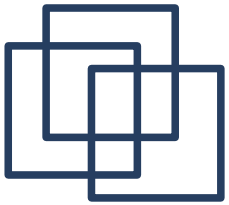
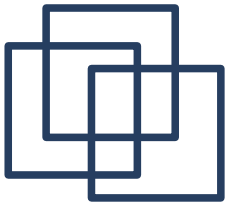


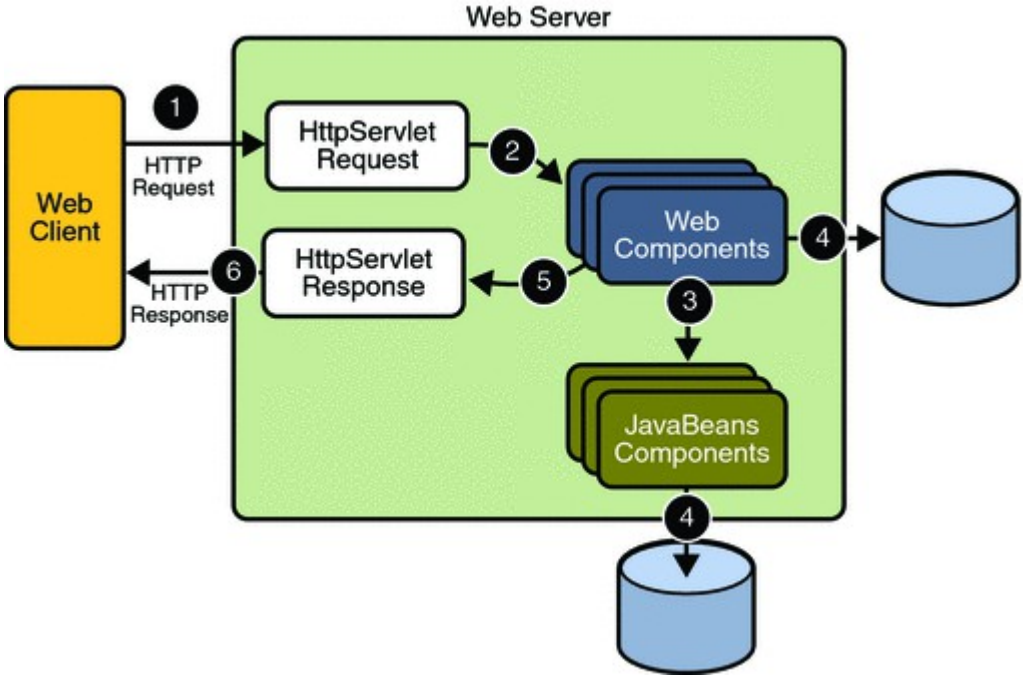
Elementy JEE

1. Wprowadzenie.
2. Prerekwizyty.
3. Pierwszy servlet.
- obsługa parametrów żądań
4. JavaServer Pages.

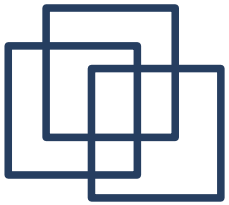


Java Enterprice Edition

Java Enterprice Edition (JEE) jest rozszerzeniem Java Standard Edition (JSE) o usługi wykorzystywane w tzw. aplikacjach biznesowych. Jednym z najczęściej wykorzystywanych elementów JEE są usługi związane z aplikacjami www.



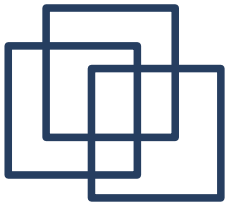
http://java.sun.com/javaee/5/docs/tutorial/doc/



Prerekwizyty

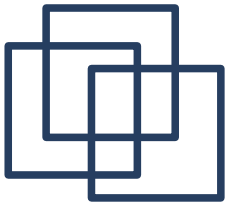
- Java SDK
- Apache Tomcat http://tomcat.apache.org/,
- ewentualnie plugin do Eclipse

http://www.eclipsetotale.com/tomcatPlugin.html#A3.



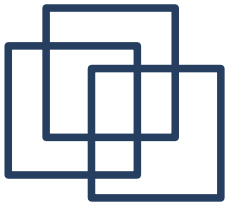
Pierwszy serwlet

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class FirstServlet extends HttpServlet{

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

        PrintWriter out = response.getWriter();
        out.print("Hello world");
        out.close();
    }
}
```



Pierwszy serwlet - web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         version="2.5"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
            http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <servlet>
    <servlet-name>first</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>first</servlet-name>
    <url-pattern>/firstUrl</url-pattern>
  </servlet-mapping>

</web-app>
```

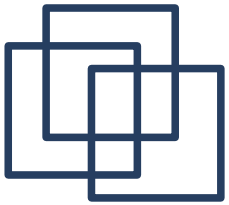


Struktura przykładu

WEB-INF
- **classes**
  - FirstServlet.class
- **lib**
- web.xml

Tak przygotowaną strukturę katalogów i plików umieszczamy w archiwum HelloTomcat.war i wgrywamy na serwer Tomcat.

Uruchomienie przykładu: http://localhost:8080/HelloTomcat/firstUrl.



Pierwszy serwlet – obsługa żądań

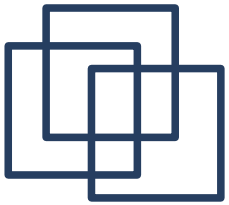
Żądania, przekazywane do serwletu, w zależności od formy są obsługiwane przez jedną z dwóch metod:

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

oraz

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                        throws ServletException, IOException
```

Z tego powodu serwlet chcący reagować na oba rodzaje wywołań powinien obsługiwać obie metody.



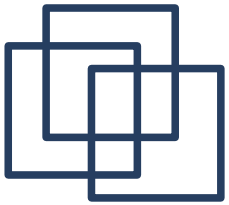
Pierwszy serwlet – obsługa żądań

Dostęp do danych przekazanych w żądaniu (argumenty GET, POST, ustawienia Cookies, identyfikatory sesji) uzyskujemy za pośrednictwem obiektu `HttpServletRequest`.

Obiekt `HttpServletResponse response` jest przez serwlet używany do skonstruowania odpowiedzi.

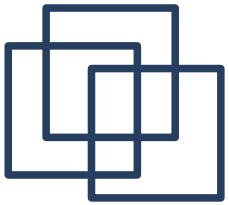
Pierwszy serwlet - obsługa żądań

```
public void doGet(...)throws ... {
    performTask(request, response);
}
public void doPost(...) throws ... {
    performTask(request, response);
}
public void performTask(HttpServletRequest request,
                        HttpServletResponse response) {
    HttpSession session = request.getSession();
    Object obj = session.getAttribute("lp");
    if (obj==null)
        obj = "0";
    session.setAttribute("lp",
         String.valueOf(Integer.parseInt(obj.toString()) + 1));
    response.setContentType("text/html");
    PrintWriter out;
    try {
        out = response.getWriter();
        out.println("Hello world: " + obj.toString());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



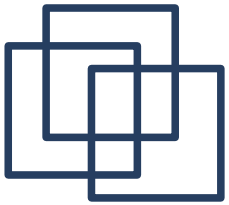
JavaServer Pages

Aby ułatwić programistom tworzenie stron internetowych wprowadzono technologię JSP, która przypomina inne, powszechnie stosowane języki skryptowe służące do generowania dokumentów HTML (ASP, PHP, ...). Dokument JSP jest zapisany w formacie XML.



Pierwszy skrypt JSP

```
<%@ page language = "java" %>
<%@ page import = "java.util.*" %>
<%@ page contentType = "text/html" %>
<html>
  <title>JSP Demo</title>
  <body bgcolor="white" text="red">
    <H1>
    <%
     if (Calendar.getInstance().get(Calendar.AM_PM)==Calendar.AM){
    %>
      Milego poranka!
    <% }else{ %>
      Milego popoludnia!
    <% } %>
    </H1>
    <H3>
     Aktualny czas: <% out.println(new java.util.Date()); %>
    </H3>
  </body>
</html>
```



Uruchomienie strony JSP

Skrypt umieszczamy np. w głównym katalogu aplikacji:

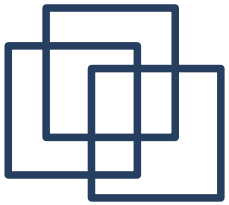
WEB-INF
    classes
    lib
    web.xml
hello.jsp

Dostęp do niego odbywa się poprzez adres:

http://localhost:8080/HelloTomcat/hello.jsp

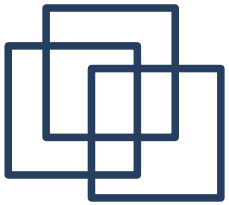
W rzeczywistości na podstawie skryptu jsp tworzona jest automatycznie obiekt Javy (przypominający servlet), który generuje opisaną w skrypcie dokument.

Aby nie przeplatać kodu Javy z elementami HTML'a używa się specjalnych tagów:



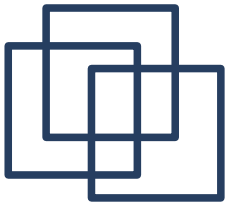
Przykład JSP

```
<%@ page contentType="text/html; charset=UTF-8" %>

<!-- import biblioteki StandardTagLibrary -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!-- import własnej biblioteki tagów -->
<%@ taglib uri="/WEB-INF/functions.tld" prefix="f" %>

<html>
<head>
  <title>Daty</title>
</head>
<body bgcolor="white">

<!-- użycie własnego obiektu w ramach strony -->
<jsp:useBean id="locales" scope="application" class="mypkg.MyLocales"/>
```



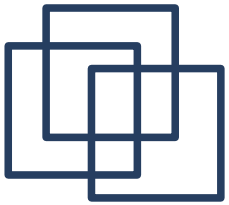
Przykład JSP

```
<form name="localeForm" action="date.jsp" method="post">
<c:set var="selectedLocaleString" value="${param.locale}" />
<c:set var="selectedFlag" value="${!empty selectedLocaleString}" />
<b>Lokalizacja:</b>
<select name=locale>
<c:forEach var="localeString" items="${locales.localeNames}" >
  <c:choose>
    <c:when test="${selectedFlag}">
       <c:choose>
           <c:when test="${f:equals(selectedLocaleString,localeString)}">
           <option selected>${localeString}</option>
           </c:when>
           <c:otherwise>
              <option>${localeString}</option>
           </c:otherwise>
       </c:choose>
    </c:when>
    <c:otherwise>
       <option>${localeString}</option>
    </c:otherwise>
  </c:choose>
</c:forEach>
</select>
```



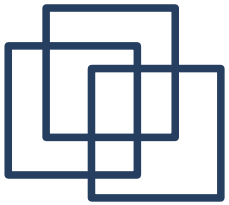
Przykład JSP

```
<input type="submit" name="Submit" value="Pokaz date">
</form>

<c:if test="${selectedFlag}" >
  <jsp:setProperty name="locales"
                   property="selectedLocaleString"
                   value="${selectedLocaleString}" />
  <jsp:useBean id="date" class="mypkg.MyDate"/>
  <jsp:setProperty name="date"
                   property="locale"
                   value="${locales.selectedLocale}"/>
  <b>Data: </b>${date.date}
</c:if>

</body>
</html>
```



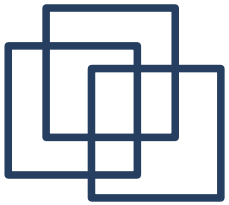
Własne funkcje

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        web-jsptaglibrary_2_0.xsd"
                        version="2.0">

  <description>Definicja funkcji</description>
  <tlib-version>1.0</tlib-version>
  <short-name>FunctionTagLibrary</short-name>
  <uri>/FunctionLibrary</uri>
  <function>
    <name>equals</name>
    <function-class>mypkg.MyLocales</function-class>
    <function-signature>
       boolean equals( java.lang.String, java.lang.String )
    </function-signature>
  </function>
</taglib>
```



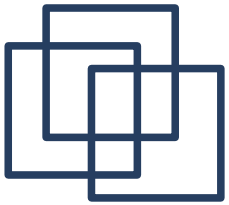
Bean MyLocales

```
package mypkg;

import java.util.*;
import java.text.DateFormat;
public class MyLocales {
    ArrayList<String> localeNames;
    HashMap<String, Locale> locales;
    Locale selectedLocale;
    String selectedLocaleString;

    public MyLocales() {
        locales = new HashMap<String, Locale>();
        localeNames = new ArrayList<String>();
        Locale[] list = DateFormat.getAvailableLocales();
        for (int i = 0; i < list.length; i++) {
            locales.put(list[i].getDisplayName(),
                        list[i]);
            localeNames.add(list[i].getDisplayName());
        }
        Collections.sort(localeNames);
        selectedLocale = null; selectedLocaleString = null;
    }
```



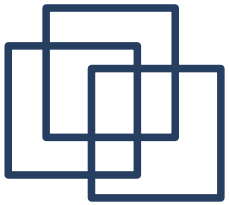
Bean MyLocales

```
    public static boolean equals(
        String l1,
        String l2) {
        return l1.equals(l2);
    }

    public Collection getLocaleNames() {
        return localeNames;
    }

    public void setSelectedLocaleString(String displayName) {
        this.selectedLocaleString = displayName;
        this.selectedLocale = (Locale) locales.get(displayName);
    }

    public Locale getSelectedLocale() {
        return selectedLocale;
    }

    public String getSelectedLocaleString() {
        return selectedLocaleString;
    }
}
```



Bean MyDate

```
package mypkg;

import java.text.DateFormat;
import java.util.*;


public class MyDate {
    Date today;
    DateFormat dateFormatter;

    public MyDate() {
        today = new Date();
    }

    public String getDate() {
        return dateFormatter.format(today);
    }

    public void setLocale(Locale l) {
        dateFormatter = DateFormat.getDateInstance(DateFormat.FULL, l);
    }
}
```



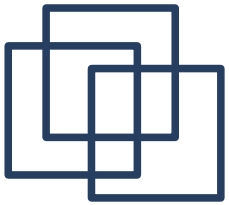
Struktura przykładu

Skrypt umieszczamy np. w głównym katalogu aplikacji:

WEB-INF
- **classes**
  - **mypkg**
    - MyDate.class
    - MyLocale.class
- **lib**
  - jstl.jar
  - standard.jar
- web.xml
- functions.tld

date.jsp

Adres: http://localhost:8080/HelloTomcat/date.jsp



Podsumowanie

Podstawową zaletą technologii servletów jest możliwość tworzenia serwisów internetowych z wykorzystaniem całej infrastruktury dostarczanej przez język programowania Java wraz z wszelkimi jego rozszerzeniami. Głównym obszarem zastosowaniem tego rodzaju rozwiązań są aplikacje webowe wymagające dużej skalowalności ze względu na używane zasoby. Kontener serwletów jest wtedy składnikiem odpowiedzialnym za interakcje z użytkownikiem poprzez interfejs www.