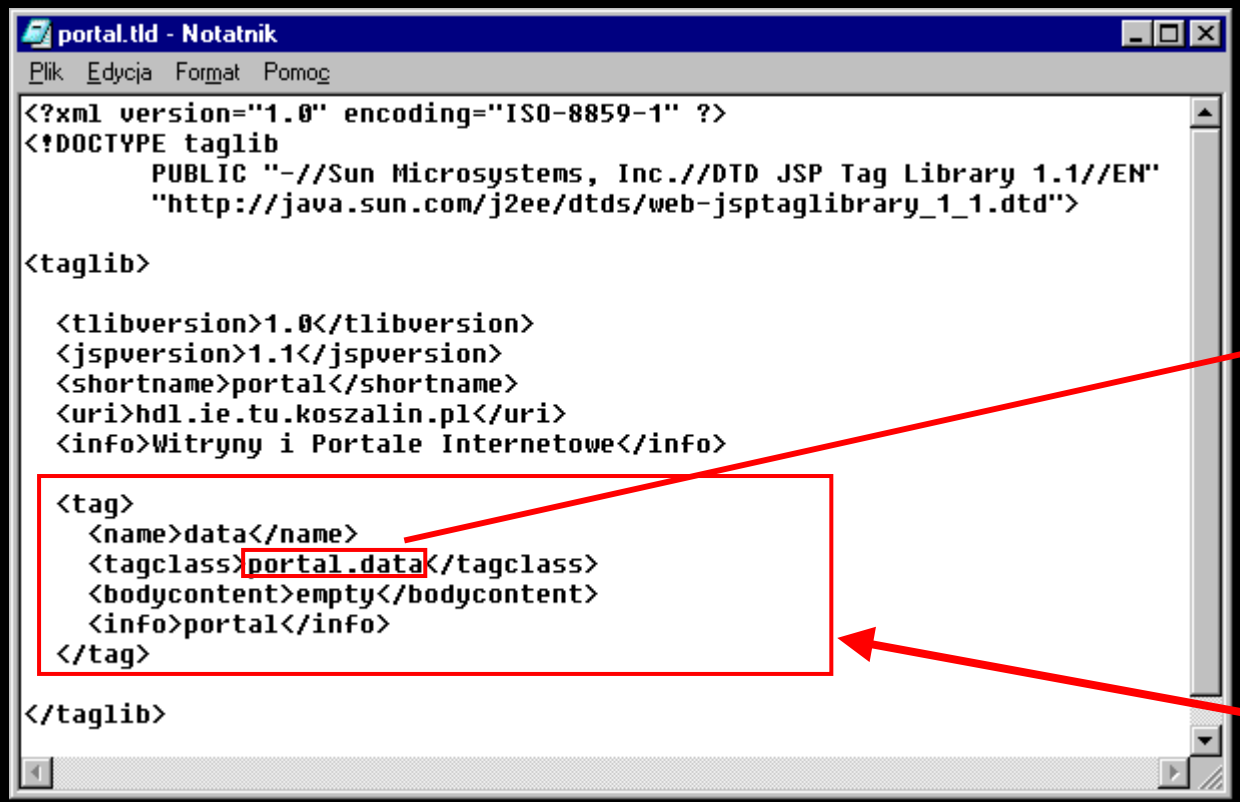


Deskryptor (TLD - Tag Library Descriptor) - definiowanie znaczników i ich połączeń z procedurami ich obsługi

Procedura obsługi - klasa Javy, która opisuje sposób postępowania po napotkaniu danego znacznika w kodzie JSP

Plik portal.tld

Plik TLD - deskryptor biblioteki znaczników



Klasa data.class pakietu portal

Deklaracja etykiety data znacznika prostego (bez ciała)

Plik TDL jest dokumentem XML

Klasa znacznika data

Klasa opisuje akcję, która będzie wykonana po wywołaniu znacznika

data.java - Notatnik

Plik Edycja Format Pomoc

```
package portal;

import java.text.DateFormat;
import java.util.Date;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class data extends TagSupport {

public int doStartTag() throws JspException {
        try {
                Date today = new Date();
                DateFormat formatter = DateFormat.getDateInstance(DateFormat.LONG);

                this.pageContext.getOut().write(formatter.format(today));
        } catch(java.io.IOException e) {
                throw new JspException("IO Error: " + e.getMessage());
        }
        return SKIP_BODY;
        }
}
```

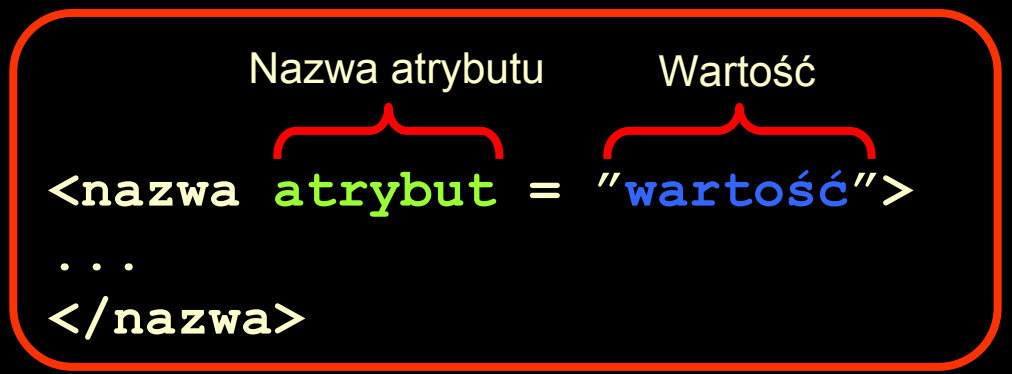
doStartTag - *funkcja wywoływana po napotkaniu początkowego znacznika* etykiety *data*

Biblioteki znaczników

Rodzaje znaczników

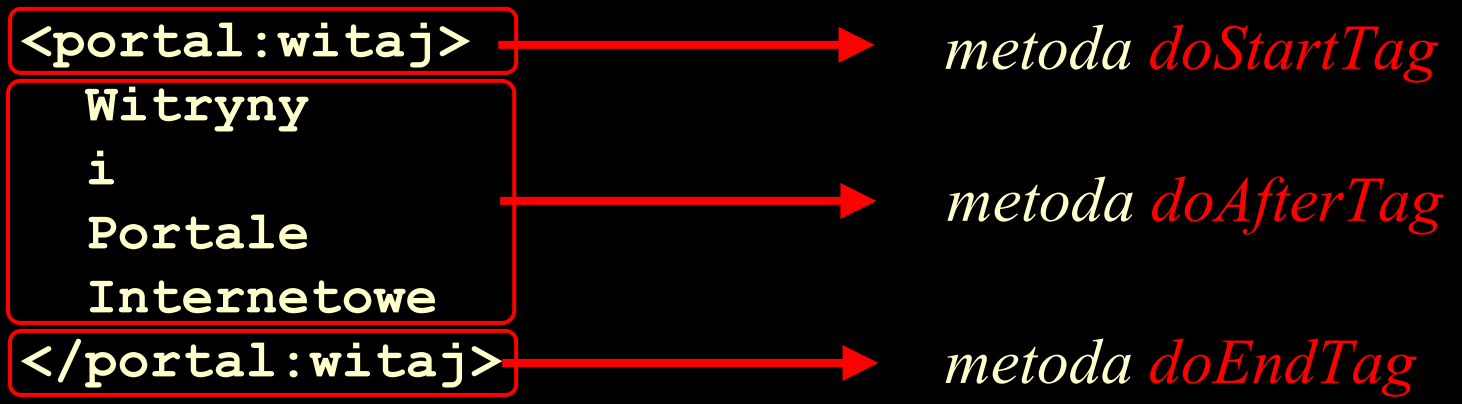
- proste (bez ciała - puste)
- proste z ciałem
- zagnieżdżone
- współpracujące
- iteracyjne

Każdy znacznik dodatkowo może posiadać własne atrybuty



Przypomnienie: składnia XML wymusza zamykanie wartości atrybutów w cudzysłowy.

Biblioteki znaczników




```
...
<portal:open_db
  url="..."
  login„..."
  pass„..." >
</nazwa:open_db>
```

otwarcie bazy danych

```
...
<portal:close_db/>
```

zamknięcie bazy danych

Znacznik prosty

<nazwa:witaj/>

```
public witaj extends TagSupport {

    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().print("Witaj!");
        } catch (Exception ex) {
            throw new JspTagException("SimpleTag: " + e.getMessage());
        }
        return SKIP_BODY;
    }

    public int doEndTag() {
        return EVAL_PAGE;
    }
}
```

Plik TLD

```
<tag>
    ...
    <bodycontent>empty</bodycontent>
</tag>
```

Znacznik z atrybutami

```
<nazwa:witaj atr="wartość"/>
```

```
public witaj extends Tag Support {

   private AttributeClass atr;

   setAtr(AttributeClass ac) {...}

   AttributeClass getAtr() {...}
   ...
}
```

Plik TLD

```
<tag>
   ...
   <attribute>
     <name>atr</name>
     <required>true|false|yes|no</required>
     <rtexprvalue>true|false|yes|no</rtexprvalue>
   </attribute>
</tag>
```

<rtexprvalue> - Request Time Value

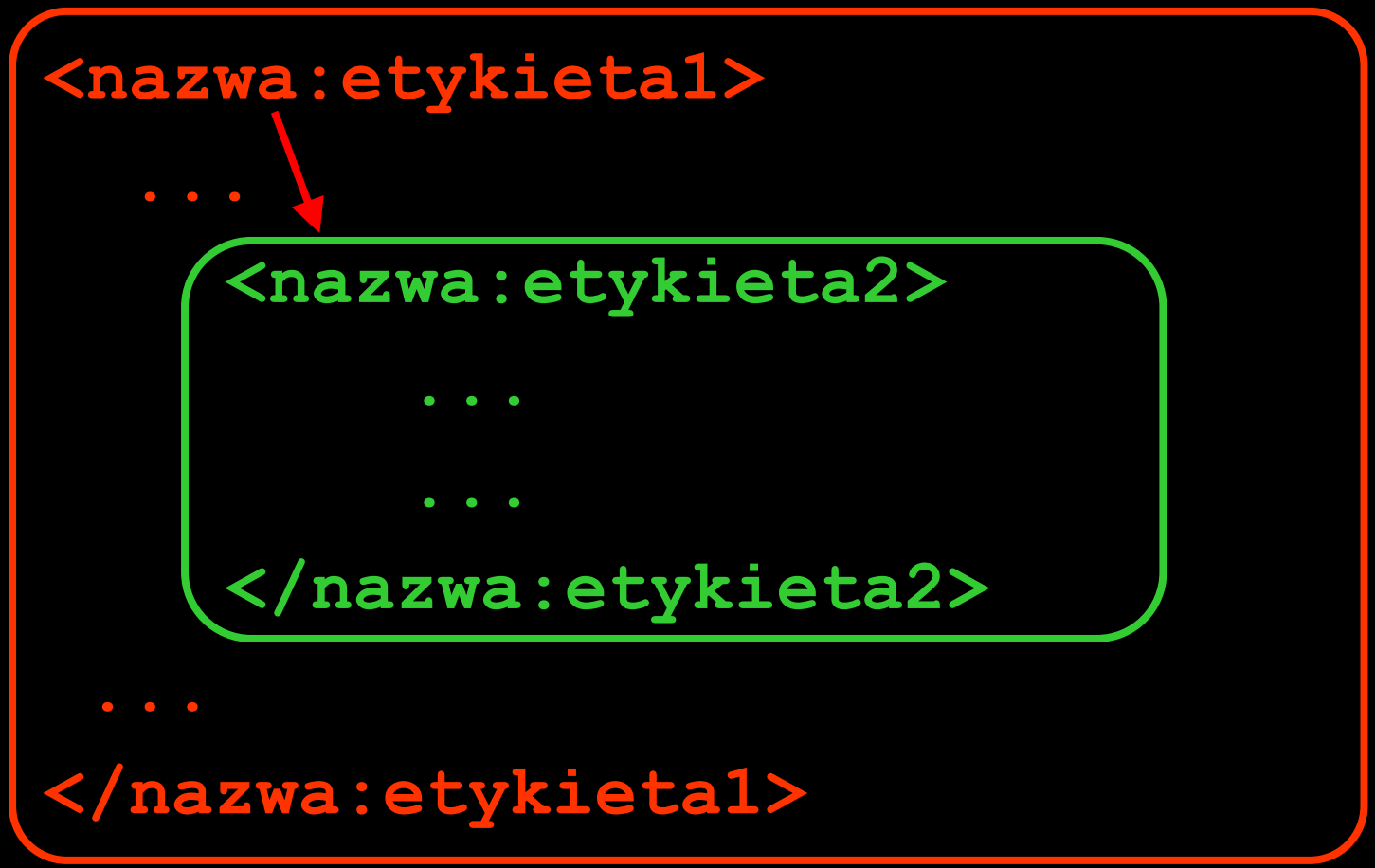
Znacznik prosty z ciałem


```
public witaj extends BodyTagSupport {

   ...
   public int doAfterBody() throws JspTagException {
     ...
     //własny kod...

     return SKIP_BODY;
   }
   ...
}
```

Plik TLD

```
<tag>
   ...
   <bodycontent>JSP|tagdependent</bodycontent>
</tag>
```

Znaczniki zagnieżdzone



Znaczniki współpracujące

Znacznik iteracyjny

```
<nazwa:etykieta>
...
</nazwa:etykieta>
```

Bezpieczeństwo

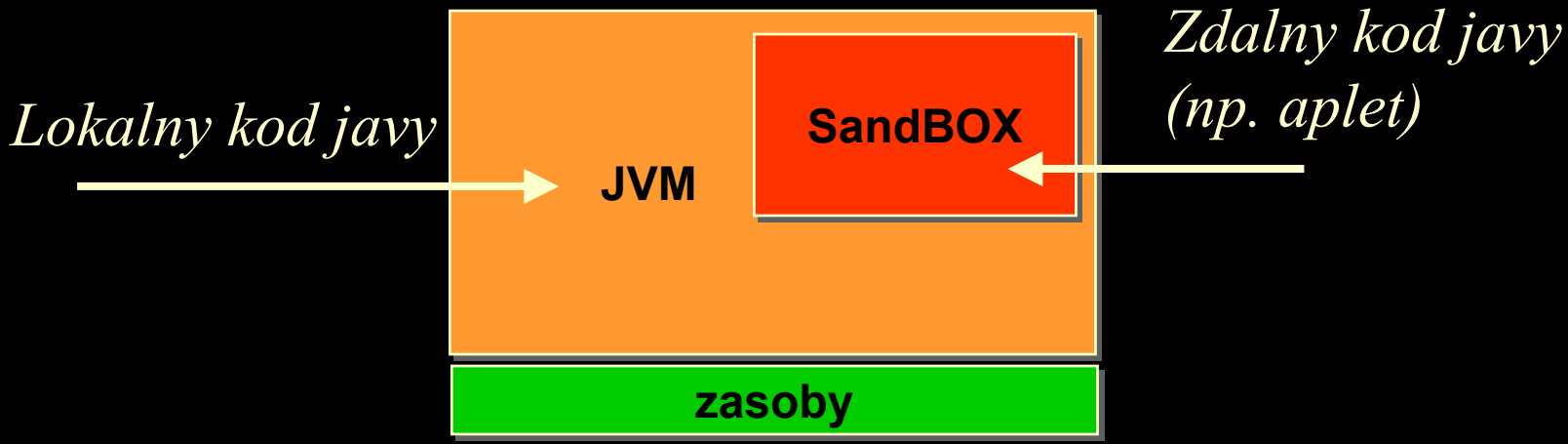
Bezpieczeństwo

Bezpieczeństwo – zabezpieczenie sprzętu, aplikacji i danych przed przypadkowym lub celowym wykorzystaniem, modyfikacją, uszkodzeniem, czy też dostępem do chronionej informacji

- autentykacja (autoryzacja)
- kontrola dostępu do zasobów
- integralność danych
- poufność i prywatność danych

Implementacja zabezpieczeń

SandBox – specjalne środowisko uruchomieniowe do umieszczania kodu z nieznanego źródła



Weryfikacja poprawności kodu - ładowarka klas sprawdza np.. czy dany kod nie łamie praw dostępu

Weryfikacja użytkowników

Sposoby weryfikacji użytkowników

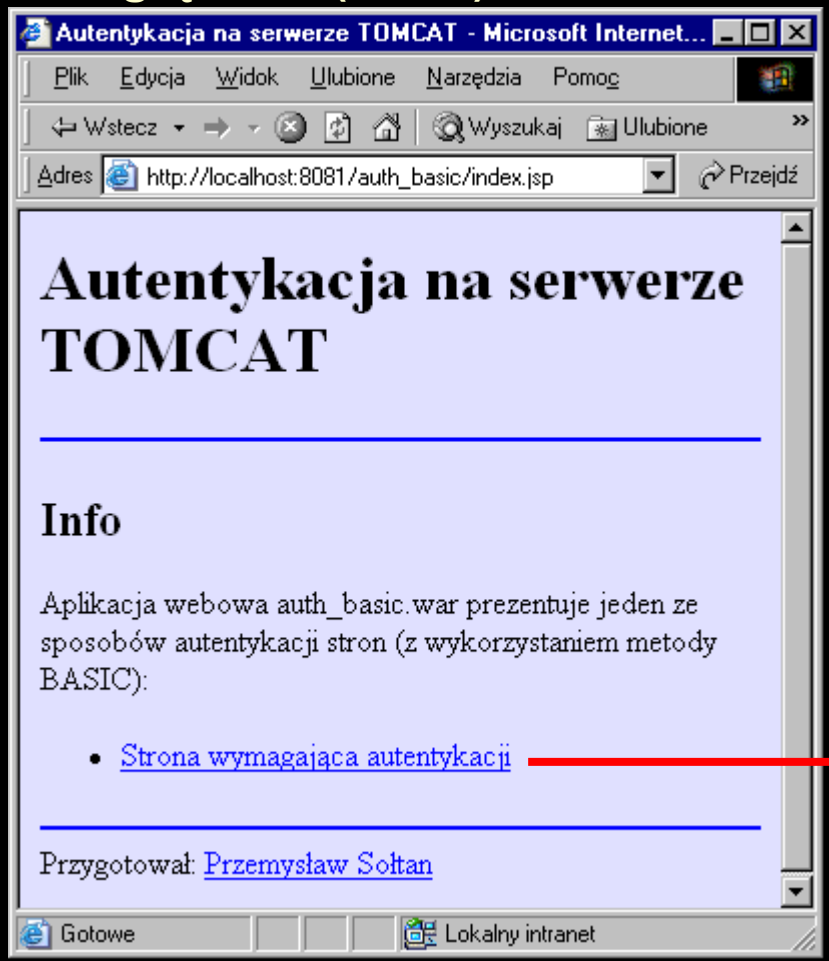
- identyfikator (login) i hasło
- adres IP
- adres DNS
- certyfikat X.590v3
- systemy tokenów
- JavaCart/SmartCart
- inne mechanizmy uwierzytelniania

Sposoby autentykacji

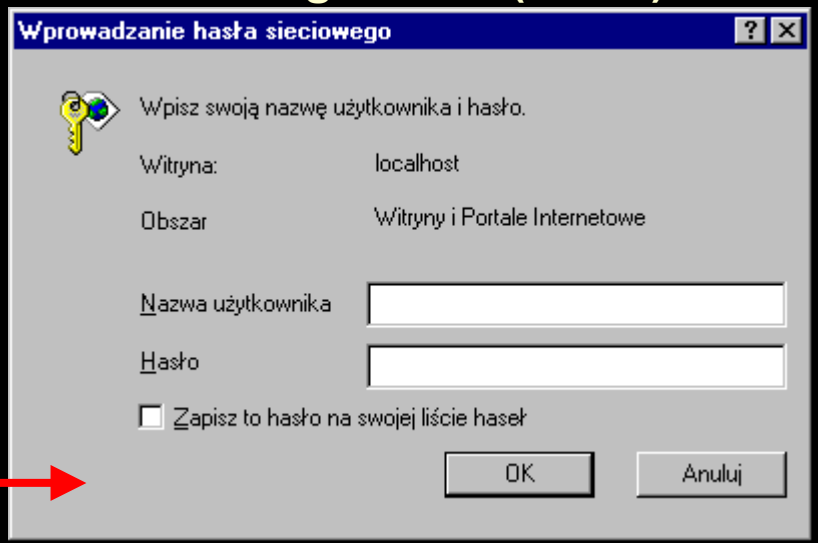
- za pomocą metody BASIC (obsługa wprowadzania loginu i hasła spoczywa na przeglądarce klienta)
- za pomocą metody FORM (obsługa wprowadzania loginu i hasła za pomocą formularza strony html)
- z wykorzystaniem certyfikatów
- własne metody oparte o logowanie i bazy danych

Autentykacja - BASIC

Przeglądarka (klient)

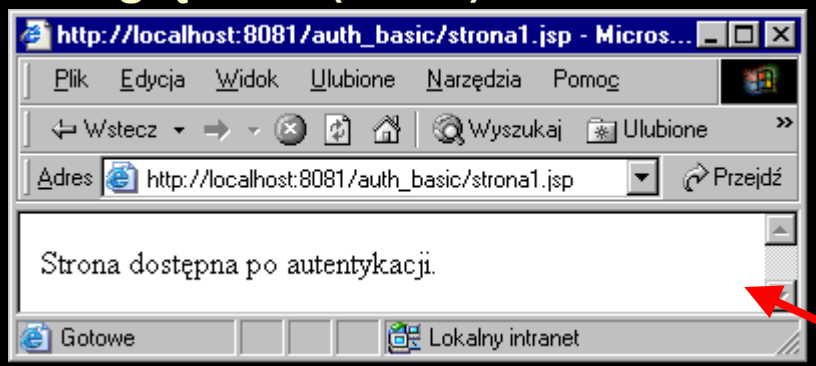


Formularz logowania (klient)

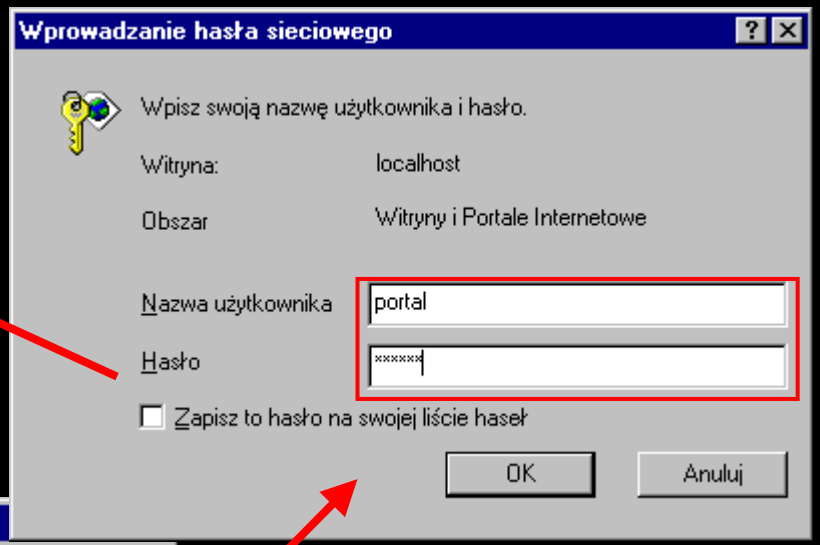


Autentykacja - BASIC

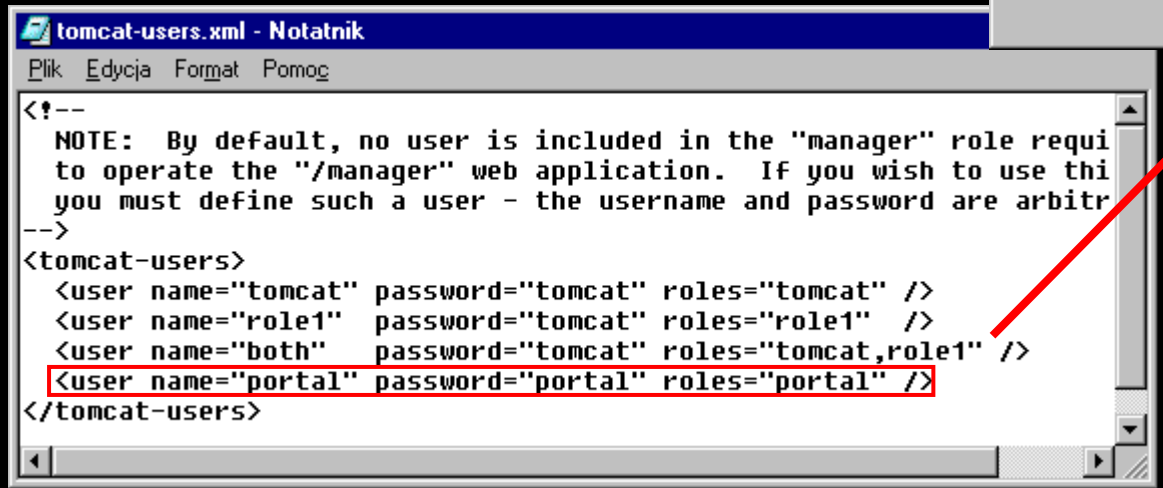
Przeglądarka (klient)



Formularz logowania (klient)



Plik tomcat-users.xml (Tomcat)

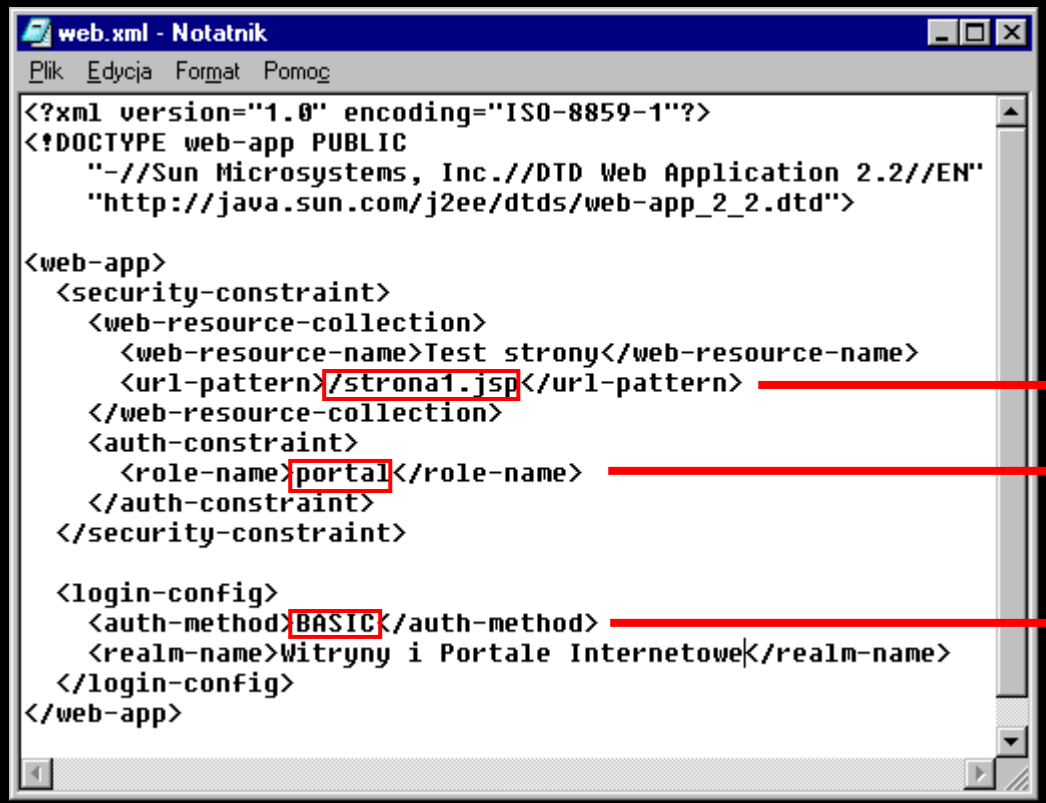


Loginy i hasła autentykacji są przechowywane na serwerze tomcat w pliku

tomcat-users.xml

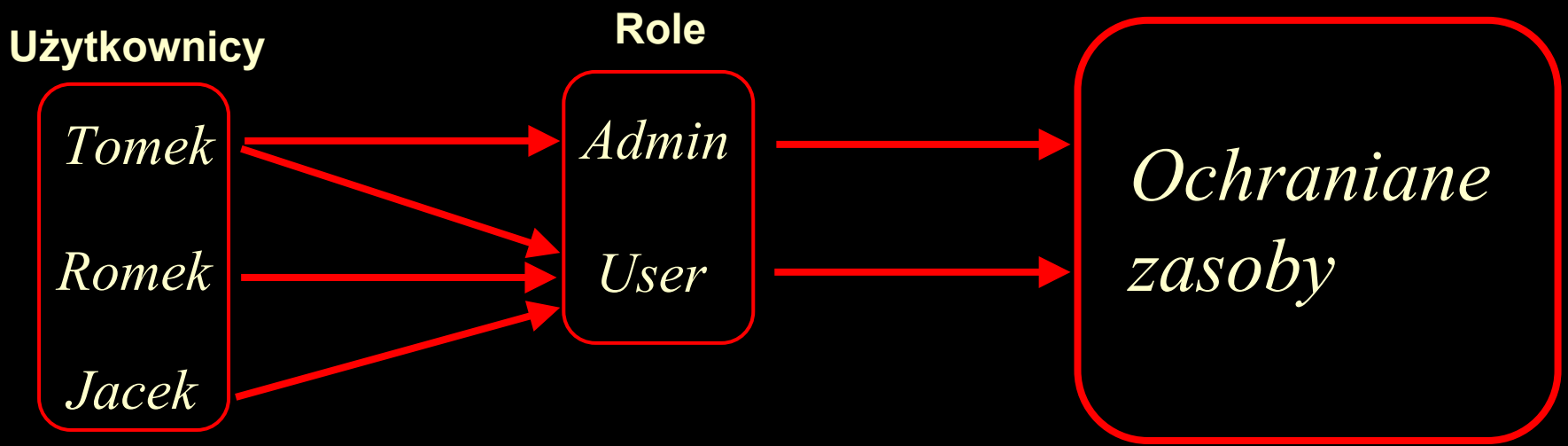
Autentykacja - BASIC

Plik web.xml (Tomcat)



Role

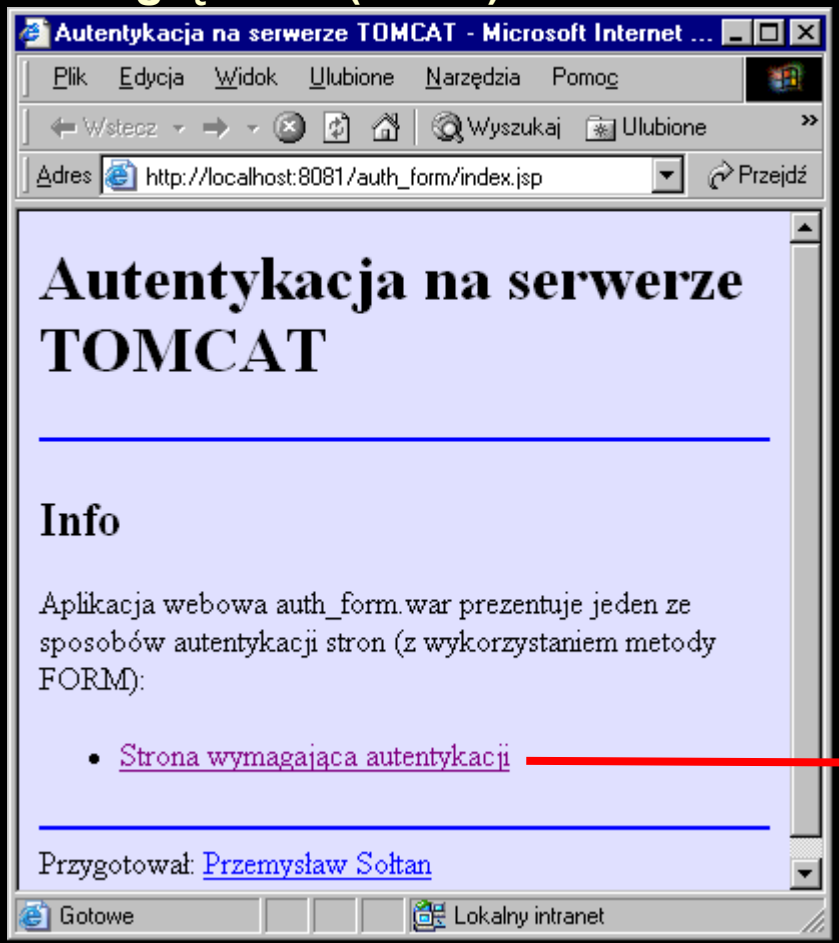
ROLE - sposób na nadawanie uprawnień dla grupy użytkowników



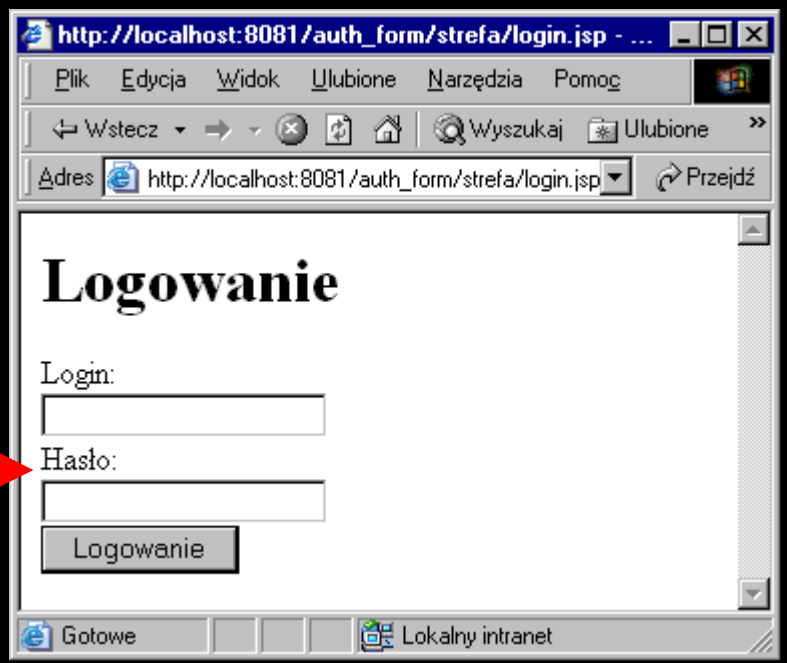
Użytkownik Tomek może przeglądać zasoby przeznaczone dla Admin i User

Autentykacja - FORM

Przeglądarka (klient)



Przeglądarka (klient)



Autentykacja - FORM

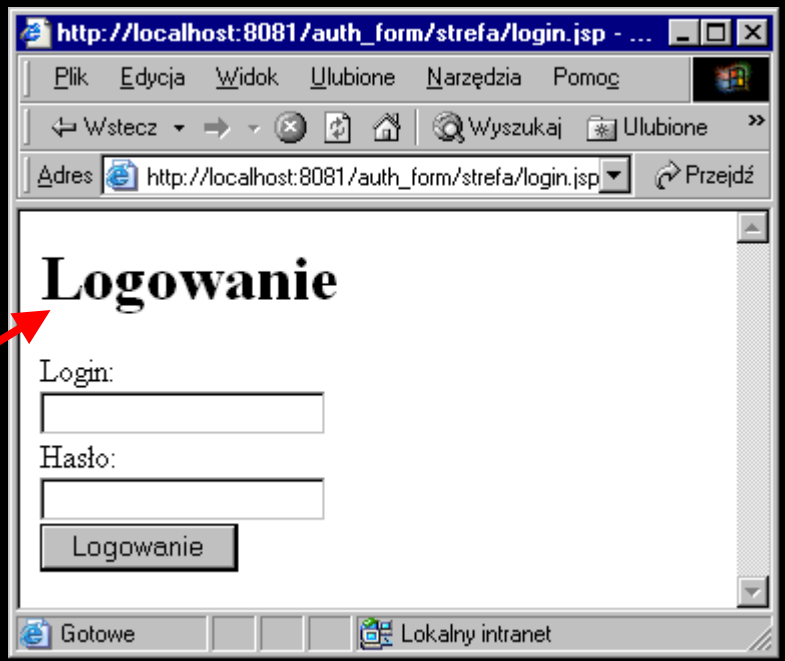
Strona jsp (Serwer)

login.jsp - Notatnik

Plik Edycja Format Pomoc

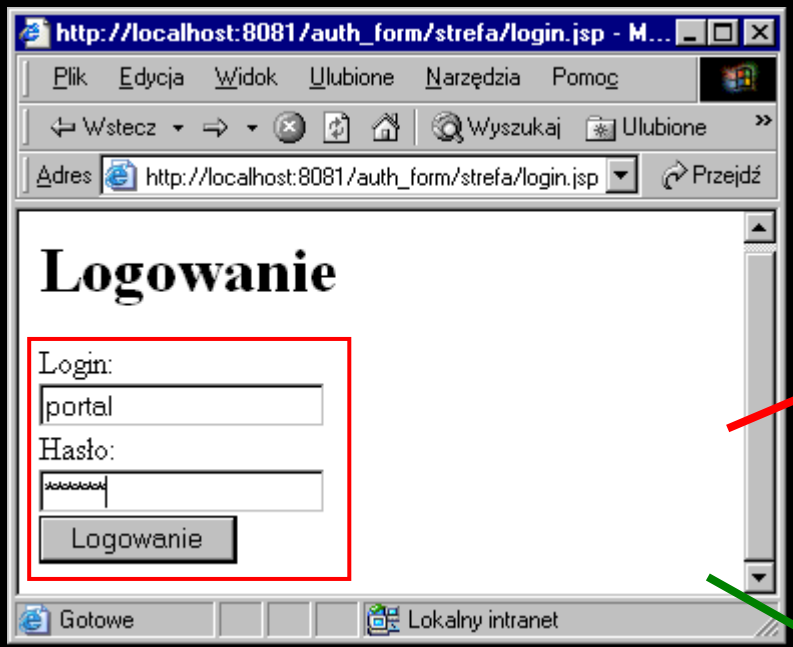
```
<%@ page contentType="text/html; charset=windows-1250" %>
<html>
  <head>
  </head>
  <body>
   <h1>Logowanie</h1>

   <form action="j_security_check" pethod="POST">
     Login:
     <br>
     <input type="text" name="j_username">
     <br>
     Hasło:
     <br>
     <input type="PASSWORD" name="j_password">
     <br>
     <input type="SUBMIT" VALUE="Logowanie">
   </form>

  </body>
</html>
```

Przeglądarka (klient)

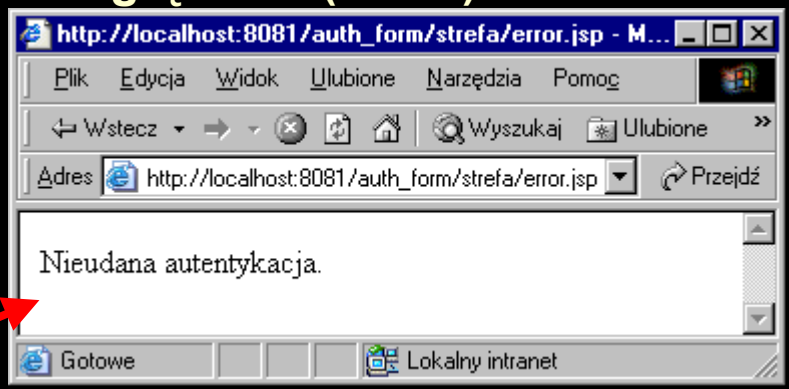


Autentykacja - FORM

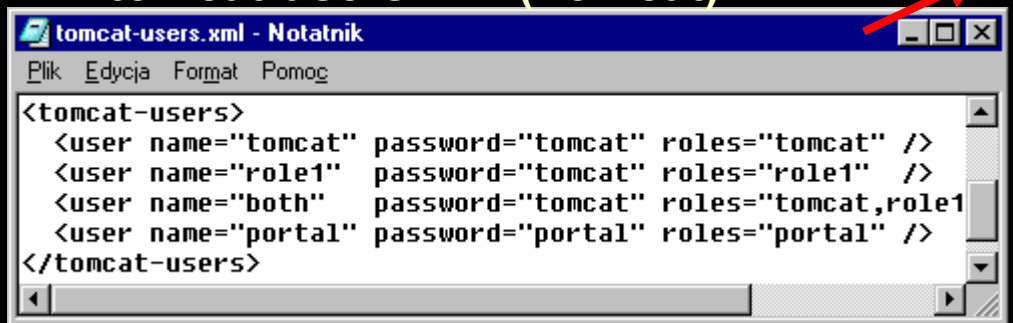
Przeglądarka (klient)

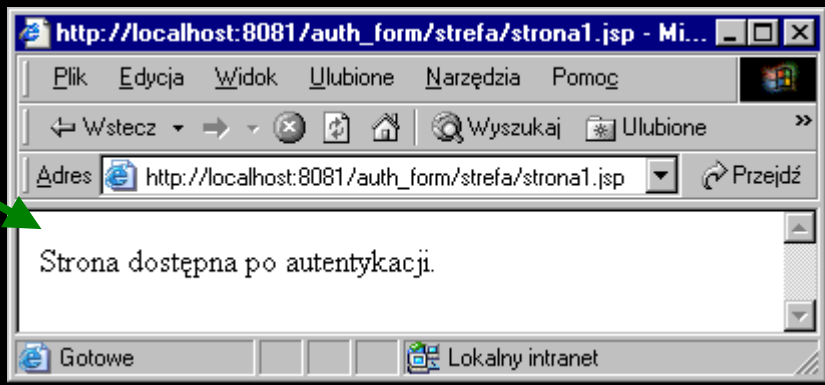


Przeglądarka (klient)



Plik tomcat-users.xml (Tomcat)





Przeglądarka (klient)

Autentykacja - FORM

Lister - [D:\java\Tomcat\webapps\auth_form\WEB-INF\web.xml]

File Edit Options Help 100 %

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC
    "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/strefa/*</url-pattern>
      <http-method>DELETE</http-method>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <http-method>PUT</http-method>
    </web-resource-collection>

    <auth-constraint>
      <role-name>portal</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>Obszar autentykacji</realm-name>
    <form-login-config>
      <form-login-page>/strefa/login.jsp</form-login-page>
      <form-error-page>/strefa/error.jsp</form-error-page>
    </form-login-config>
  </login-config>

</web-app>
```

Plik web.xml (Tomcat)

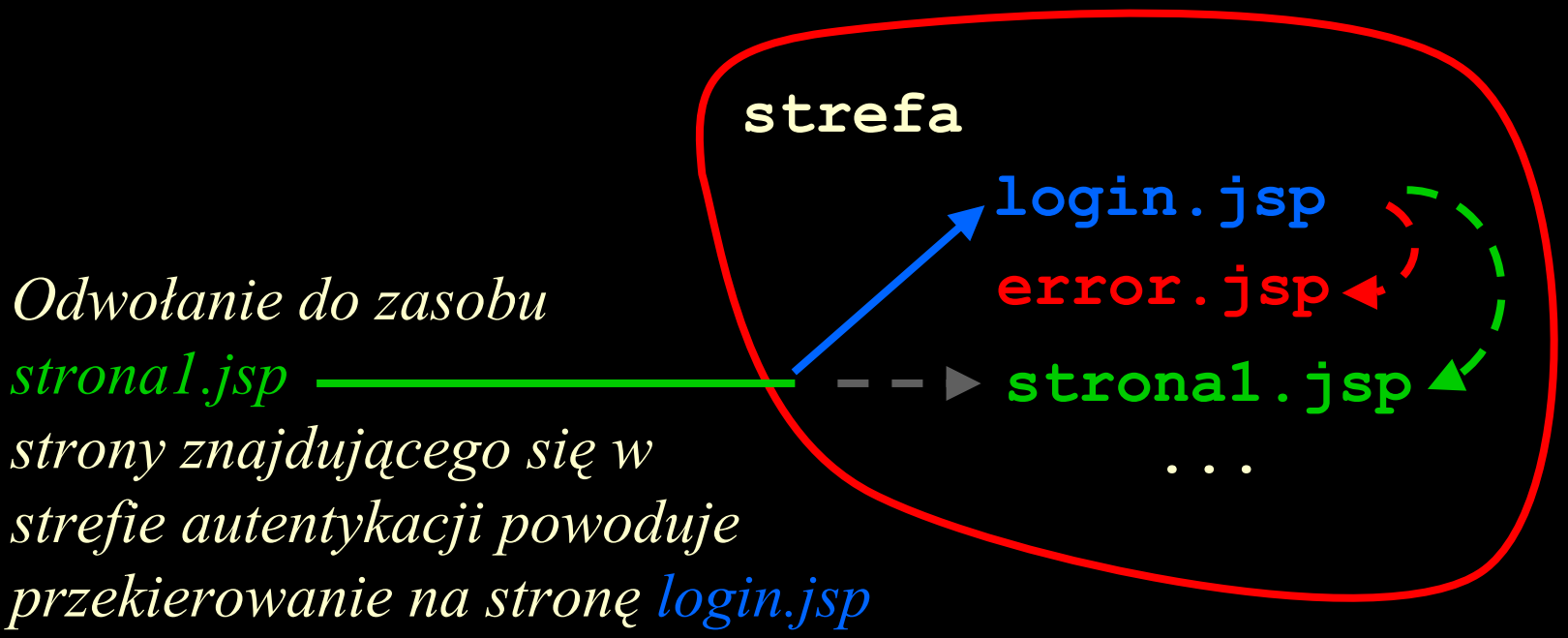
Strefa chronionych zasobów – katalog strefa

Nazwa roli (grupy użytkowników portal)

Metoda autentykacji (FORM)

Lokalizacja strony logowania i obsługi błędnej autentykacji

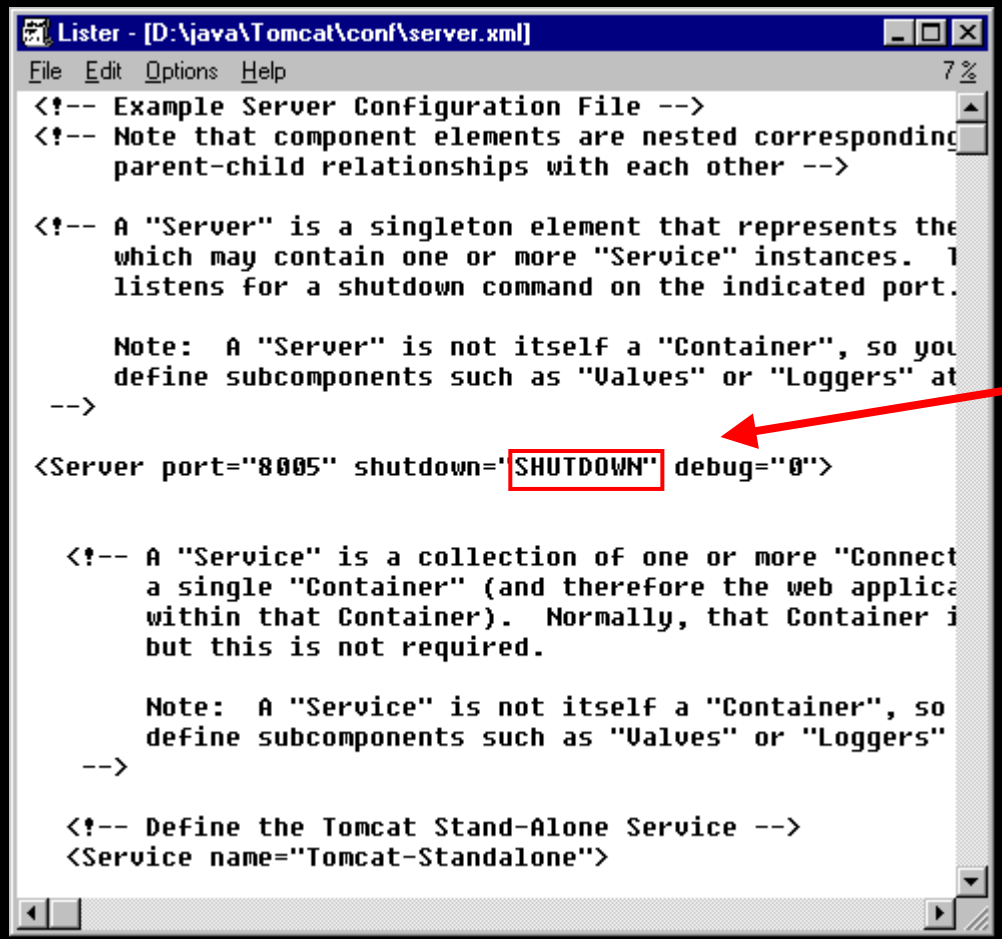
Autentykacja - FORM



W przypadku błędnej autentykacji następuje przekierowanie do strony obsługi błędu error.jsp

TOMCAT - shutdown

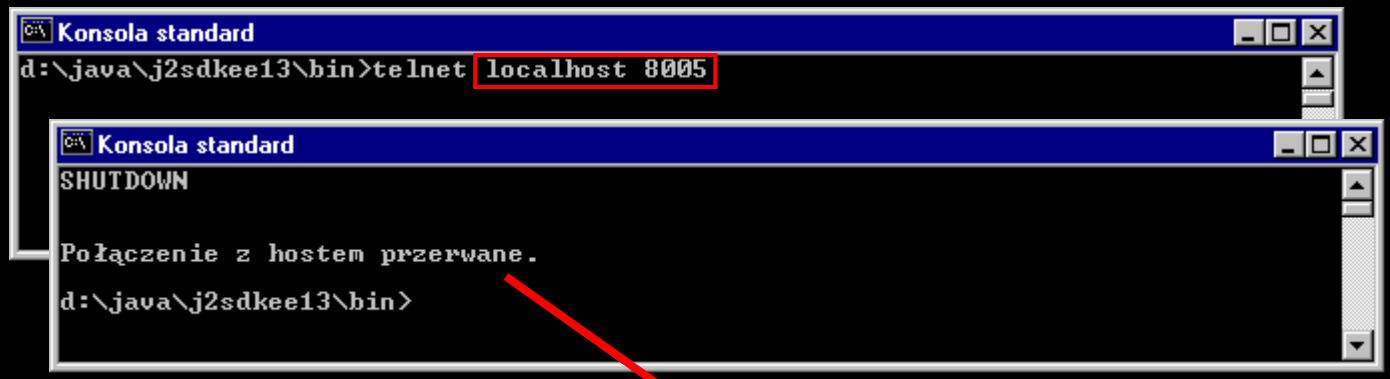
Plik server.xml (Tomcat)



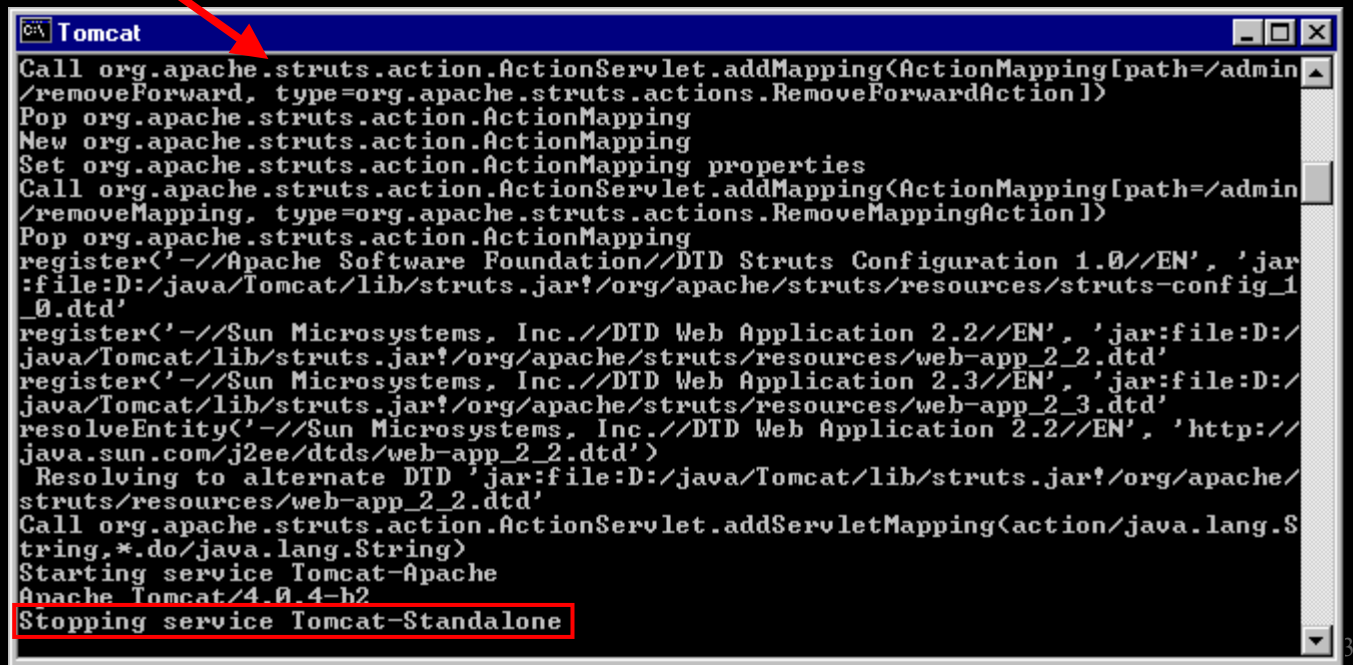
Dla standardowych ustawień servera Tomcat należy zmienić hasło „SHUTDOWN" zdalnego wyłączenia serwera

TOMCAT - shutdown

Telnet



Tomcat



Zdalne wyłączenie Serwera Tomcat

Security Manager

Security Manager

Security Manager – zarządca bezpieczeństwa kontrolujący uruchamiane programy Javy

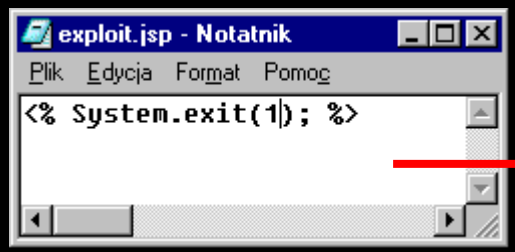
```
-Djava.security.manager
```

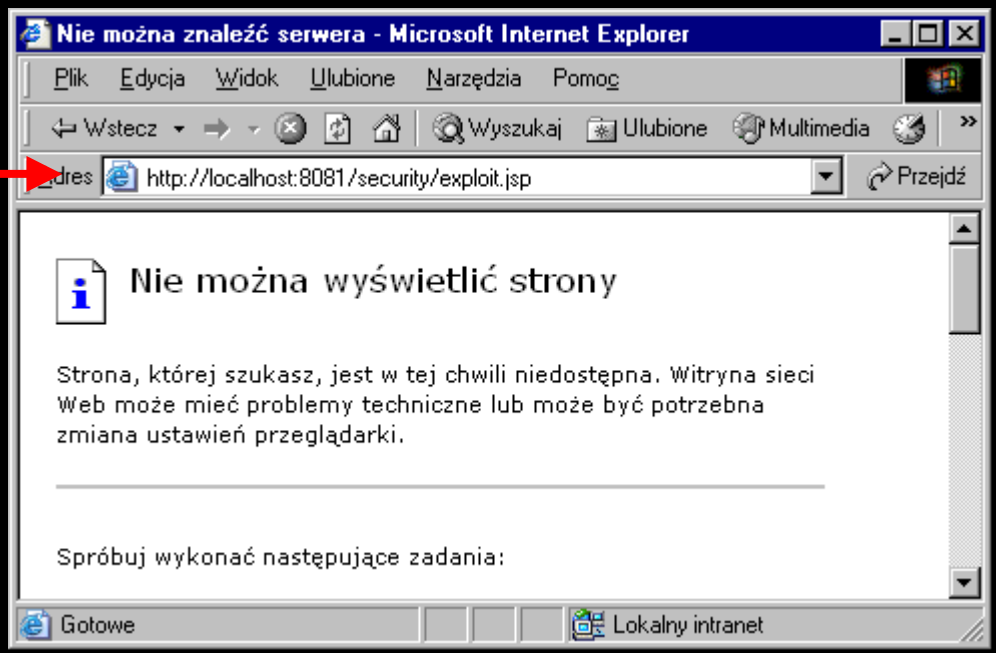
- monitorowanie dostępu do zasobów (własny zarządca bezpieczeństwa)
- stosowanie niestandardowych metod autoryzacji do zasobów (pełen audyt do zasobów)

Security Manager

Standardowe ustawienia serwera TOMCAT umożliwiają wywołanie polecenia zamykającego system

```
<% System.exit(1); %>
```



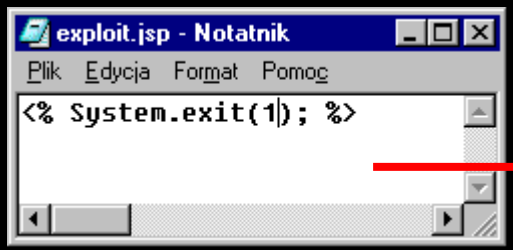


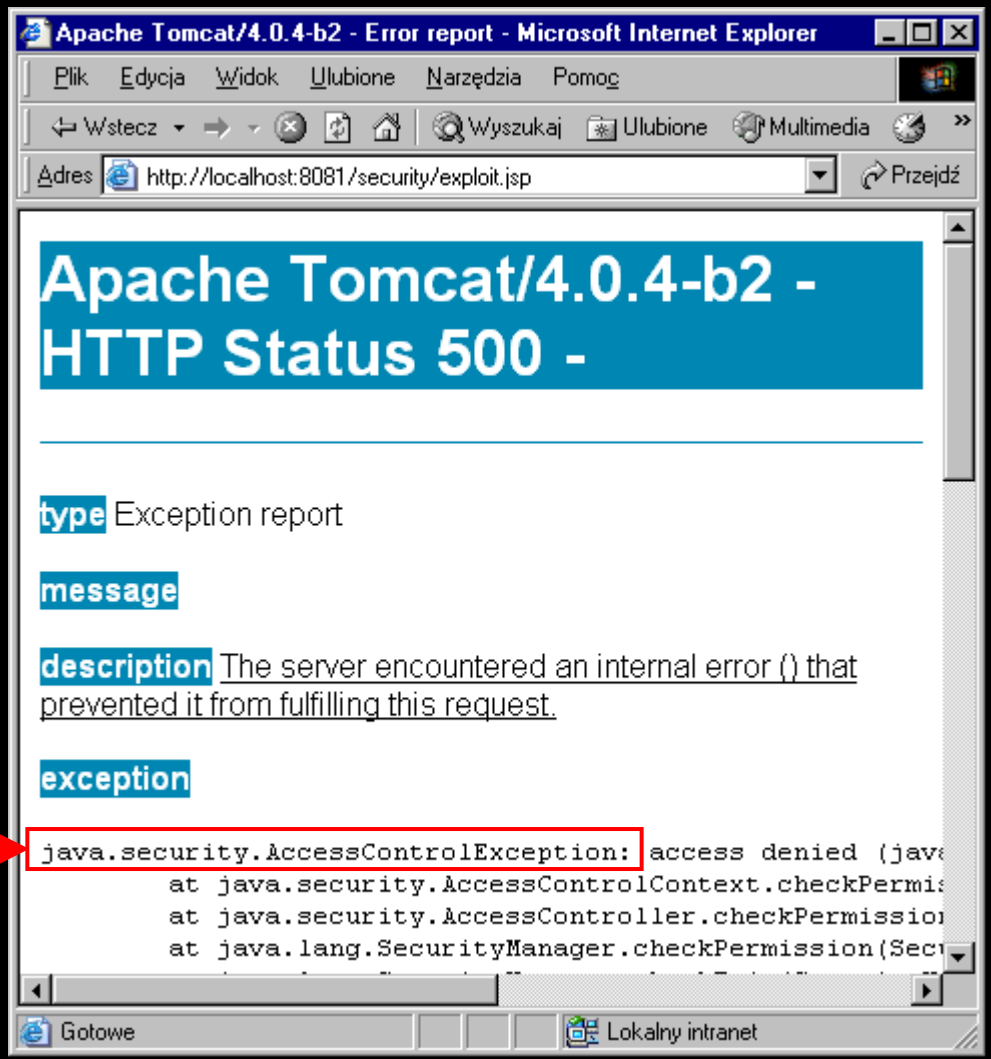
Wywołanie strony jsp z podanym kodem powoduje zamknięcie aplikacji javy jakim jest serwer Tomcat

Security Manager

Server Tomcat
należy uruchomić
w trybie ochrony

```
startup.bat -security
```





Secrity Manager
kontroluje kod javy

Program policytool

Standardowy program policytool (w JDK) służy do tworzenia i modyfikacji plików definiujcych przywileje

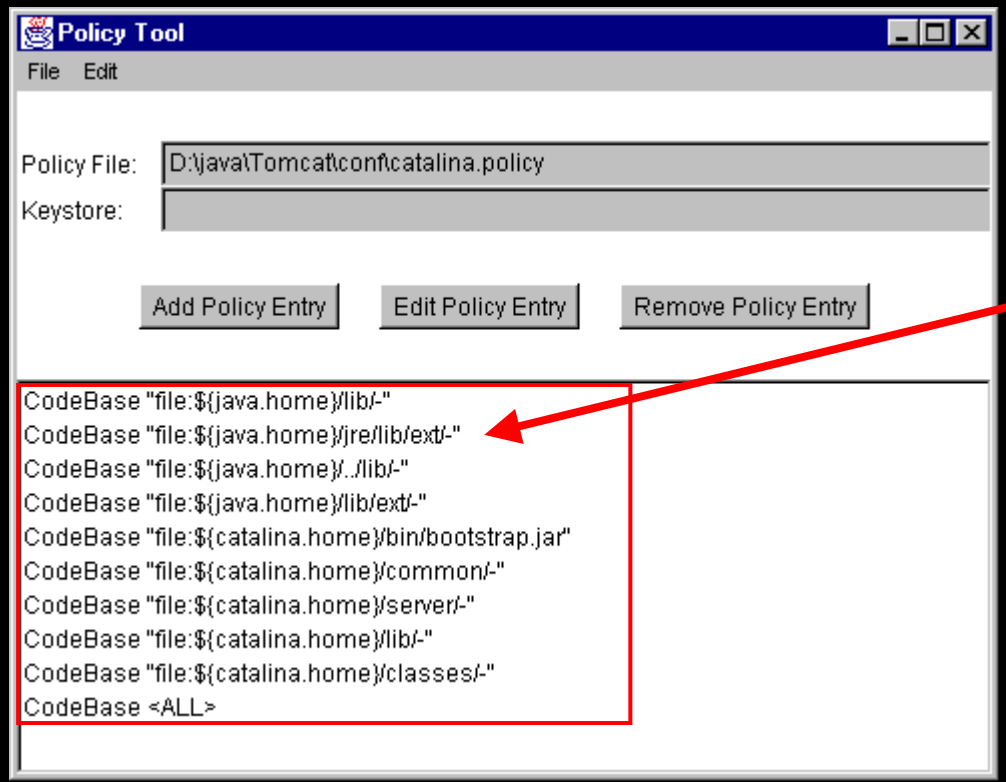
```
-Djava.security.policy=mój_plik
```

W ten sposób można:

- nadawać prawa dostępu do zasobów
- uzyskiwać informacje o użytkowniku
- uruchamiać inne programy
- korzystanie z portów zewnętrznych, itd....

Program policytool

Plik catalina.policy (policytool.exe)



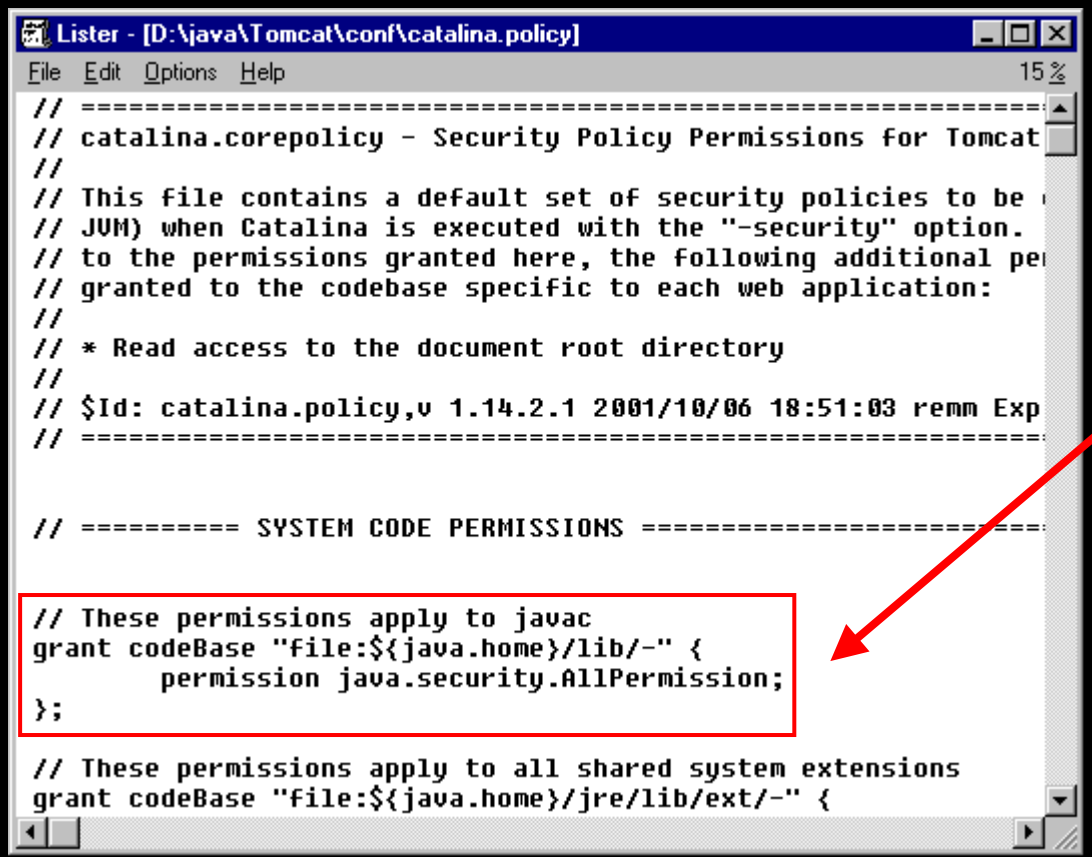
Biblioteki umieszczone w katalogu lib/ext JVM posiadają wszystkie uprawnienia

Plik catalina.policy jest odpowiednio mapowany przy starcie serwera Tomcat

```
set SECURITY_POLICY_FILE=%CATALINA_BASE%\conf\catalina.policy
```

catalina.policy

```
grant [signedBy <signer> [,codeBase <code source>] {
        permission <class> [<name> [, <action list>]];
};
```



Lista uprawnień do odpowiednich zasobów serwera Tomcat

Plik catalina.policy (Tomcat)

Certyfikaty

Certyfikaty

Certyfikat (Certificate)

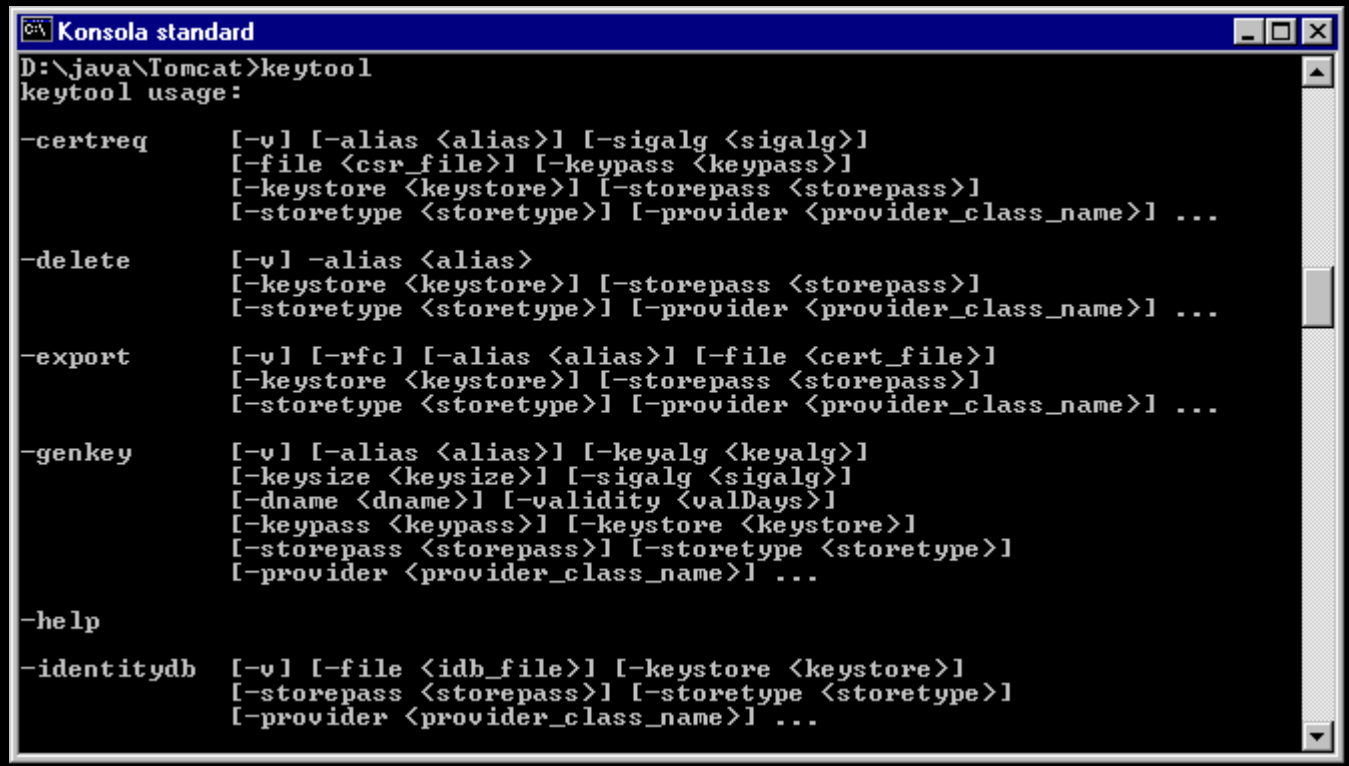
- standaryzowany dokument zawierający klucz publiczny nadawcy sygnowany podpisem elektronicznym firmy trzeciej (certificate authority)

Certificate Authority (CA)

- firma ciesząca się zaufaniem, której klucz publiczny jest powszechnie znany (np. VeriSign)

Program keytool

Program keytool służy do zarządzania bazą certyfikatów

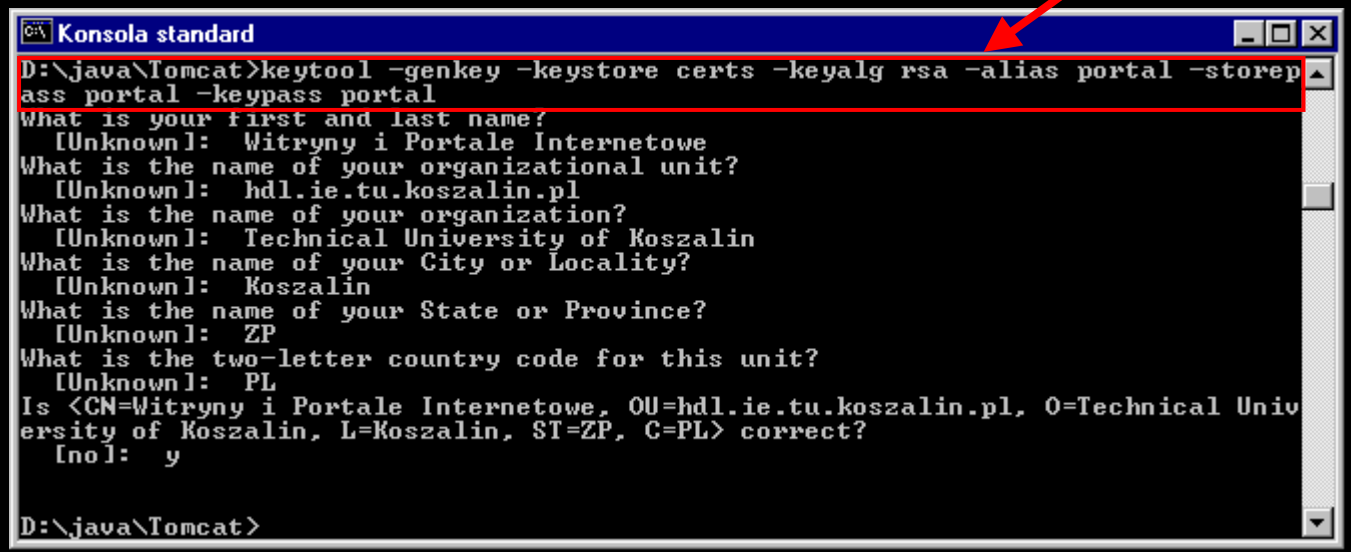


Standardowy plik cacerts w katalogu /jre/lib/security JDK zawiera przykładowe certyfikaty VeriSign

Certification Authority (CA)

Tworzenie certyfikatu serwera

```
keytool -genkey -keystore certs -keyalg rsa -alias portal
-storepass portal -keypass portal
```

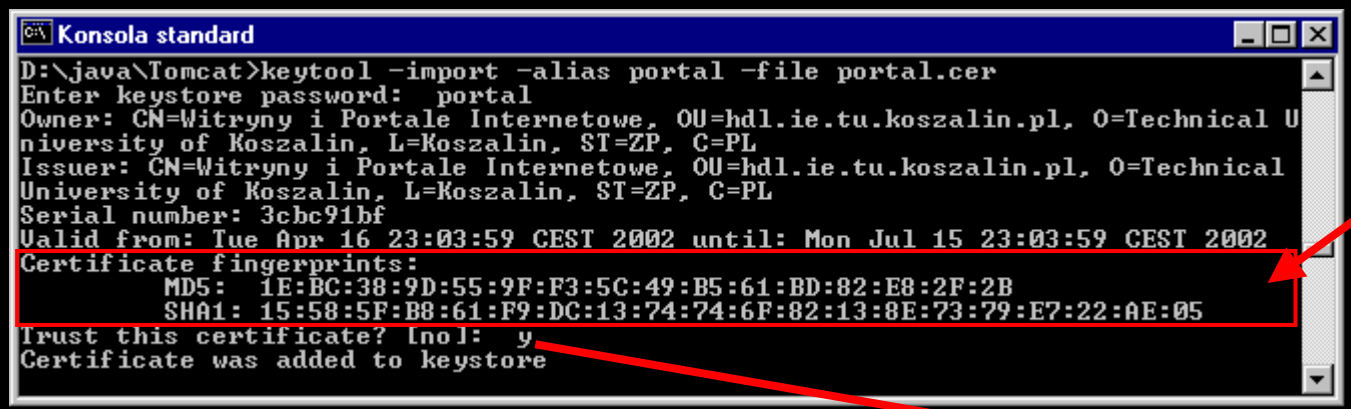


Po wprowadzeniu odpowiednich danych w katalogu bieżącym (Tomcat) powinien pojawić się plik certyfikatu - certs

Eksport i import certyfikatów

```
keytool -export -keystore certs -alias portal -file portal.cer
```


```
keytool -import -alias portal -file portal.cer
```

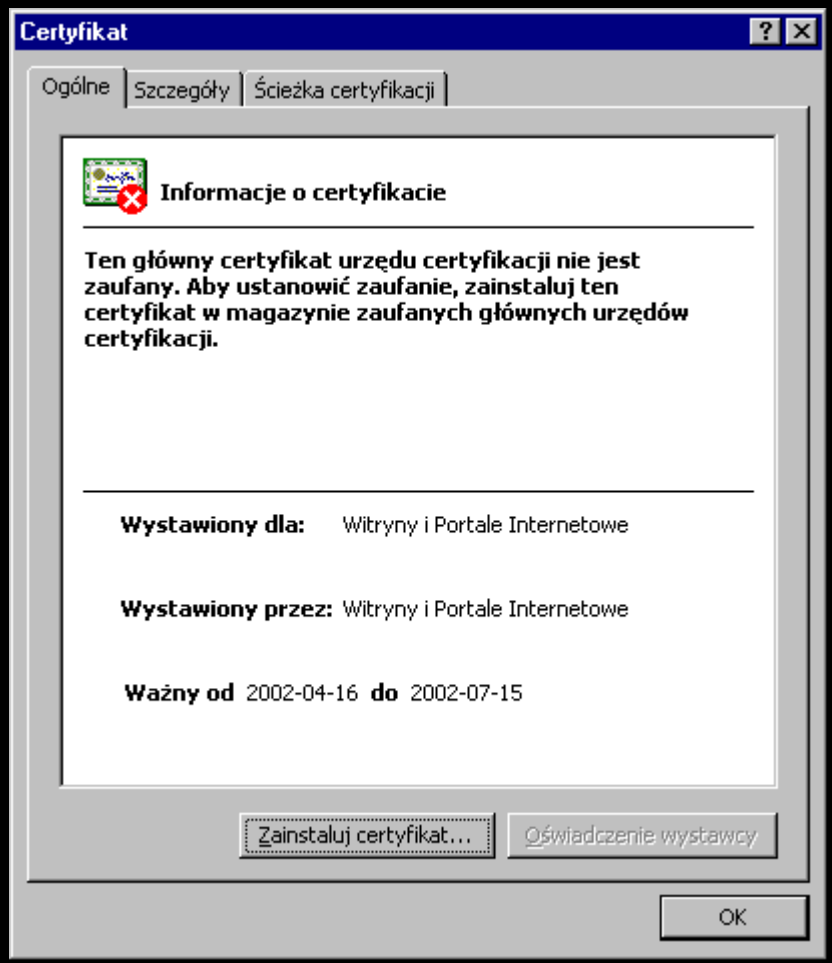


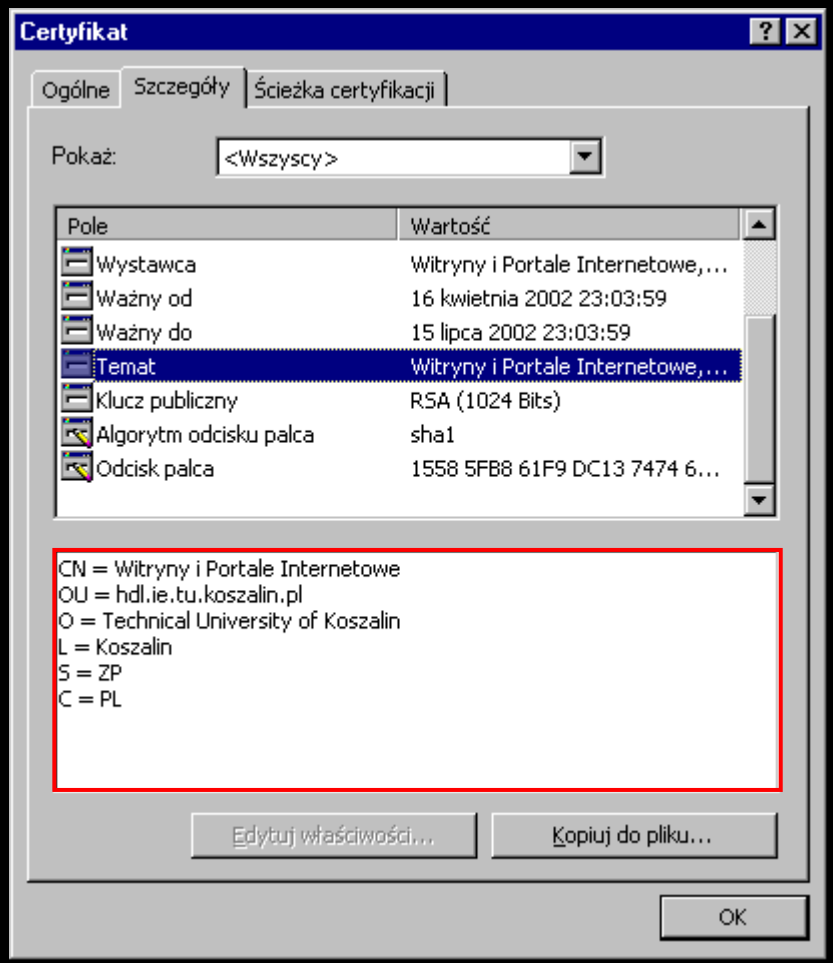
Odcisk certyfikatu

Po zatwierdzeniu zostanie stworzony plik .keystore który przenosi się do katalogu /lib/security javy

Certyfikat portal.cer

```
keytool -export -keystore certs -alias portal -file portal.cer
```





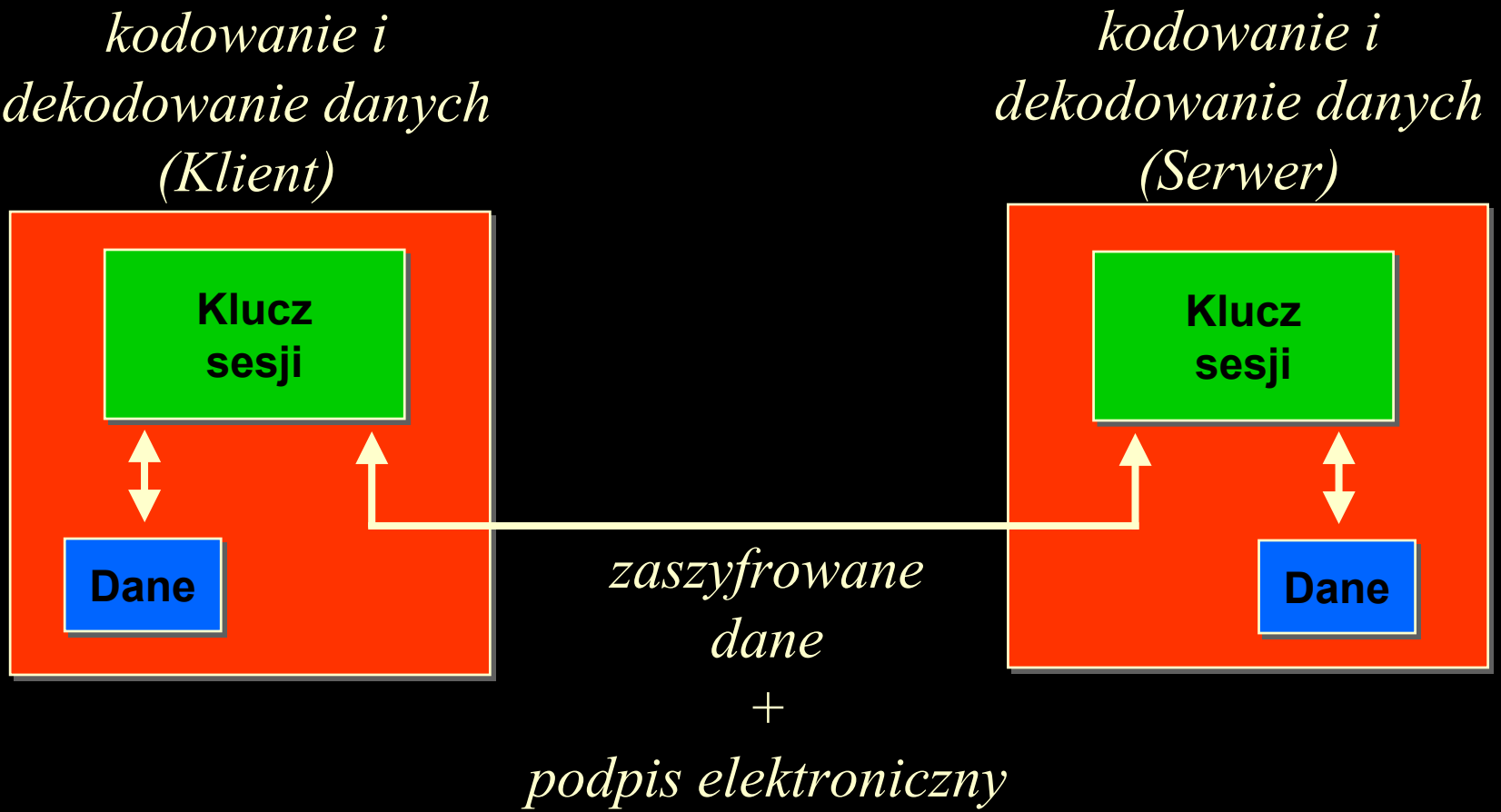
SSL i HTTPS

Security Socket Layer (SSL)

SSL służy do nawiązania bezpiecznego połączenia serwera i klienta

Podczas połączenia następuje uzgodnienie klucza sesyjnego (zmiennego w czasie) i szyfrowanie całej transmisji algorytmem symetrycznym z wykorzystaniem klucza (jedynie uzgodnienie klucza odbywa się jawnie)

Szyfrowanie synchroniczne



SSL

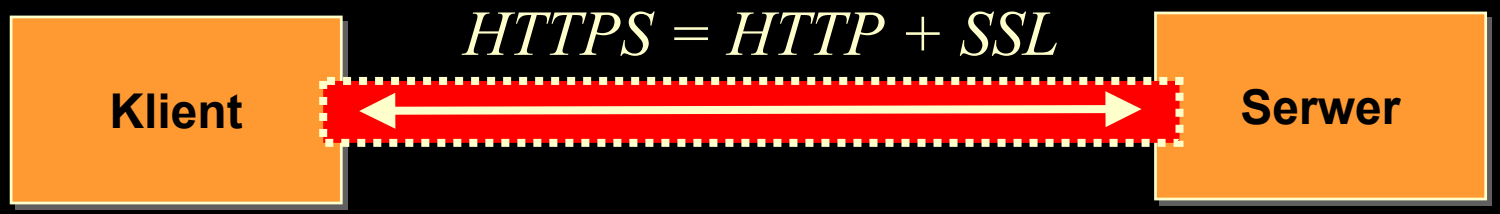
Wykorzystanie SSL:

- niejawne wsparcie w przeglądarce klienta protokołu SSL
- pominięcie wsparcia przeglądarki i wykorzystanie pakietu JSSE (biblioteka Java Secure Socket Extension)

Securite Hyper Text Transmission Protocol (S-HTTP)

Tunelowanie danych w protokole HTTPS

- tunelowanie - transmisja danych jednego rodzaju protokołu poprzez „opakowanie” ich w inny protokół



Transmisja przez domyślny port 443 lub 8443.

Java Secure Socket Extension JSSE

Instalacja JSSE 1.0.2 :

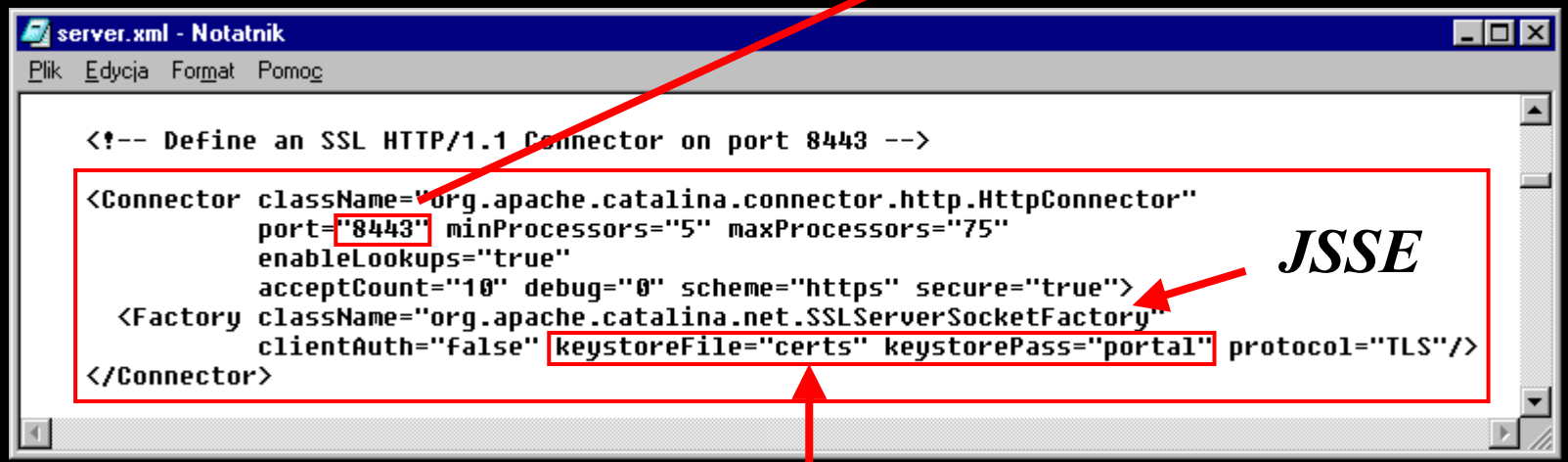
- umieszczenie plików jsse.jar, jnet.jar i jcert.jar w katalogu /jre/lib/ext JDK javy
- ustawienie zmiennej środowiskowej wskazującej na katalog zawierający biblioteki JSSE

```
set JSSE_HOME = %JAVA_HOME%/jre/lib/ext
```

W JDK 1.4 biblioteka JSSE jest już integralną częścią JDK

Konfiguracja SSL (Tomcat)

https ://localhost: **8443**

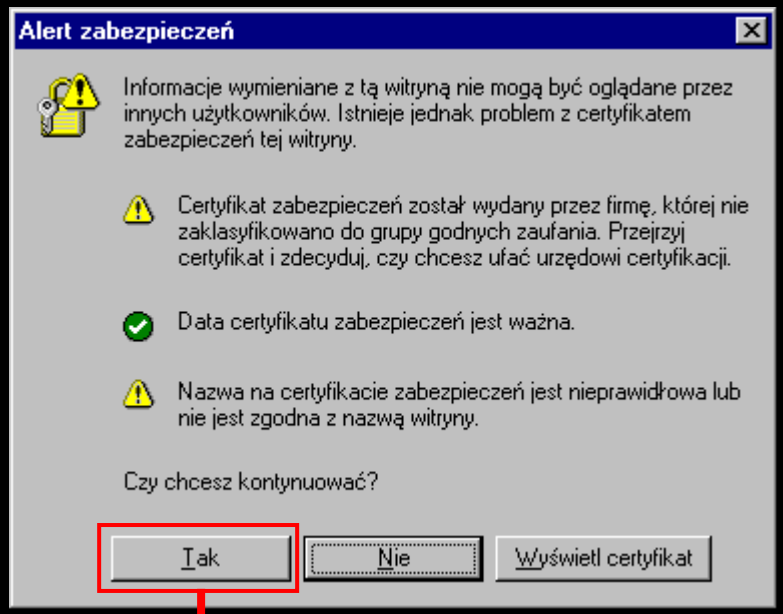


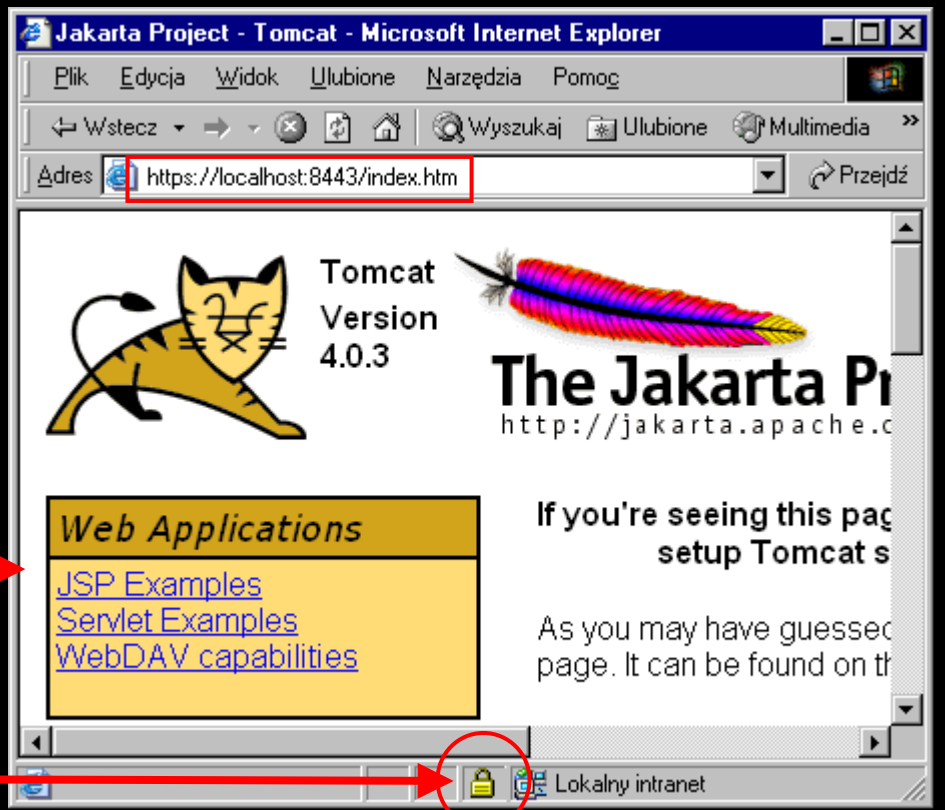
Plik server.xml (Tomcat)

W pliku konfiguracyjnym podano lokalizację plik certyfikatu certs wygenerowanego za pomocą programu keytools

Połączenie HTTPs

https://localhost:8443/index.htm

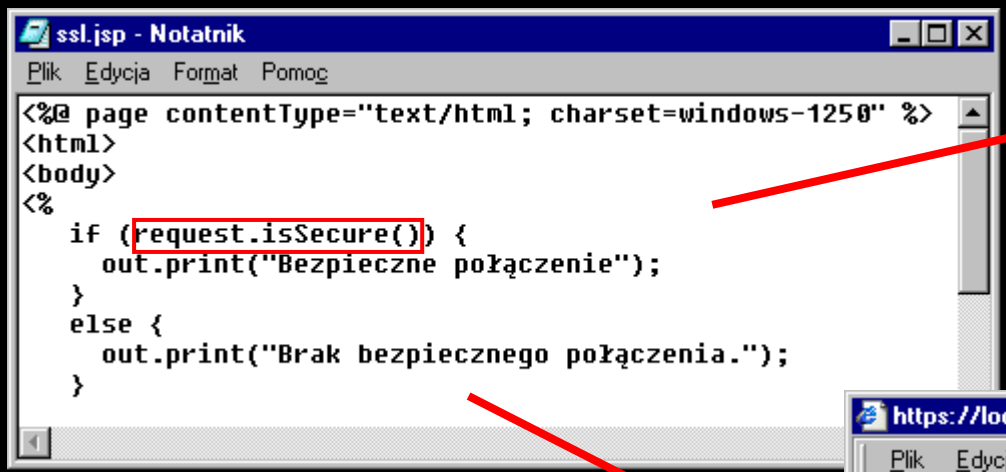


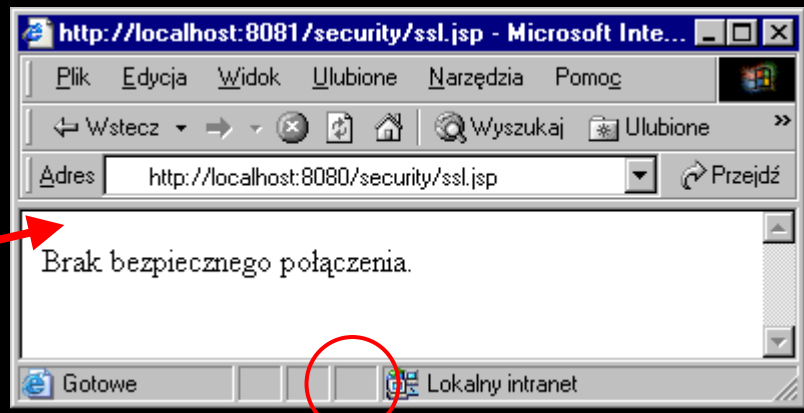


Witryna wykorzystuje bezpieczne połączenie klient-serwer szyfrowane najlepszym kluczem

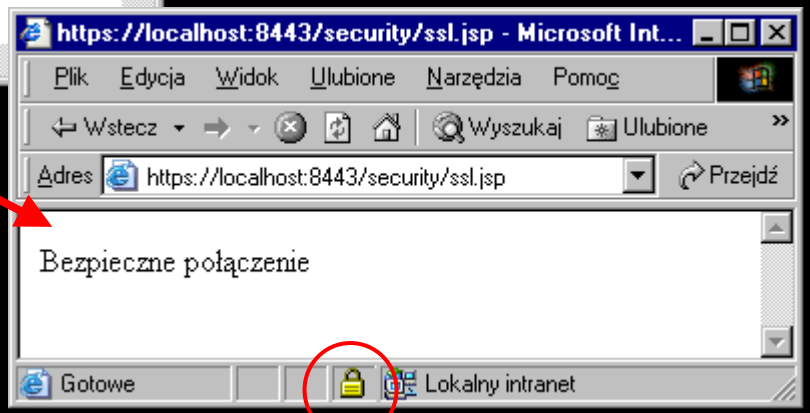
Test rodzaju połączenia

Strona JSP (serwer)



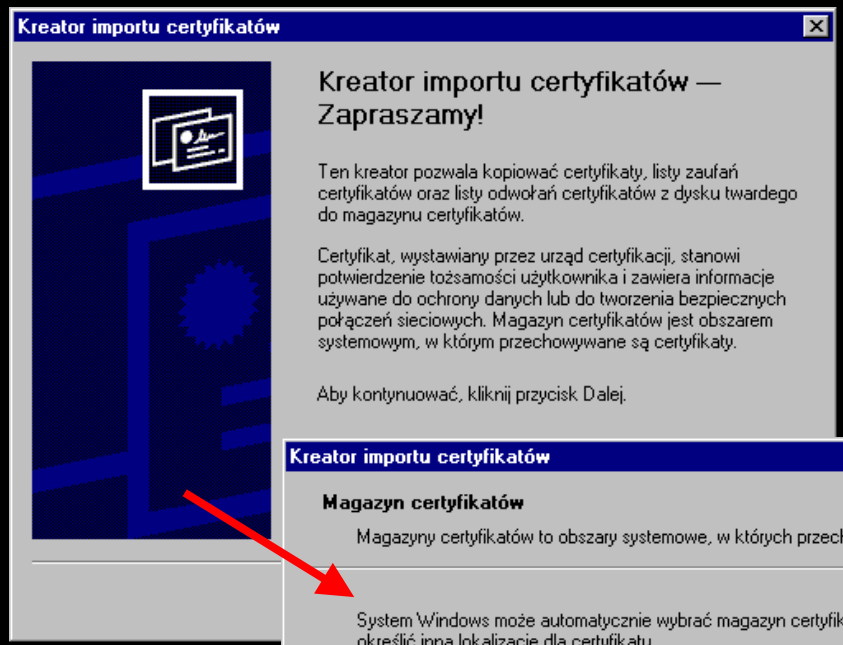


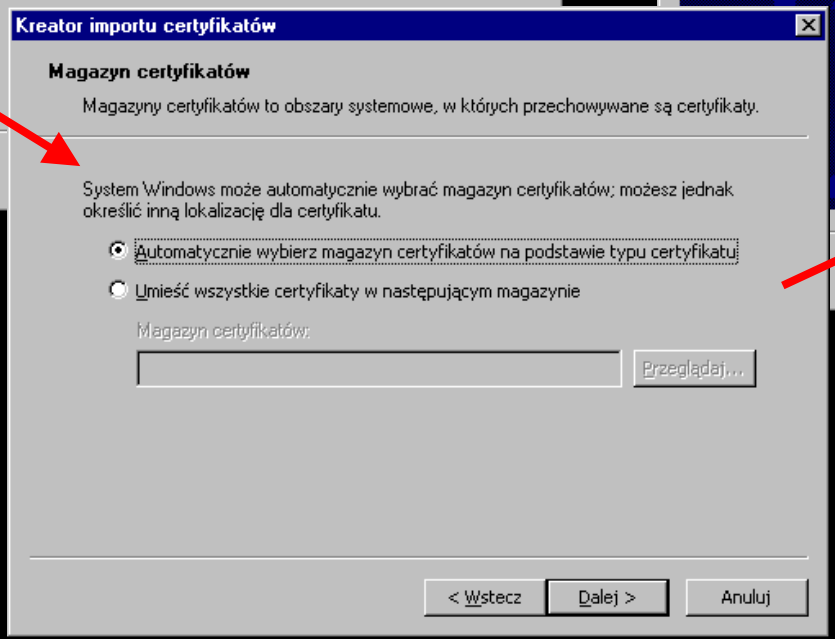
Strony http (klient)



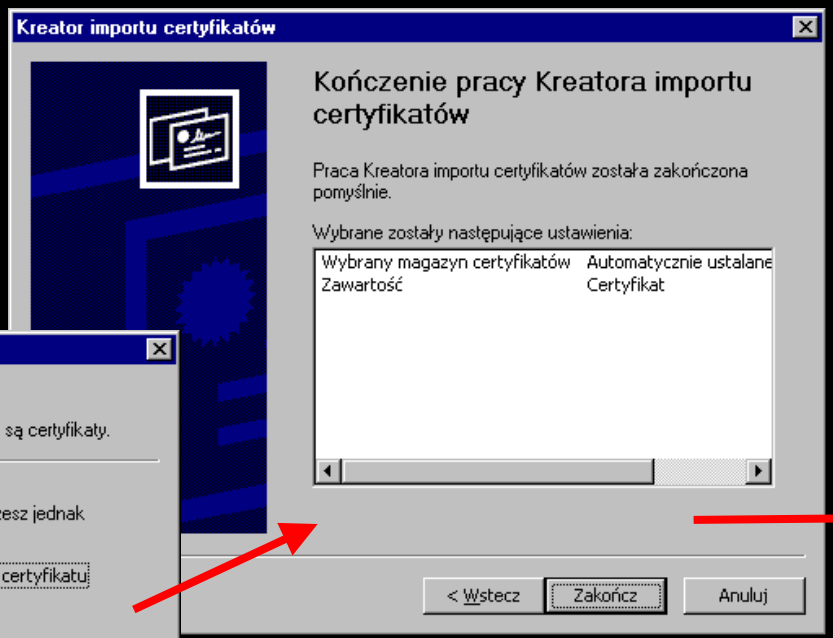
Komunikacja pomiędzy klientem i serwerem odbywa się przy wykorzystaniu HTTPS

Instalacja certyfikatu (klient)





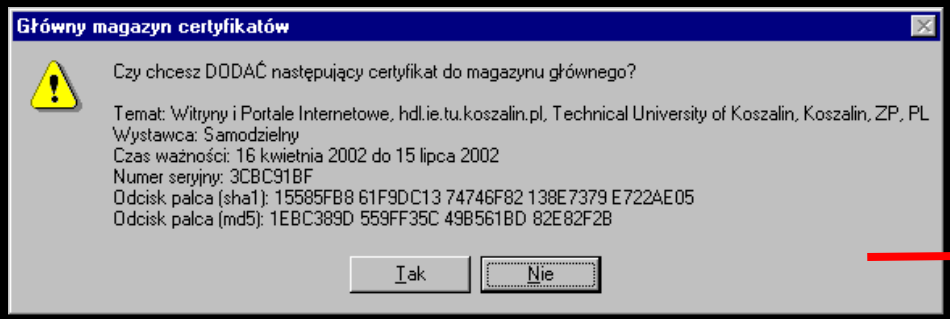
Przeglądarka (klient)



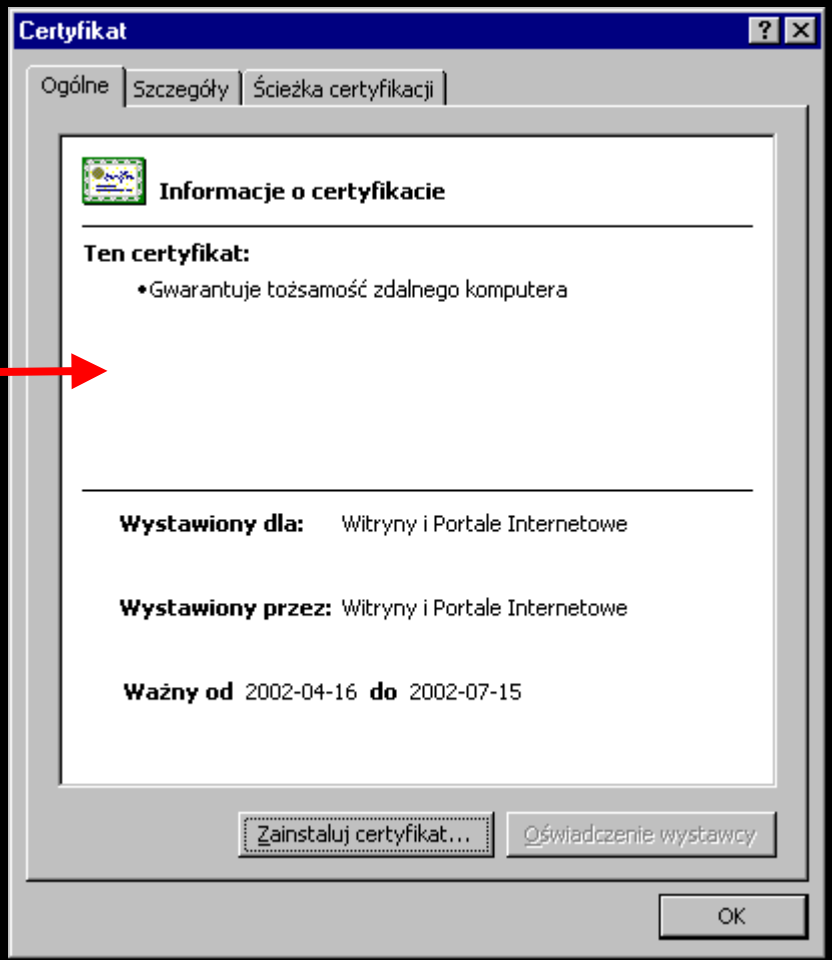
Następuje instalacja pobranego certyfikatu z serwera do magazynu certyfikatów klienta

Instalacja certyfikatu (klient)

Przeglądarka (klient)

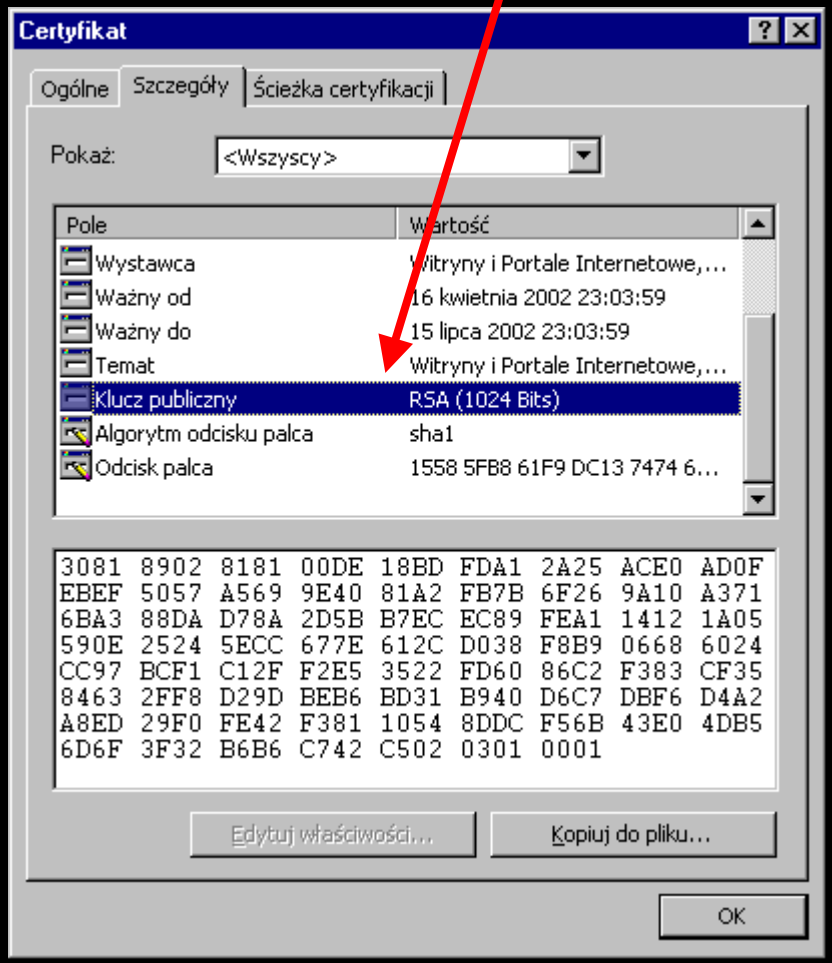


Certyfikat po zatwierdzeniu przez klienta

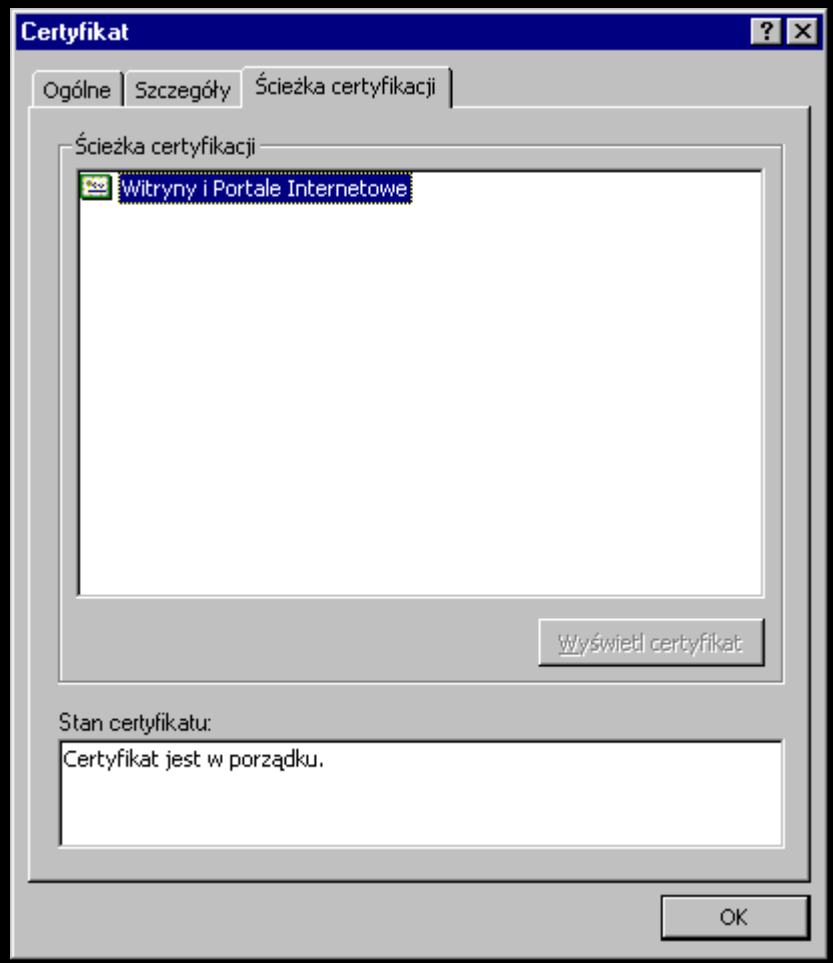


Instalacja certyfikatu (klient)

Klucz publiczny certyfikatu



Przeglądarka (klient)



Security Socket Layer (SSL)

http://java.sun.com/security/ssl/API_users_guide.html

SSL free: http://java.sun.com/products/jsse/

Tworzenie i modyfikacja certyfikatów:

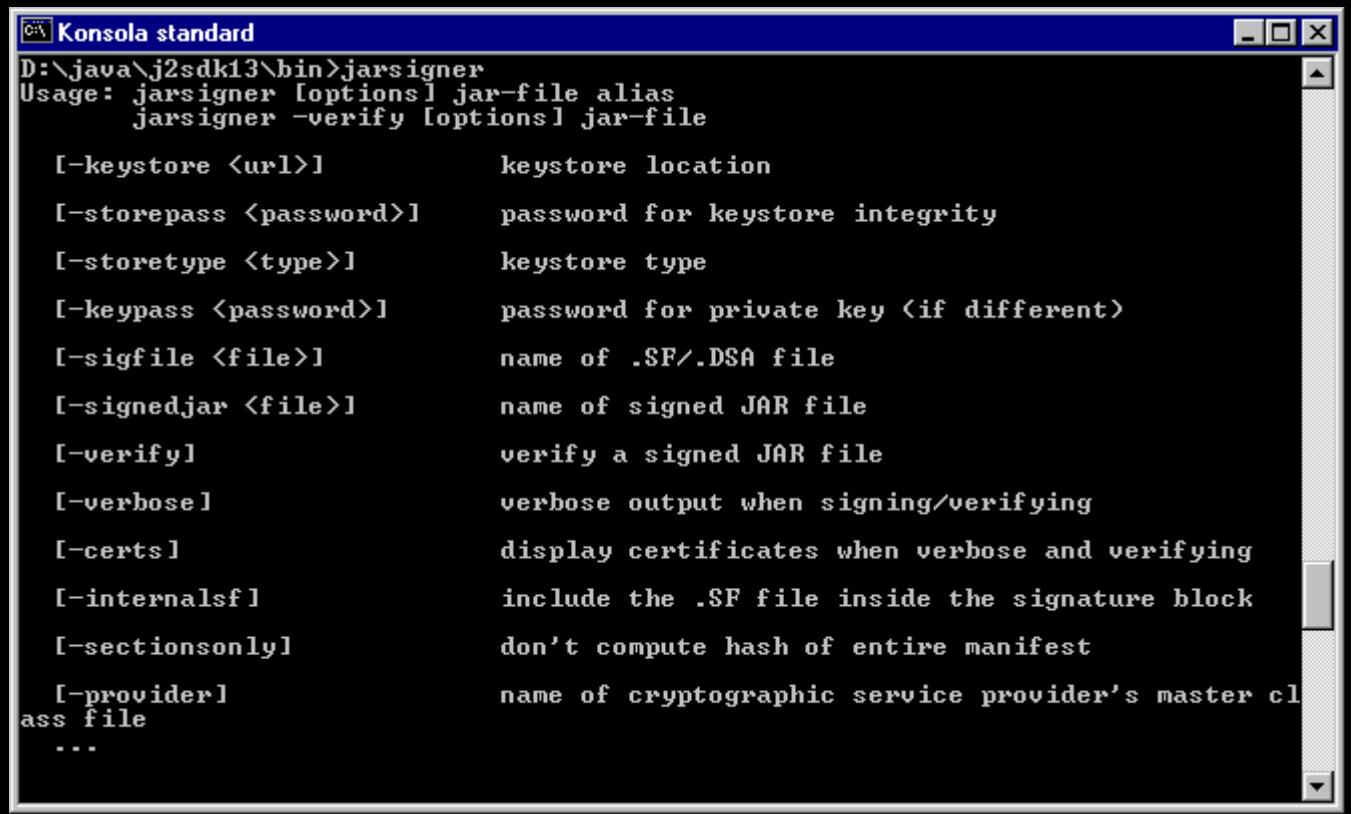
http://www.pseudonym.org/ssl/wwwj-index.html

http://www.openssl.org/docs/apps/openssl.html

JARsigner

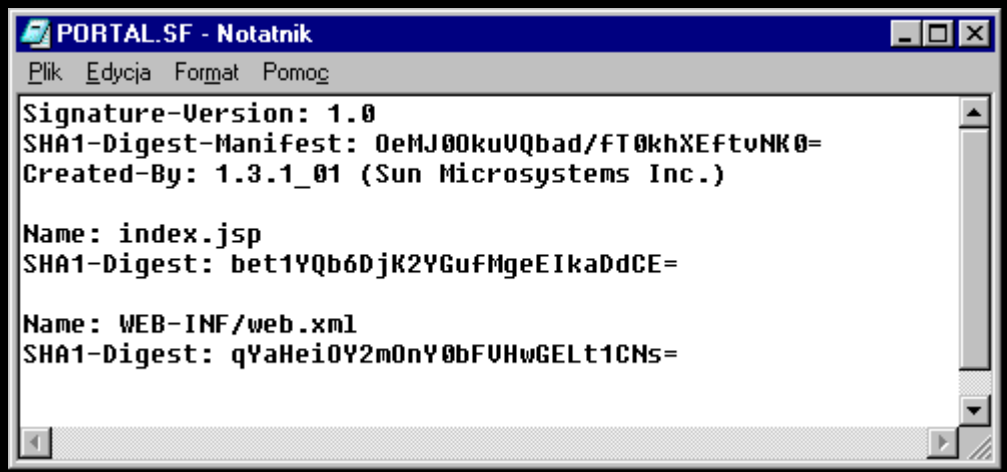
jarsigner

Program jarsigner służy do tworzenia sygnatur plików, weryfikacji i ich autentykacji



jarsigner

```
jarsigner -keystore certs -storepass portal
-signedjar sign_witaj.war witaj.war portal
```



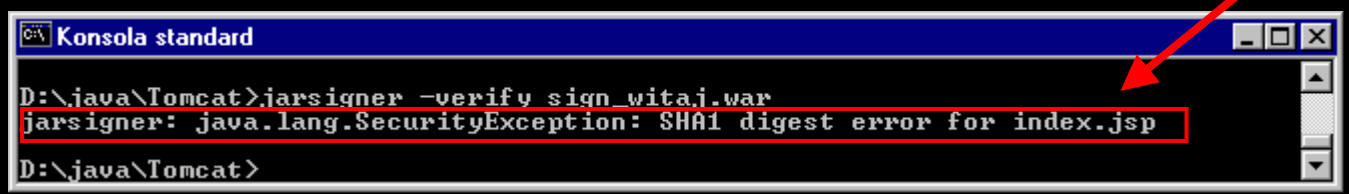
Do archiwum war zostaje dołączony blok sygnatury portal.sf oraz binarny plik klucza portal.rsa

jarsigner

Weryfikacja pliku:

```
jarsigner -verify sign_witaj.war
```

W przypadku zmiany jednego z plików: index.jsp podczas weryfikacji sygnalizowany jest błąd (archiwum uległo modyfikacji)



Java Cryptography Extension (JCE)

JCE (Java Cryptography Extension) - dostęp do algorytmów kryptograficznych za pośrednictwem API (interfejs umożliwia stosowanie wielu implementacji JCE)

- DES, RC2, IDEA
- RC4
- PBE (password-based encryption)
- Key Agreement
- MAC (Message Authentication Codes)

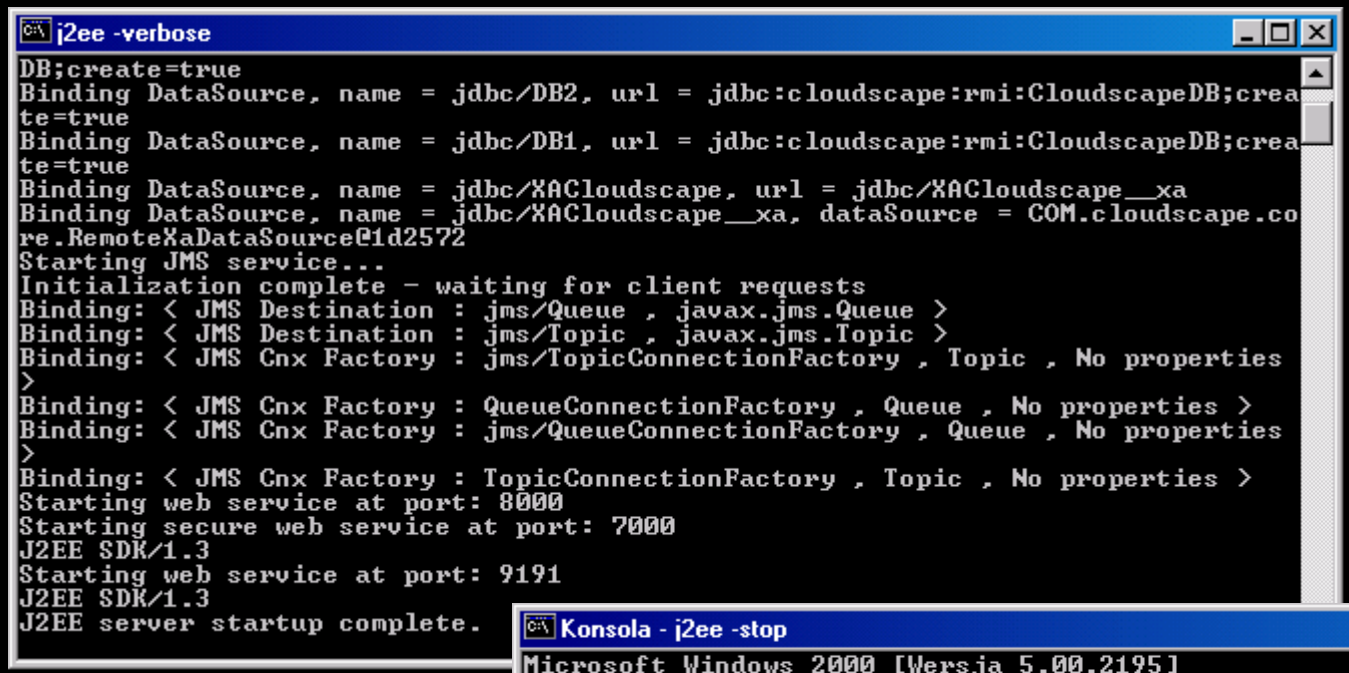
Serwer J2EE

Instalacja i uruchomienie

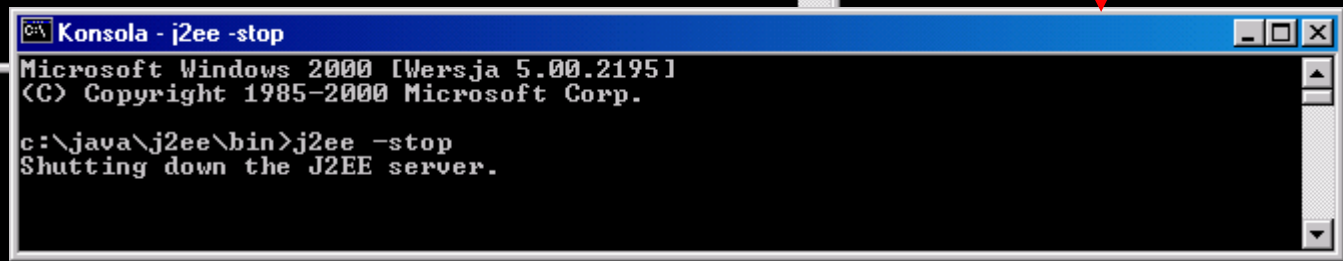
- instalacja serwera np. do katalogu c:\java\j2ee
- ustawienie zmiennych środowiskowych
  JAVA_HOME=c:\java\jdk
  j2EE_HOME=c:\java\j2ee
- start serwera c:\java\j2ee\bin\j2ee -verbose
- test serwera http://localhost:8000
- zatrzymanie serwera

  c:\java\j2ee\bin\j2ee -stop

Uruchomienie i zatrzymanie serwera



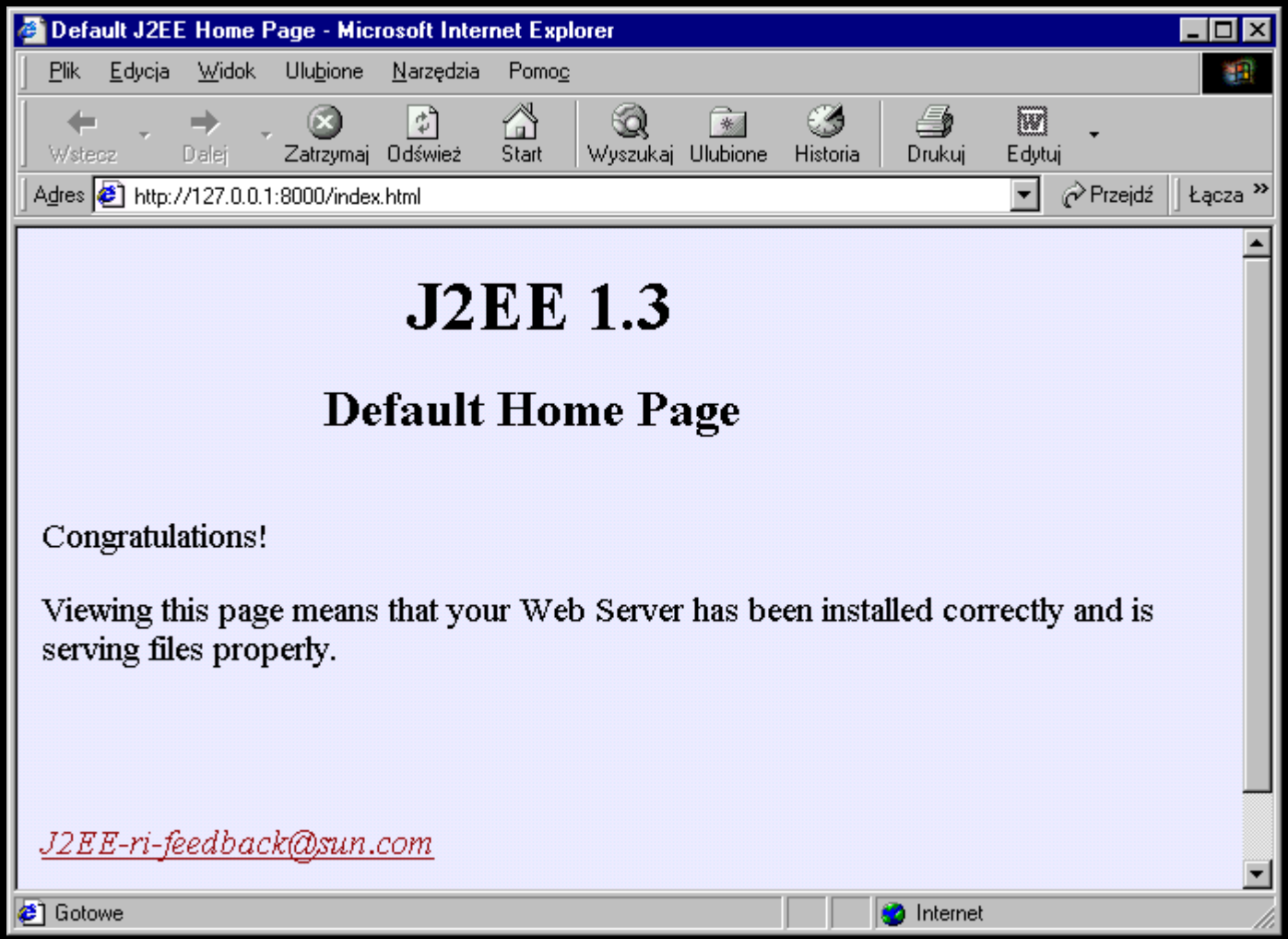
j2ee -stop



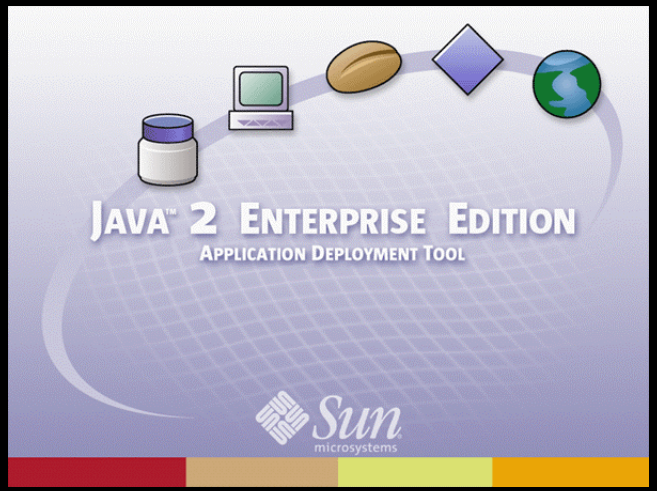
j2ee -verbose

W celu uruchomienia serwera w środowisku Windows 9x należy dokonać modyfikacji plików startowych

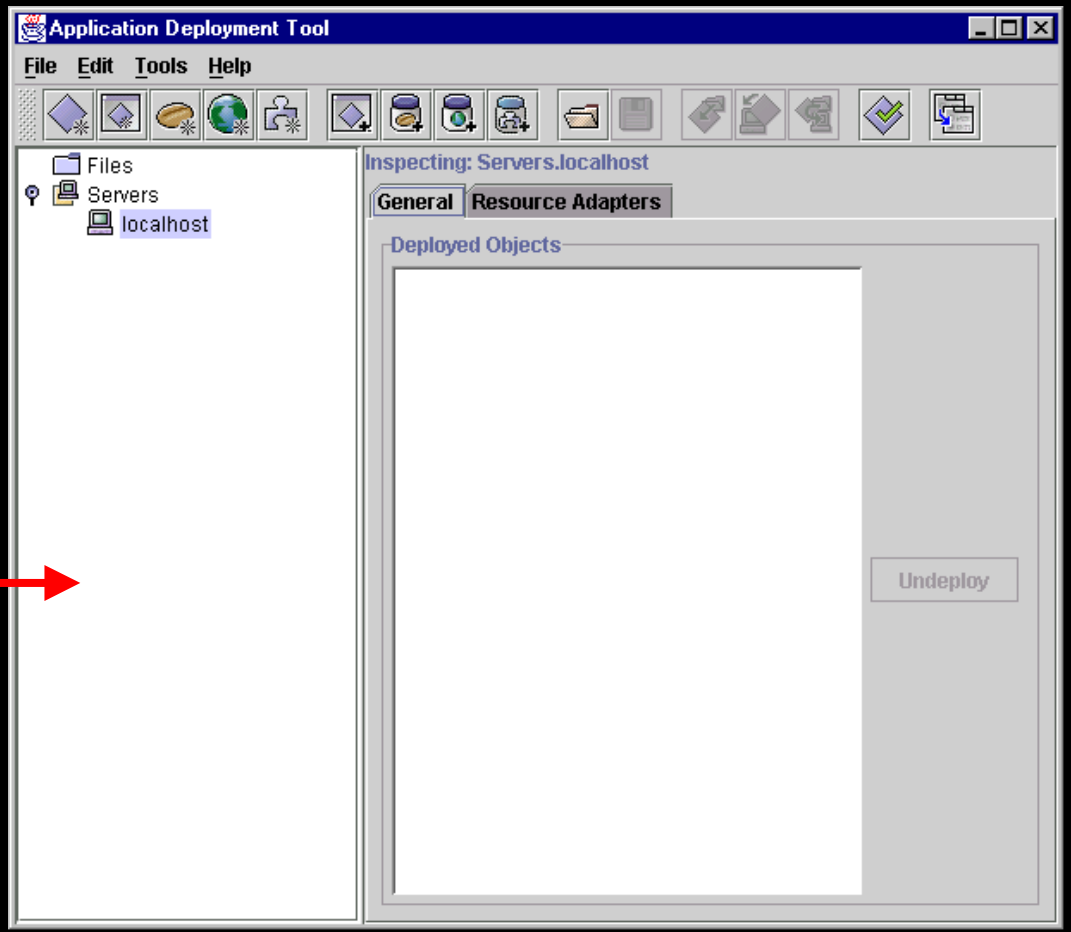
http://localhost:8000



Application Deployment Tool

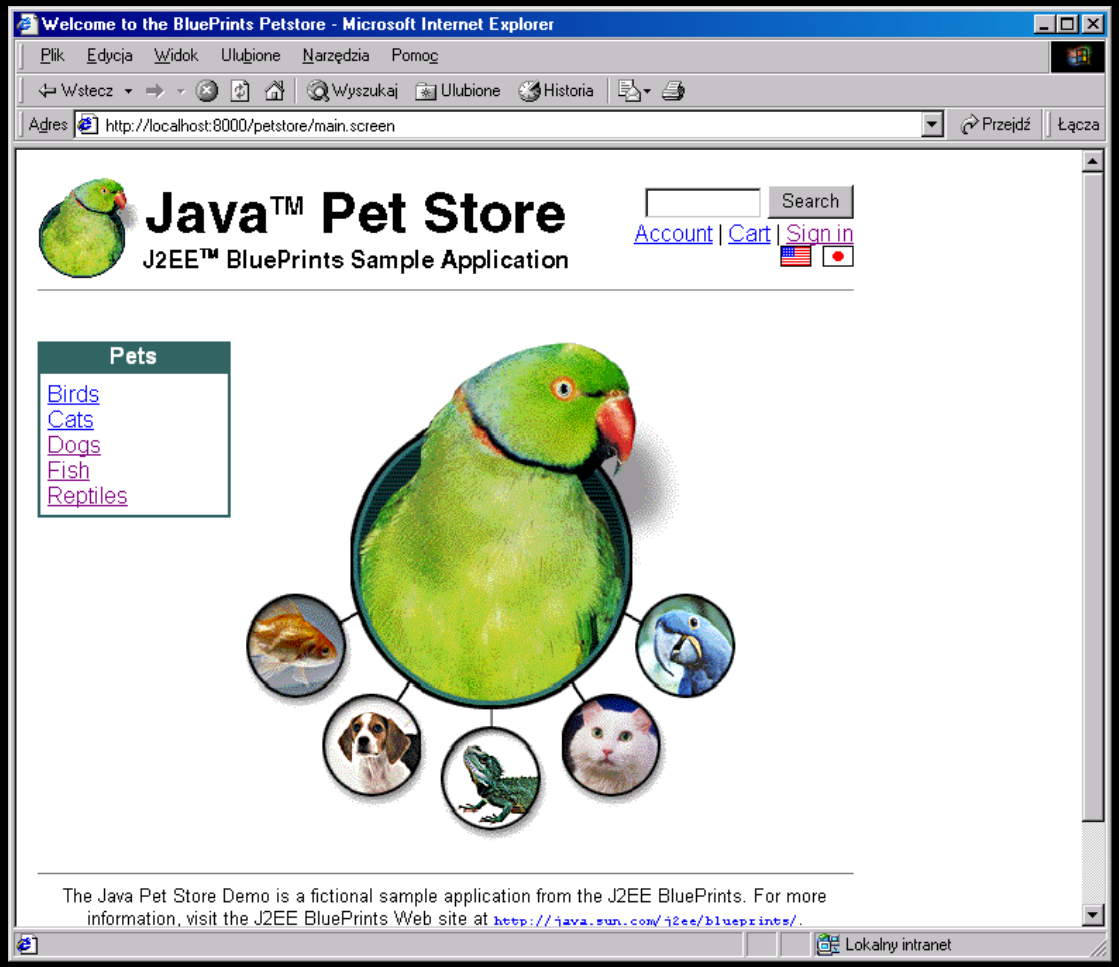


deployment



Program zarządza archiwami aplikacji Enterprise (EAR) oraz ich instalacją na serwerze aplikacyjnym

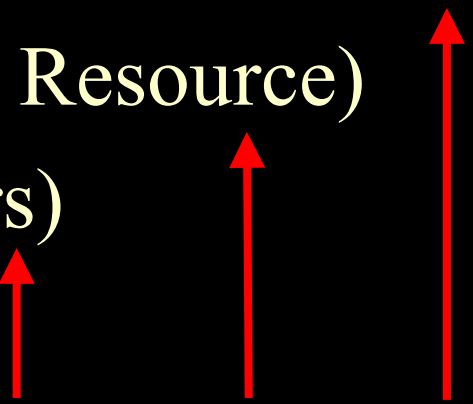
Wzorcowa aplikacja J2EE



http://localhost:8000/petstore

Deskryptory rozmieszczenia (Deployment Descriptions)

- EAR (Enterprise Application Resource)
- WAR (Web Application Resource)
- RAR (Resource Adapters)
- EJB
- Klient JAR



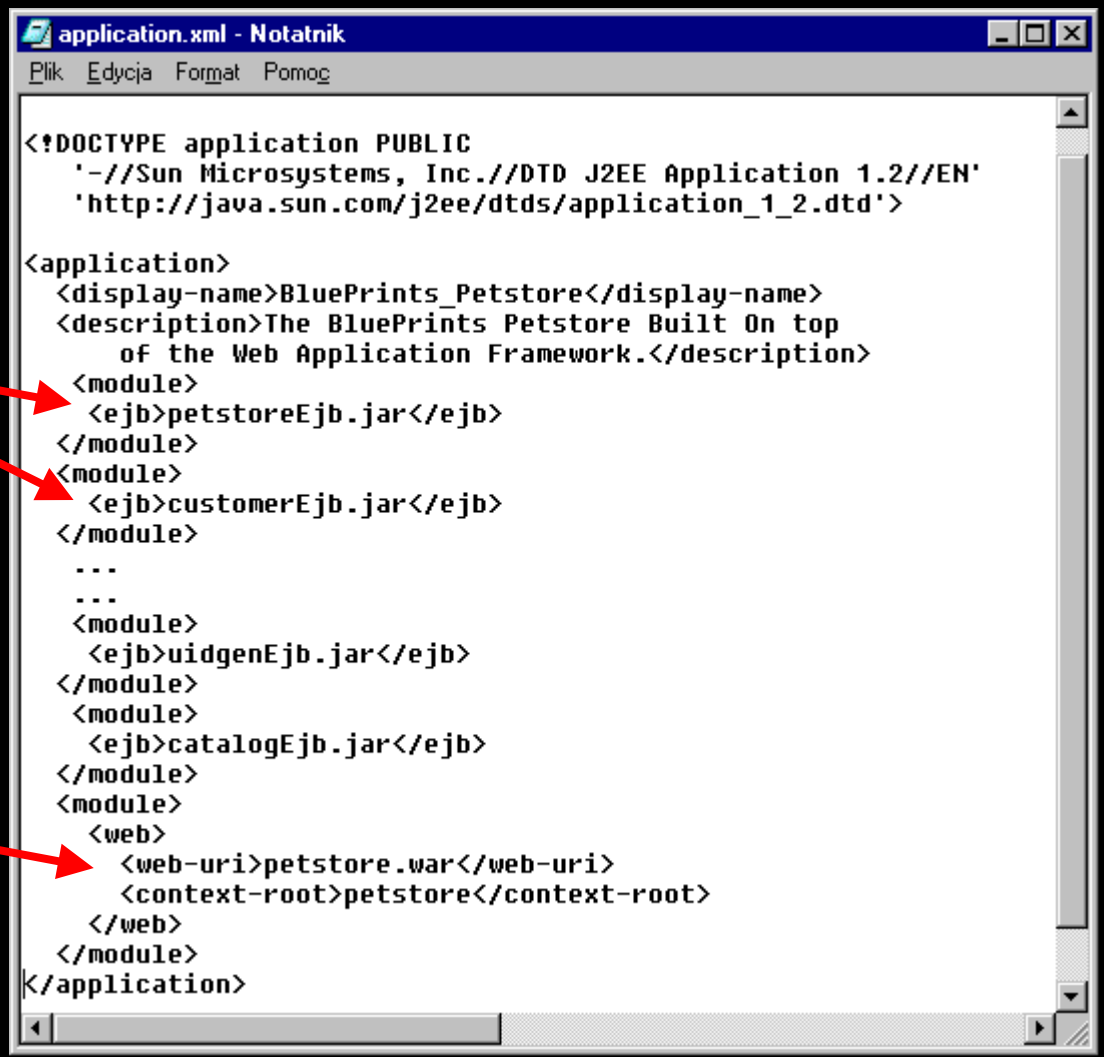
Deskryptory DD opisują wewnętrzną strukturą archiwum aplikacji

Deployment Description aplikacji Enterprise (*.ear)

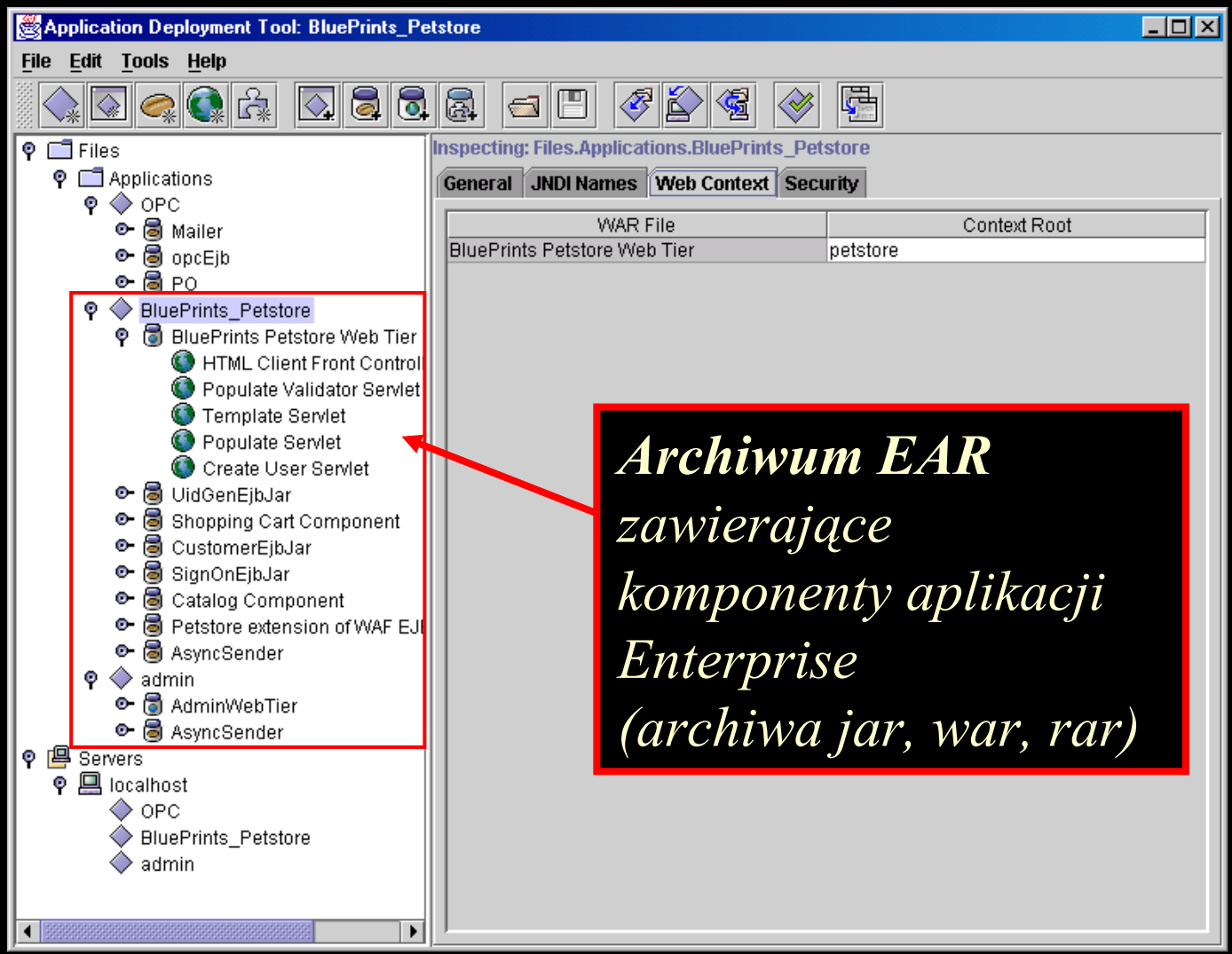
Application.xml
(deskryptor aplikacji Enterprise)

komponenty EJB
(archiwa jar)

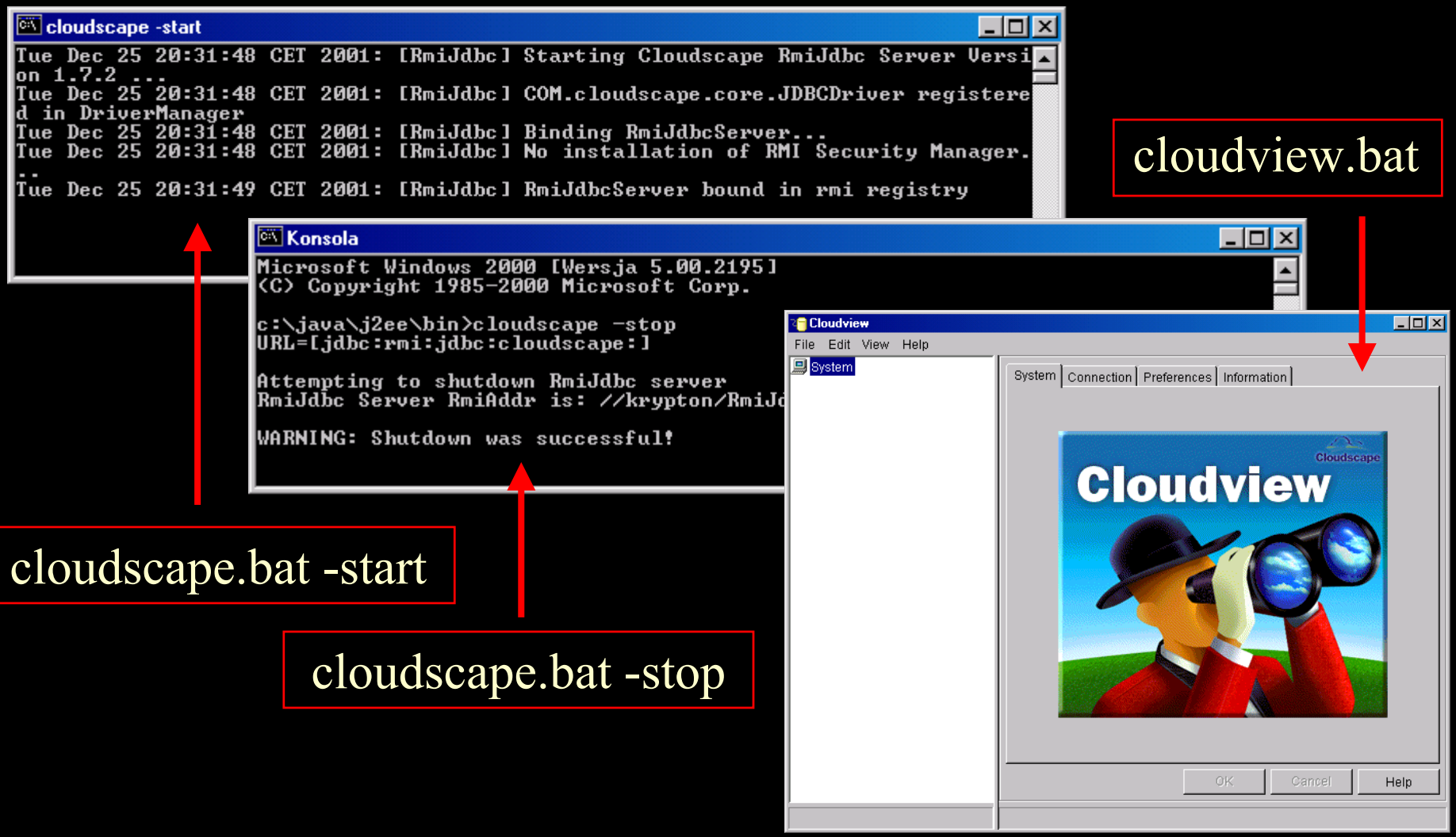
komponenty WEBowe
(archiwa war)



PetStore (deployment)



Cloudscape
(baza danych pakietu J2EE)

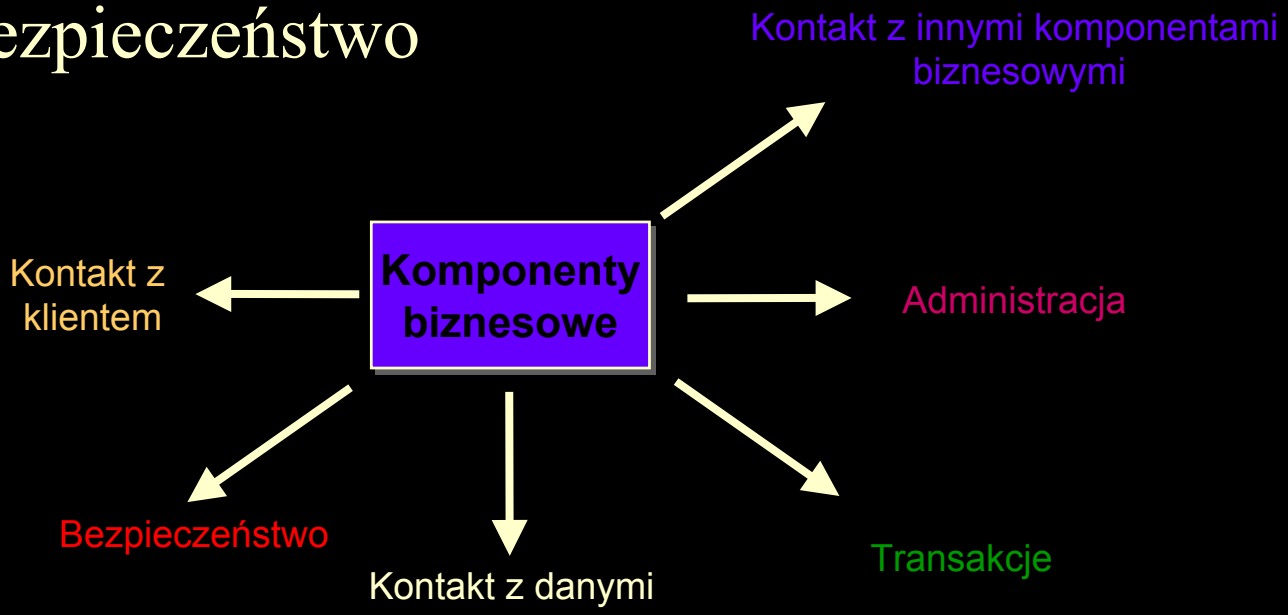


EJB

Enterprise Java Bean

Architektura EJB jest przeznaczona do tworzenia aplikacji bazujących na rozproszonych komponentach

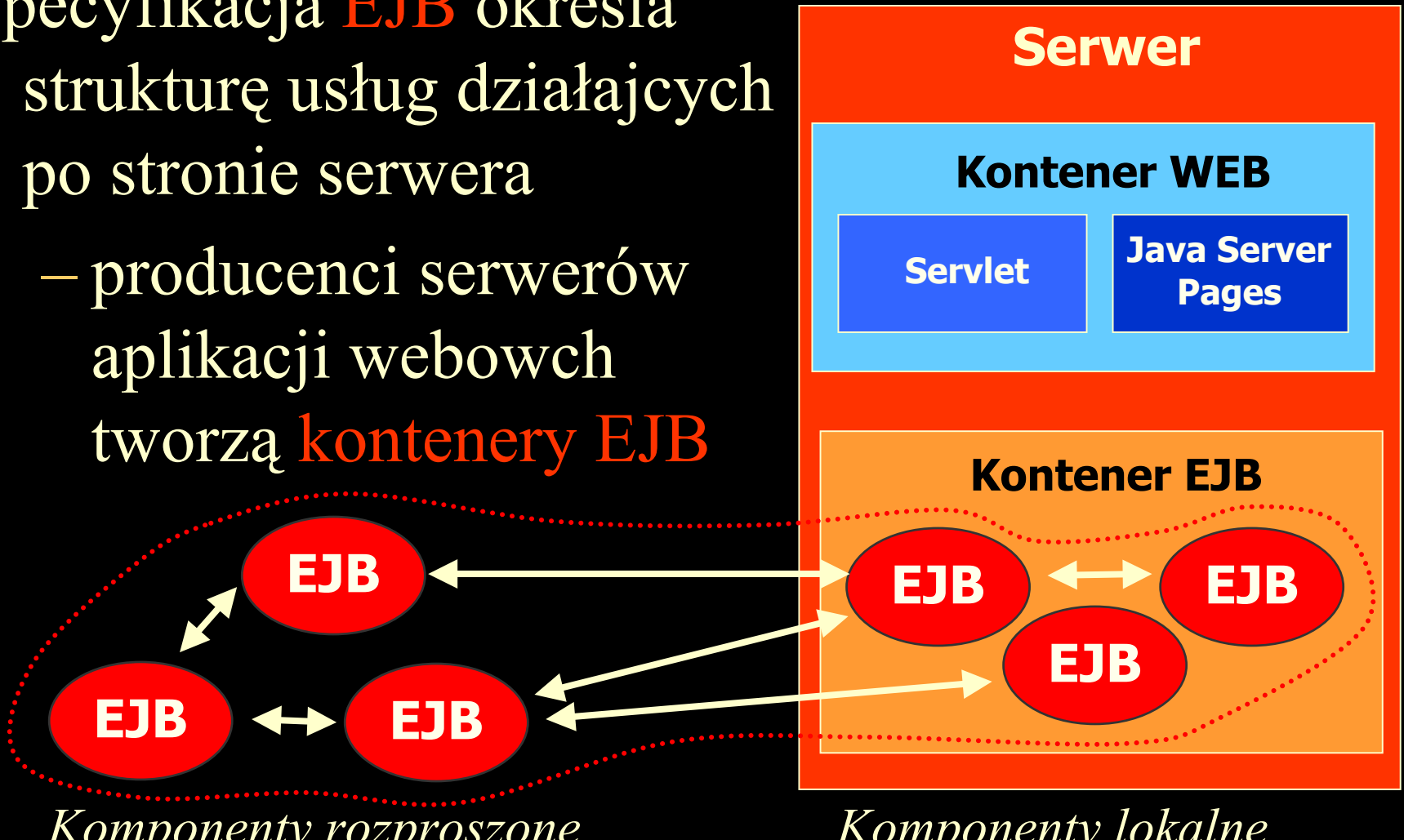
- wspiercie transakcji
- bezpieczeństwo



Enterprise Java Bean

Specyfikacja EJB określa strukturę usług działajcych po stronie serwera

- producenci serwerów aplikacji webowch tworzą kontenery EJB



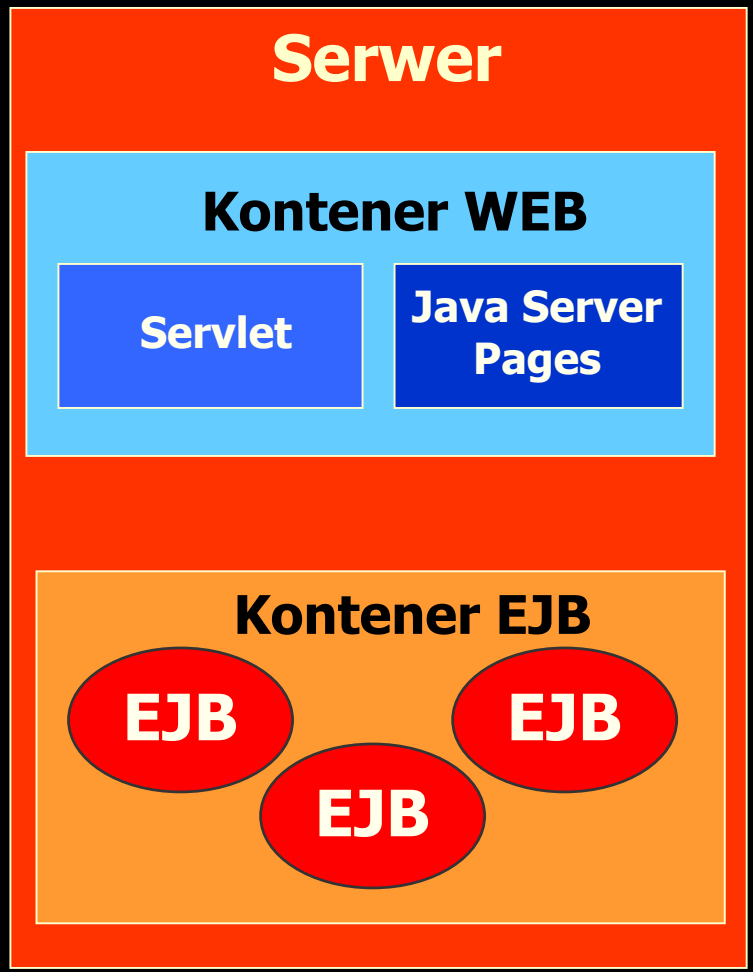
Enterprise Java Bean

Kontener EJB zarządza komponentami EJB (klasy i ich instancje)

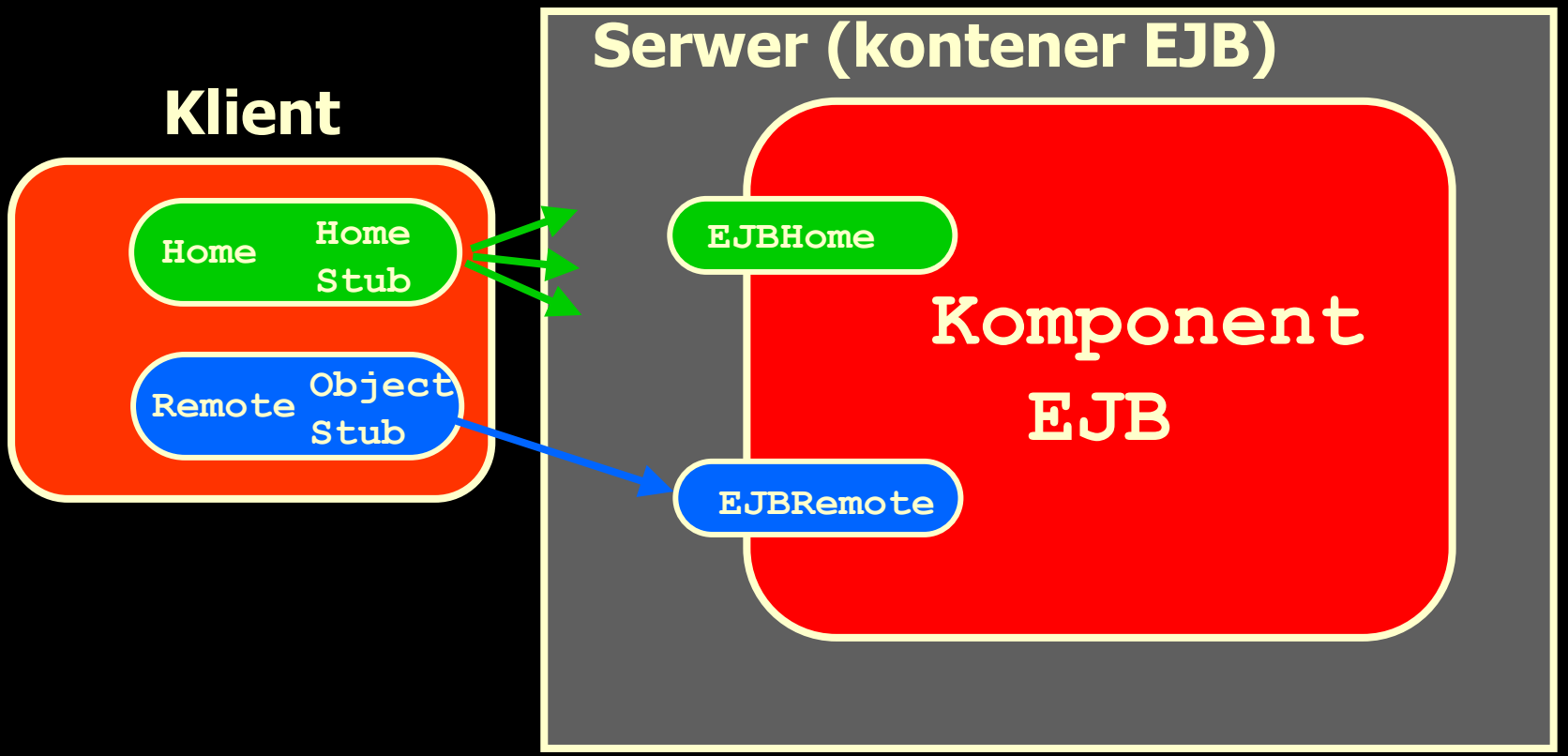
Kontener jest odpowiedzialny za dostarczanie do komponentu EJB takich usług jak:

- kontrola transakcji
- bezpieczeństwo
- zarządzanie komponentami

Kontener jest niewidoczny dla klienta ani dla przechowywanego w nim komponentu EJB.

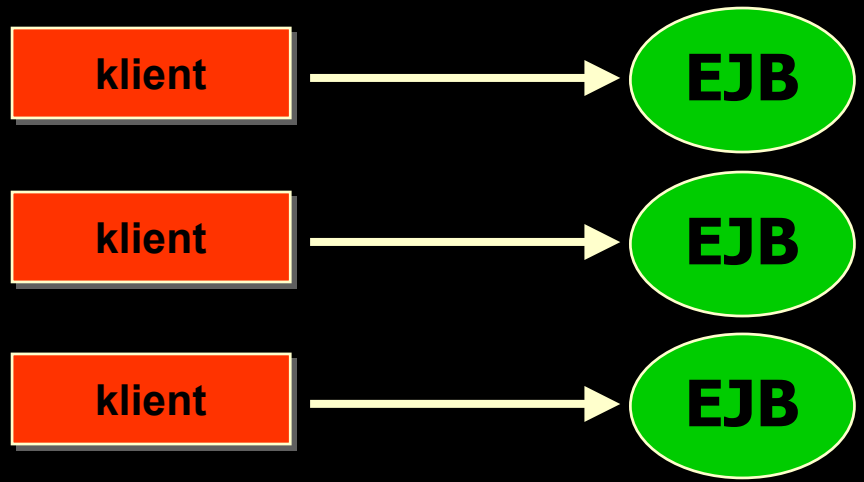


Architektura EJB

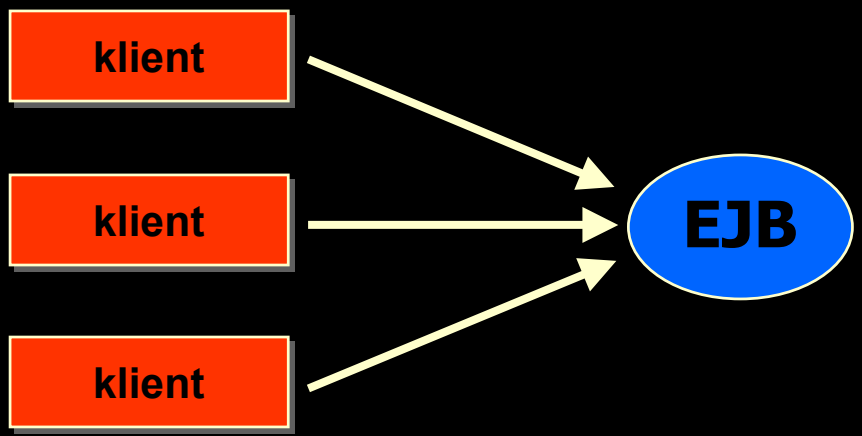


Dostęp do komponentów EJB za pomocą zdefiniowanych interfejsów (HOME i REMOTE)

Rodzaje komponentów EJB

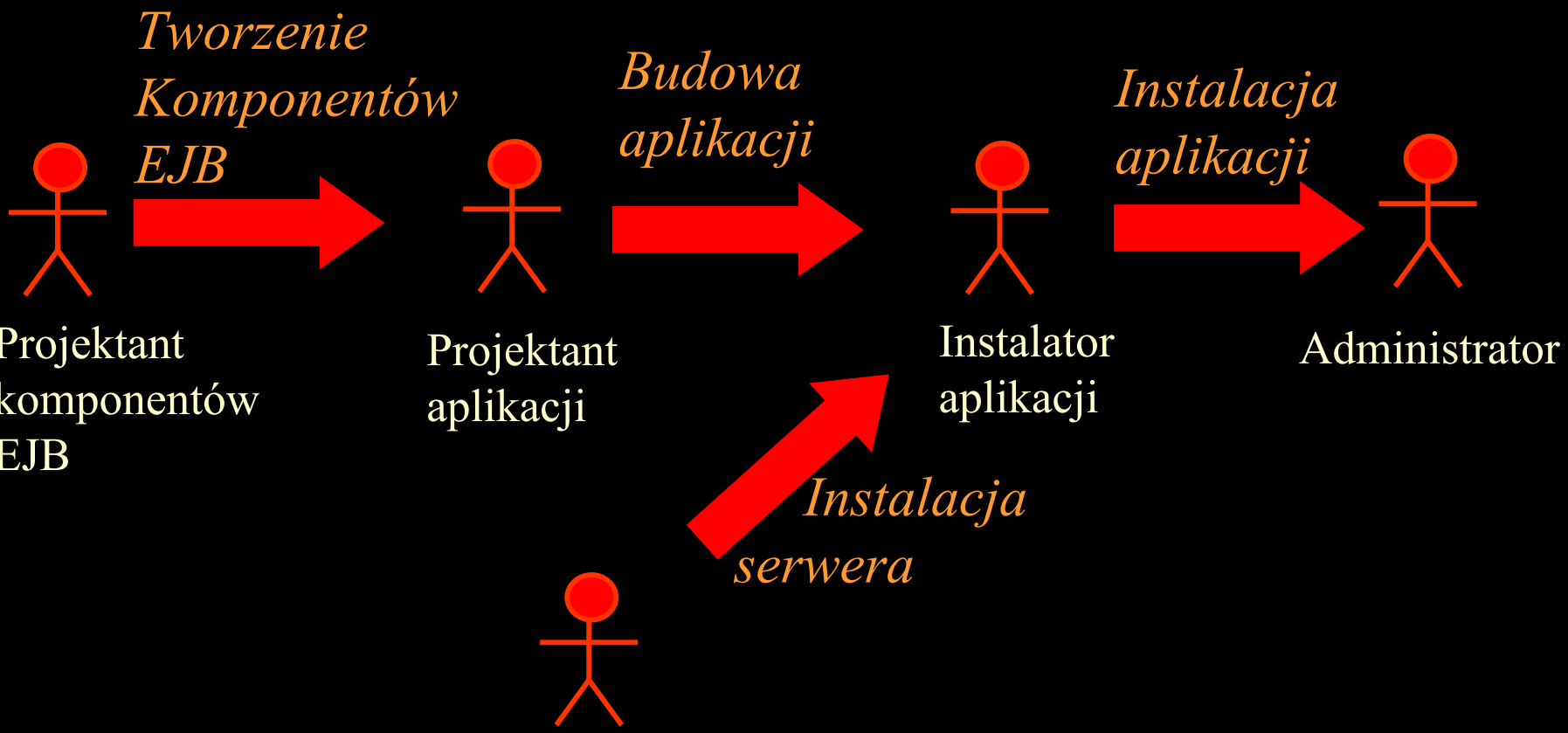


Komponent typu Session
(reprezentacja klienta po stronie serwera)



Komponent typu Entity
(obiekt biznesowy reprezentujący przechowywanie danych)

Dystrybuja komponentów EJB



Zalety komponentów EJB

- otwartość standardu
- wieloplatformowość javy
- izolacja od problemów połączeń oraz transakcji
- dystrybucja EJB w sieci serwerów
- wielokrotne użycie komponentów

Transakcje

Model ACID

ATOMIC – atomowe (niepodzielne)

CONSISTENCY – spójne (w przypadku, gdy któryś z elementów transakcji nie udał się to cała transakcja powinna zostać anulowana)

ISOLATED – izolowane (zmiany są widoczne dopiero po ich zatwierdzeniu)

DURABLE – trwałe (wynik zatwierdzonej transakcji jest dobrze zapisany w systemie i nie zginie w czasie awarii)

Rodzaje transakcji

- Not Supported - metoda nie obsługuje transakcji
- Supported - metoda obsługuje transakcje, ale jej nie wymaga
- Required - metoda wymaga transakcji (jeżeli wywołuje ją metoda nie będąca transakcją, jest tworzona nowa transakcja)
- Requires New - metoda zawsze wymaga nowej transakcji
- Mandatory - metoda wymaga, aby metoda wywołująca posiadała transakcje
- Bean managed - bean wprowadza swoja własną transakcje, bazując na swoich specyfikacjach

Java Transaction API (JTA)

Interfejs wspierajcy rozproszone transakcje wykorzystywany przez mechanizmy J2EE

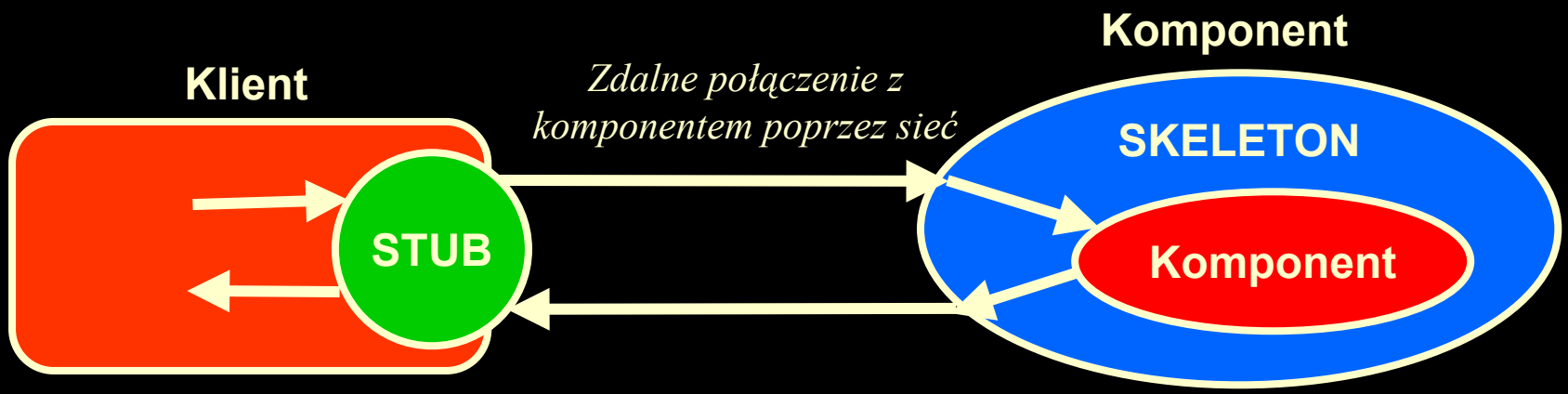
Technologia J2EE definiuje transakcje w sposób deklaratywny (serwer aplikacji J2EE)

- commit (zachowanie transakcji)
- rollback (anulowanie transakcji)

RMI

Scenariusz dystrybucji obiektu

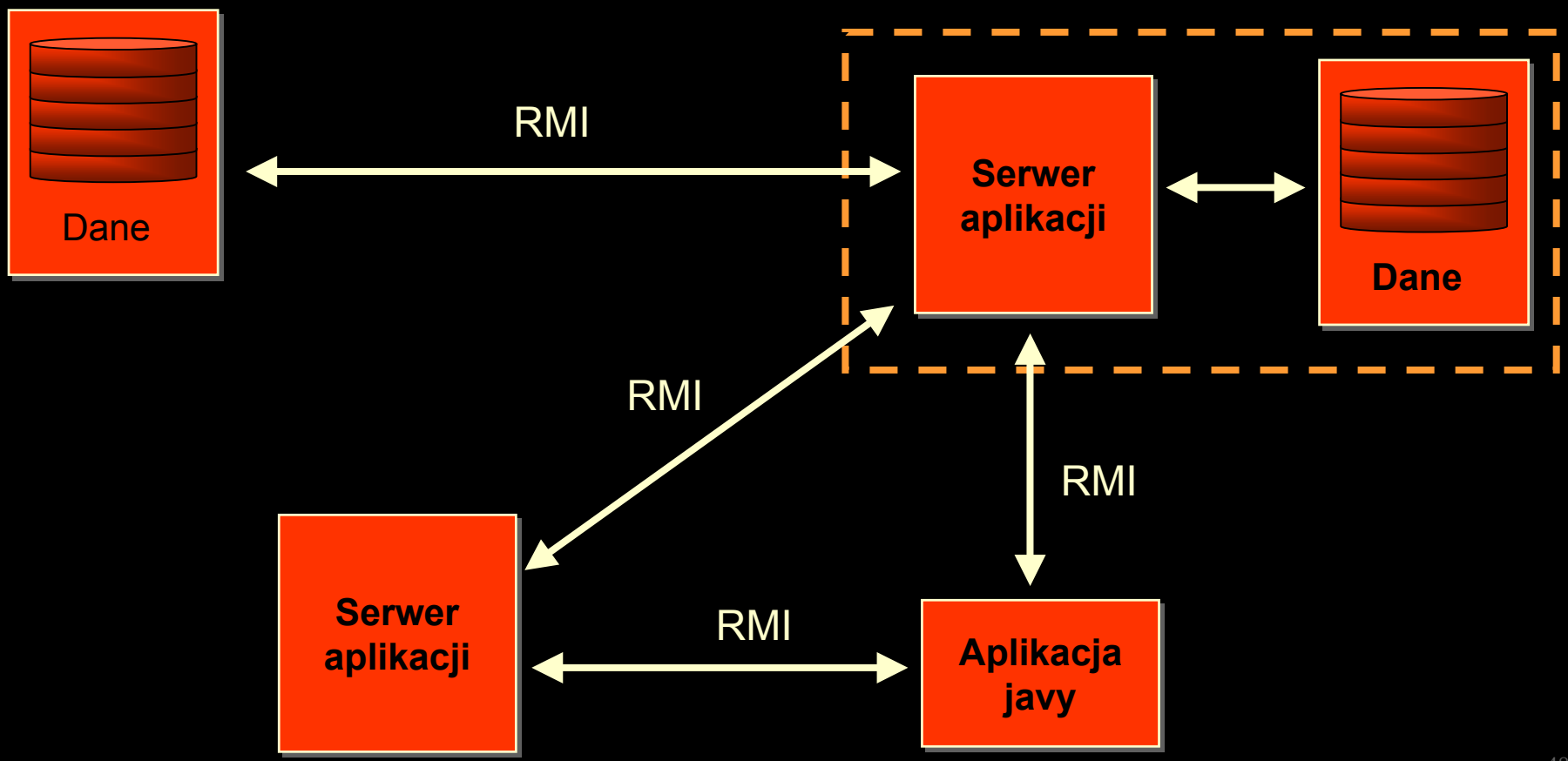
RMI – technologia umożliwiająca obiektom lokalnym na wywoływanie metod obiektów zdalnych



RMI - służy do komunikacji pomiędzy komponentami napisanymi w Javie
CORBA – komunikacja pomiędzy komponentami napisanymi w dowolnym języku

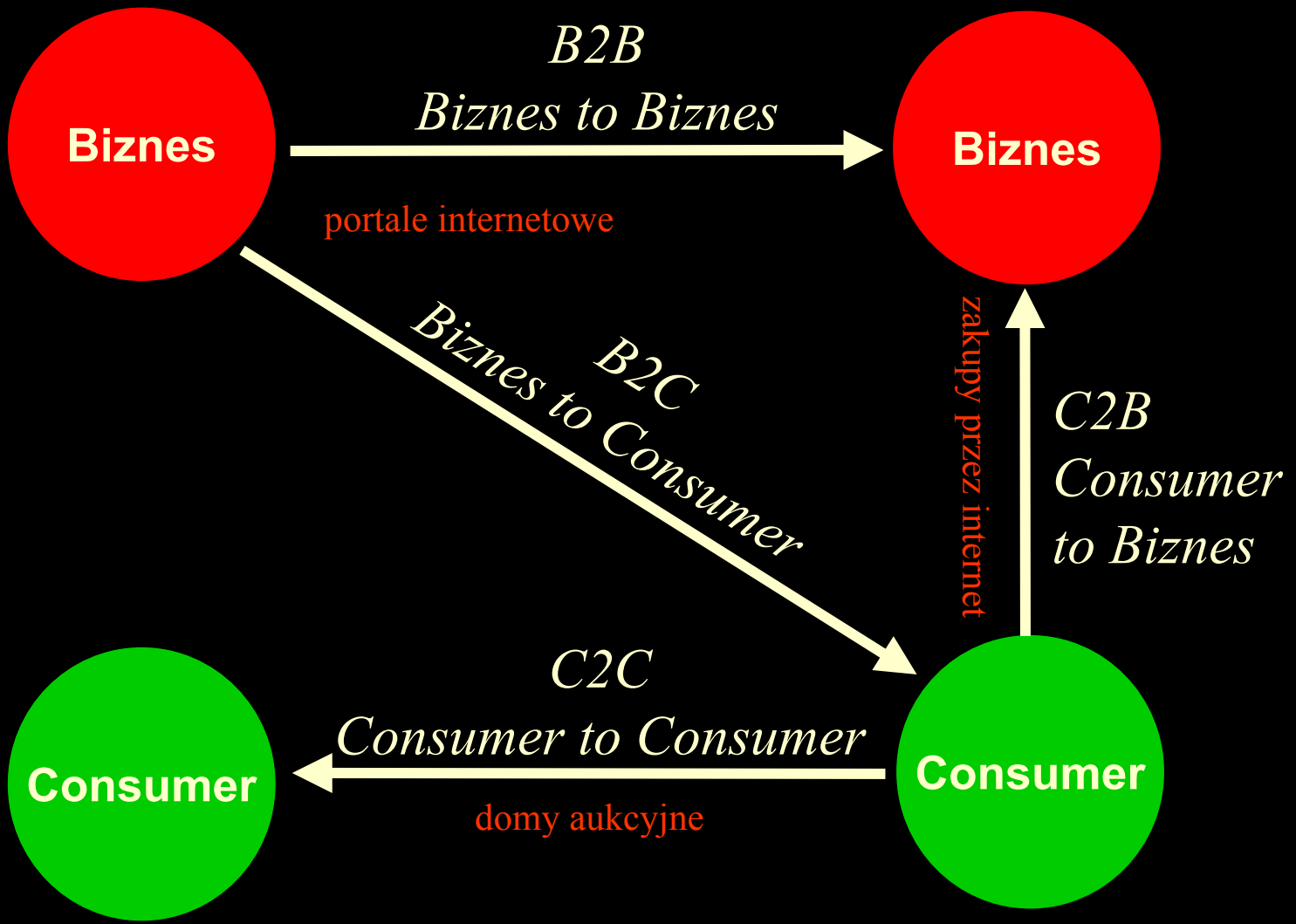
(Remote Method Invocation)

```
jdbc:rmi://192.168.170.27:1099/jdbc:cloudscape:db
```



Web Services

Rodzaje systemów



Fundamenty Web Services

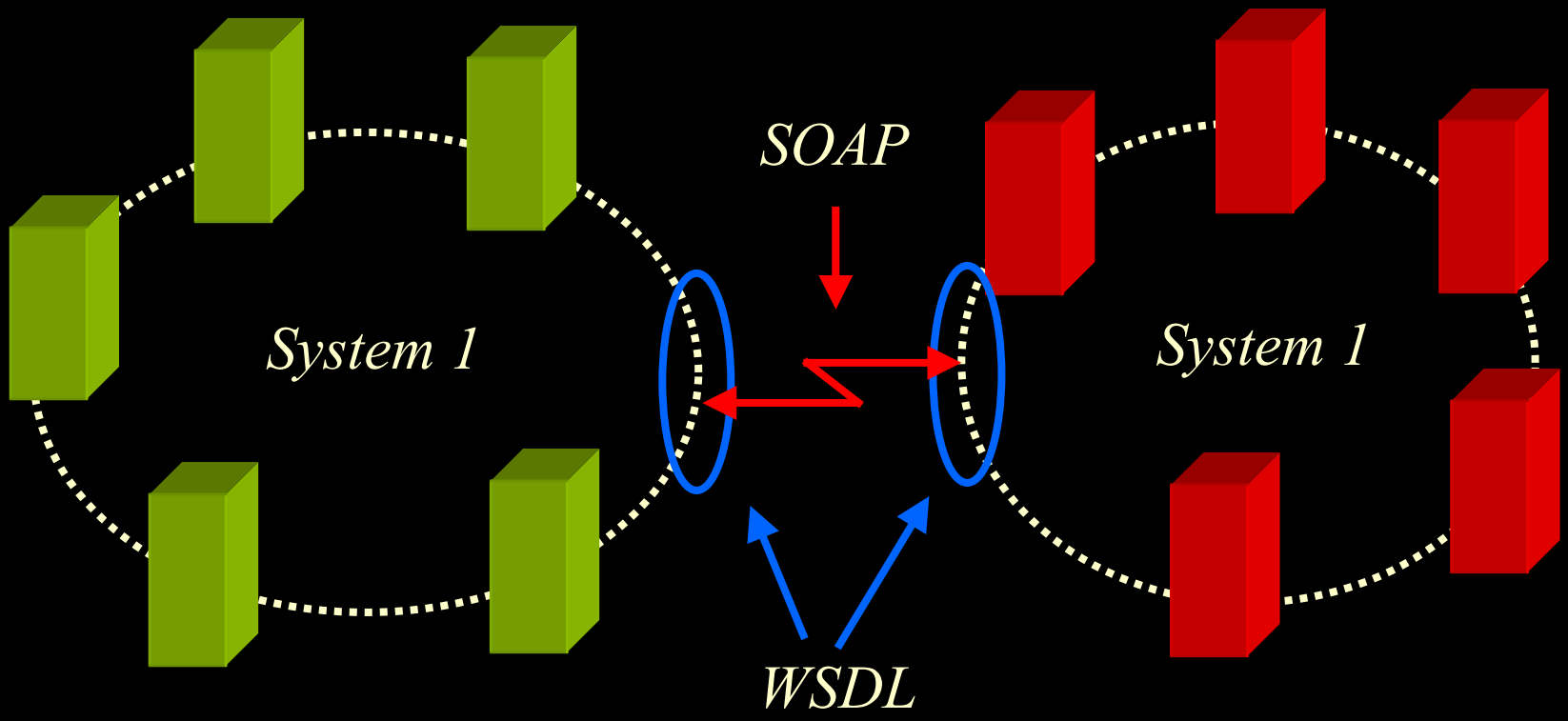
- XML - format wymiany i opisu danych
- SOAP - protokół wywoływania Web Services i przekazywania wiadomości
- WSDL - format opisu Web Services
- UDDI - miejsce wyszukiwania i rejestrowania Web Services

SOAP

Simple Object Access Protocol (SOAP) – protokół wymiany informacji w środowisku rozproszonym oparty o język XML w oparciu o przesyłanie TCP/IP

- opis zawartości informacji i sposób jej przetworzenia
- zestaw reguł definiujących typy danych
- sposób wywoływania oraz odpowiedzi zdalnych procedur

SOAP i WSDL

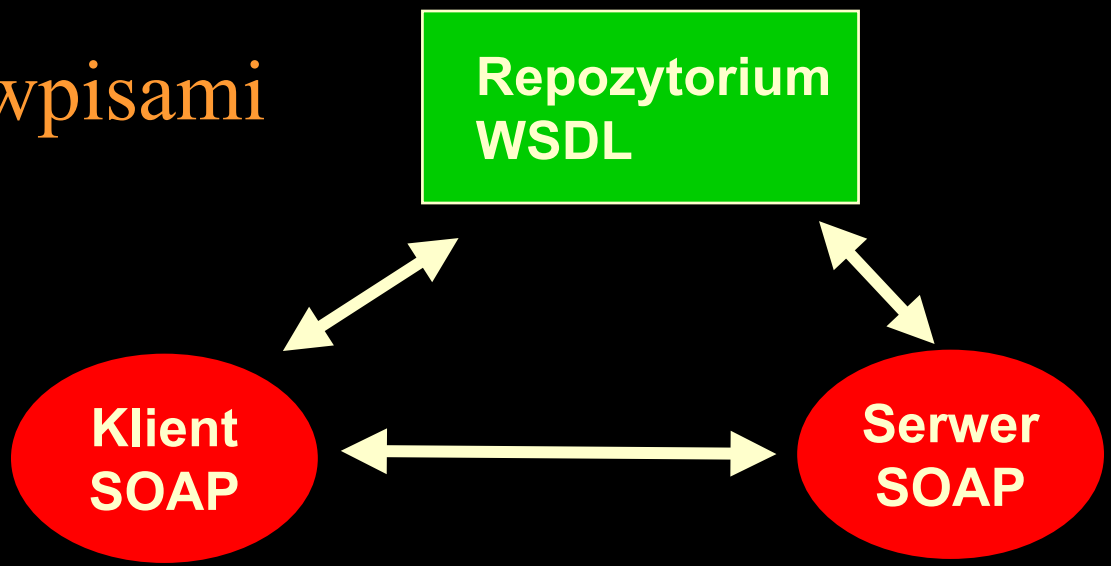


Komunikacja dwóch systemów za pomocą protokołu SOAP i języka opisu usług WSDL (Web Services Description Language)

UDDI (Universal Description, Discovery and Integration)

UDDI – system katalogowania usług opisanych za pomocą języka WSDL

- przeszukiwanie rejestru usług
- analiza usług
- manipulacja wpisami



UML (Unified Modeling Language)

Unified Modeling Language (UML) jest językiem specyfikacji, wizualizacji, konstrukcji i dokumentacji systemów informatycznych

Reprezentuje zbiór najlepszych praktyk inżynierskich, które dowiodły przydatności przy modelowaniu dużych i skomplikowanych systemów