

## Wykłady Platforma Java EE.

### Zaawansowane programowanie w Javie

Dr inż. Damian Raczyński

## Java EE podstawy

### Serwlety

Serwlet w javie jest klasą, która może przyjmować żądania i generować odpowiedzi.

W praktyce zdecydowaną większość serwletów stanowią te wykorzystujące HTTP.

3

## Java EE podstawy

- Java Enterprise Edition – Java EE – JEE (dawniej J2EE)
- Technologia złożona (bardzo wiele elementów składowych)
- Apache Tomcat – serwer obsługuje JEE (przykładowo).
- Komponenty Java EE:
  - Aplety są komponentami działającymi na kliencie
  - Serwlety, JSP, EJB są komponentami działającymi na serwerze

2

## Java EE podstawy

JSP – rozszerzenie mechanizmu serwletów.

Podejście:

Mieszanie kodu HTML z kodem javy

4

## Java EE podstawy

JSTL i EL.

???

5

## Java EE podstawy

JSF – technologia prezentacji, pozwala na skoncentrowanie wysiłków projektantów na technologiach widoku.

7

## Java EE podstawy

JPA – technologia ta zapewnia uniwersalny dostęp do źródeł danych bez względu na faktycznie stosowaną technologię bazodanową.

Hibernate - framework ORM

6

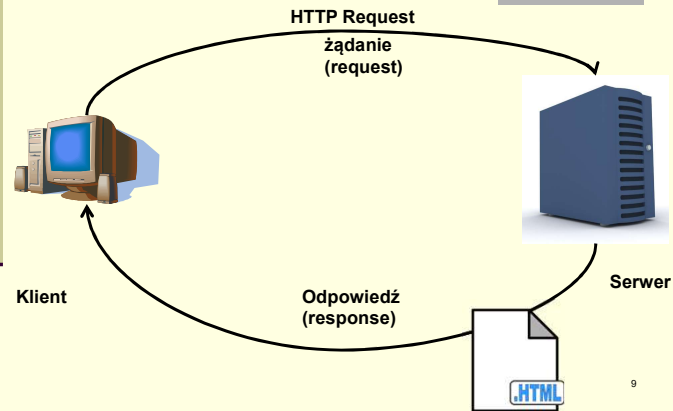
## Java EE podstawy

SPRING – FRAMEWORK dla javy

Pojęcie MVC

8

## Model żądanie-odpowieź



## Model żądanie-odpowieź

Logowanie na serwer e-mail:



11

## żądanie i odpowiedź

### HTTP Request

Kluczowe elementy żądania:

- Metoda HTTP (wywoływana akcja np. get, post ...)
- Adres strony (URL)
- Parametry formularza

### HTTP Response

Kluczowe elementy odpowiedzi:

- status code (czy żądanie zakończyło się sukcesem)
- Content-type (tekst, obrazek, html, ...)
- Zawartość (content)

10

## Model żądanie-odpowieź

Logowanie na serwer e-mail:  
1. [www.yahoo.com](http://www.yahoo.com)



Web Server  
Potrafi obsługiwać  
wyłącznie statyczne  
strony HTML

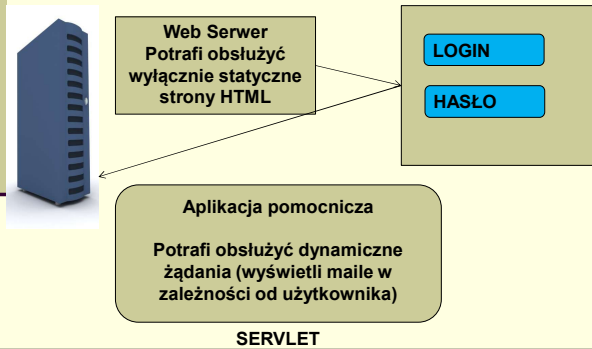
LOGIN

HASŁO

12

## Model żądanie-odpowieź

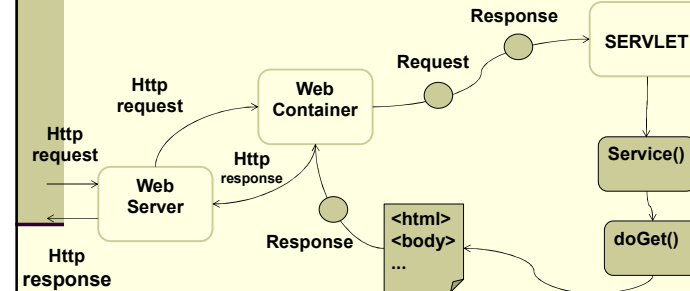
Logowanie na serwer e-mail:  
1. www.yahoo.com



13

## Kontener

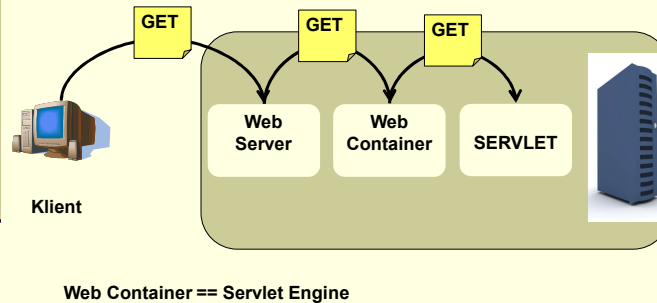
Jak kontener obsługuje żądanie



15

## Kontener

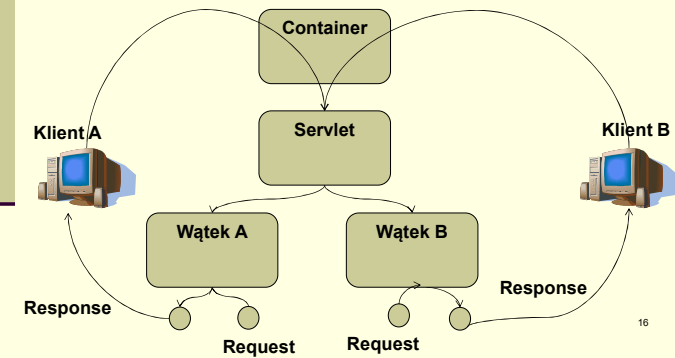
Czym jest kontener (Container)?



14

## Serwlety






Kontener uruchamia wiele wątków do przetwarzania wielu żądań dla pojedynczego serwletu



16

## Kontener

Za co odpowiedzialny jest kontener?

- Wspiera komunikację (klient-servlet), 
- Zarządza cyklem życia servletów, 
- Wspiera wielowątkowość (dla każdego żądania kontener powołuje wątek), 
- Bezpieczeństwo (tylko ważne żądania przejdą do servletów – walidacja przez kontener), 
- Wspiera JSP. 

## Java EE podstawy

Deskryptor wdrożenia (deployment descriptor) – plik XML stanowiący centralny punkt aplikacji webowej.

- Zmiana w tym pliku nie wymaga rekompilacji plików źródłowych,
- Plik ten oddziałuje na wszystkie elementy projektu.

19

## Kontener

Skąd kontener wie, który servlet chce klient?

Servlet może mieć

3 nazwy:

- Klient zna nazwę URL,
- Deployer zna ukrytą nazwę wewnętrzną,
- Nazwa pliku servletu

```
<web-app>
...
<servlet>
  <servlet-name>LoginServ</servlet-name>
  <servlet-class>com.Login</servlet-class>
</servlet>

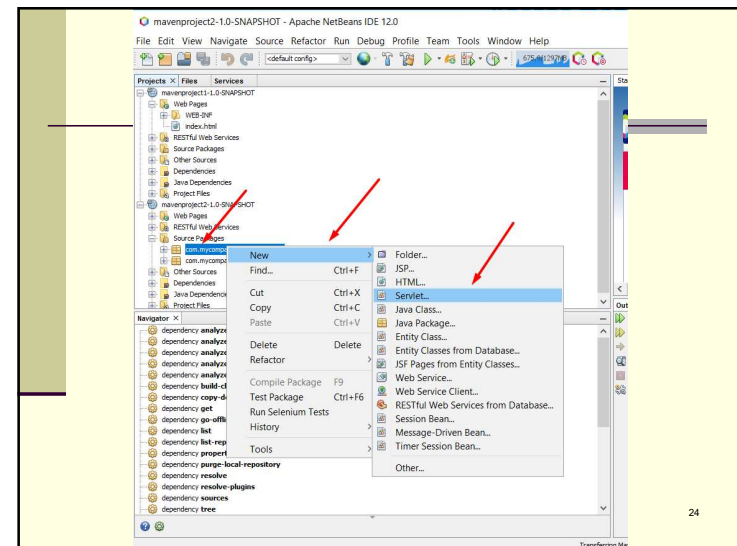
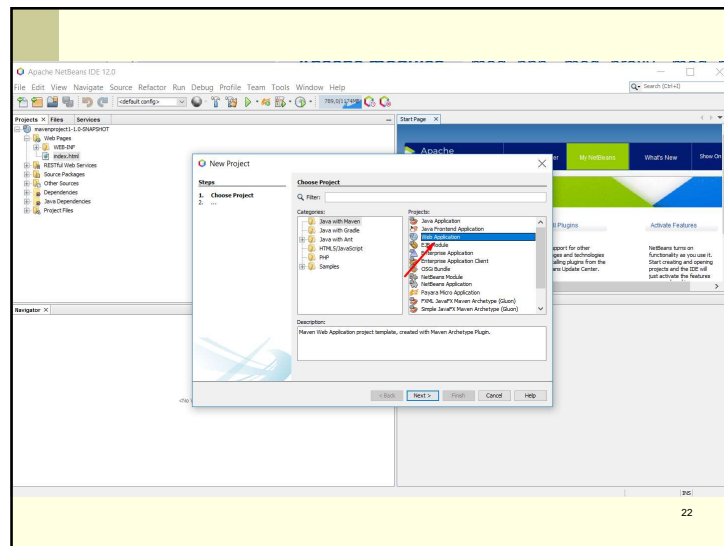
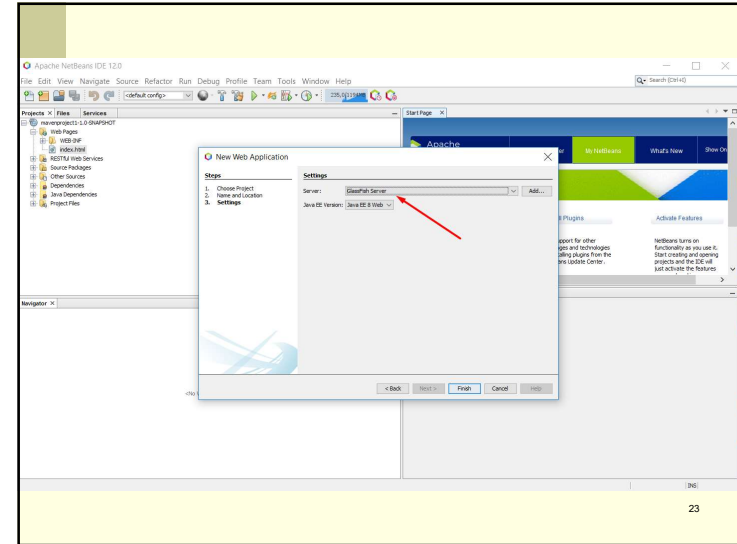
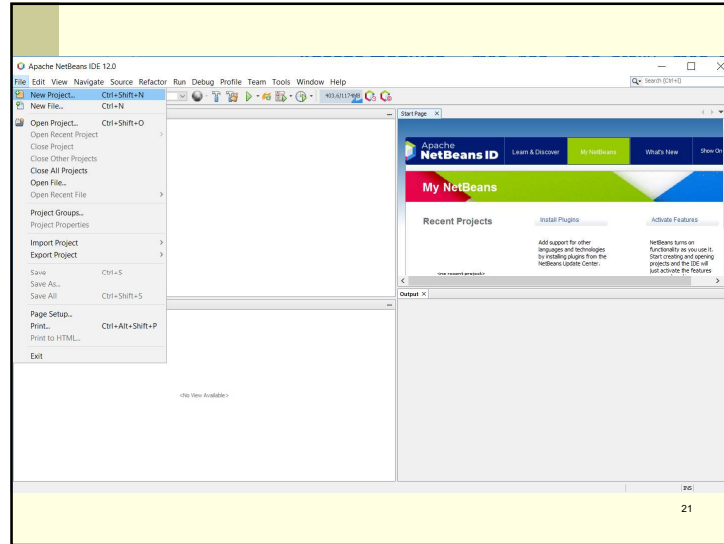
<servlet-mapping>
  <servlet-name>LoginServ</servlet-name>
  <url-pattern>/Logon</url-pattern>
</servlet-mapping>
...
</web-app>
```

## Java EE podstawy

...

Przejdźmy  
do praktyki

20



**New Servlet**

**Steps**

1. Choose File Type
2. Name and Location
3. Configure Servlet Deployment

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

< Back Next > Finish Cancel Help

25

## Prosty serwlet 1

```
<@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"%>
    <title>JSP Page</title>
  </head>
  <body>
    <form action="plik.do">
      <input type="text" name="pole1"/>
      <input type="submit" value="OK"/>
    </form>
  </body>
</html>
```

27

**New Servlet**

**Steps**

1. Choose File Type
2. Name and Location
3. Configure Servlet Deployment

**Configure Servlet Deployment**

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

URL Pattern(s):

Initialization Parameters:

Name	Value
------	-------

New Edit... Delete

< Back Next > Finish Cancel Help

26

## Prosty serwlet 1

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        String pobrane = request.getParameter("pole1");
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet pierwszy</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Wpisane " + pobrane + "</h1>");
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}
```

28

## Serwlety

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    String pobrane = request.getParameter("pole1");
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet pierwszy</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Wpisałeś " + pobrane + "</h1>");
    out.println("</body>");
    out.println("</html>");
}
```

29

## Serwlety

Następnie pobieramy obiekt zapisujący klasy `PrintWriter`:

```
PrintWriter out = response.getWriter();
```

Za jego pomocą zapisujemy treść, która zostanie przesłana do klienta.

Serwlet wykorzystuje dostarczony obiekt klasy `HttpServletRequest`, aby pobrać parametr przesłany w formularzu:

31

## Serwlety

Po wysłaniu formularza ze strony `index.jsp` następuje wywołanie serwletu – dokładnie metoda `processRequest()`.

Na początku określamy typ MIME nagłówka – rodzaj treści, jaką serwer przesyła do klienta (np. obrazek, film, dźwięk, tekst ...):

```
response.setContentType("text/html;charset=UTF-8");
```

30

## Serwlety

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
```

```
...
```

```
String pobrane = request.getParameter("pole1");
```

Następnie przekazywany jest do strumienia:

```
out.println("<h1>Wpisałeś " + pobrane + "</h1>");
```

32



## Serwlety

W jaki sposób serwer aplikacji skojarzył adres URL serwletu (plik.do) z serwletem pierwszy. Odpowiedź jest zawarta w deskrytorze wdrożenia, znajdującym się w pliku /WEB-INF/web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  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>pierwszy</servlet-name>
    <servlet-class>pl.nysa.pwsz.pierwszy</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>pierwszy</servlet-name>
    <url-pattern>plik.do</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

33

## Pokaz prostych możliwości serwletów

```
protected void processRequest
(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        for(int i=1; i<=10; i++)
            out.println(i+"<br/>");
    } finally {
        out.close();
    }
}
```

## Pokaz prostych możliwości serwletów

Przykłady serwletów – co zrobią ????

34

Formularz na stronie startowej:

```
<form action="jeden">
    <input name="pole1"/><input type="submit"/>
</form>
```

Fragment web.xml:

```
<servlet>
    <servlet-name>jeden</servlet-name>
    <servlet-class>pl.nysa.pwsz.jeden</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>jeden</servlet-name>
    <url-pattern>/jeden</url-pattern>
</servlet-mapping>
```

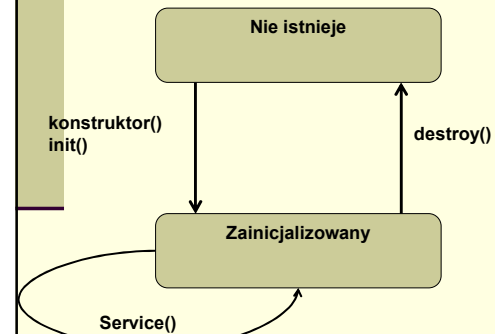
```
protected void processRequest(...) {
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        int koniec=Integer.parseInt(request.getParameter("pole1"));
        for(int i=1; i<=koniec; i++) out.println(i+"<br/>");
    } finally {
        out.close();
    }
}
```

## Pokaz prostych możliwości serwletów

```
int zagadka(int n)
{
    return (n==0)?1:n*zagadka(n-1);
}
protected void processRequest(
    HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        int x=Integer.parseInt(request.getParameter("pole1"));
        int s=zagadka(x);
        out.println("Wynik to "+s);
    } finally {
        out.close();
    }
}
```

## Serwlety

Cykl życia serwletu:



Cykl życia serwletu:

1. Użytkownik wysyła żądanie na serwer
2. Serwer ładuje klasę serwletu
3. Serwer tworzy instancję serwletu
4. Wywoływana jest metoda init()
5. Wywoływana jest metoda service()
6. Wywoływana jest odpowiednia metoda serwletu

39

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        int x=Integer.parseInt(request.getParameter("pole1"));
        out.println("<table style='border:1px solid black;'>");
        for(int i=0; i<x; i++)
        {
            out.println("<tr>");
            for(int j=0; j<x; j++)
            {
                out.println("<td>"+i*j+"</td>");
            }
            out.println("</tr>");
        }
        out.println("</table>");
    } finally {
        out.close();
    }
}
```

## Serwlety

Interface

Klasa abstrakcyjna

Klasa abstrakcyjna

Konkretna klasa

**Servlet**

**GenericServlet**

**HttpServlet**

**MyServlet**

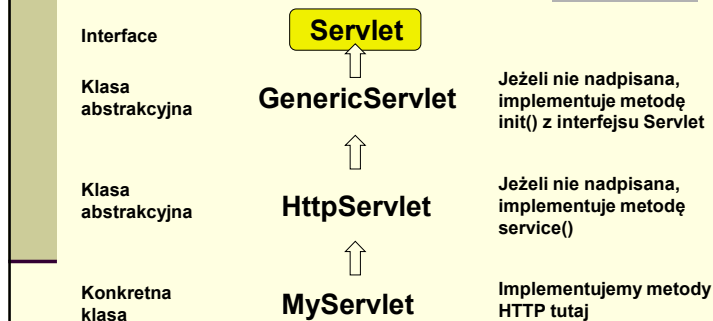
Jeżeli nie nadpisana, implementuje metodę init() z interfejsu Servlet

Jeżeli nie nadpisana, implementuje metodę service()

Implementujemy metody HTTP tutaj

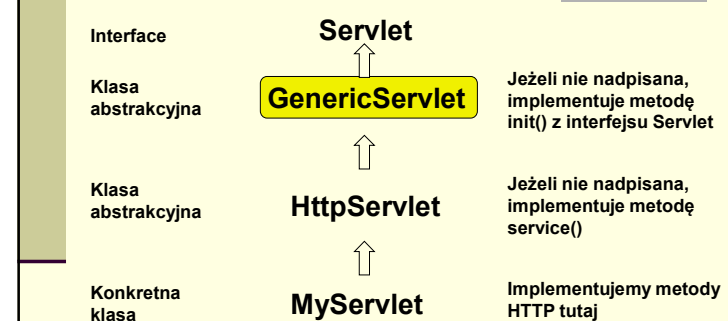
40

## Serwlety



41

## Serwlety



43

## Interfejs Servlet

Dostarcza ogólnego sposobu zachowania dla wszystkich serwletów.

Musi zostać zaimplementowany przy tworzeniu każdego serwletu.

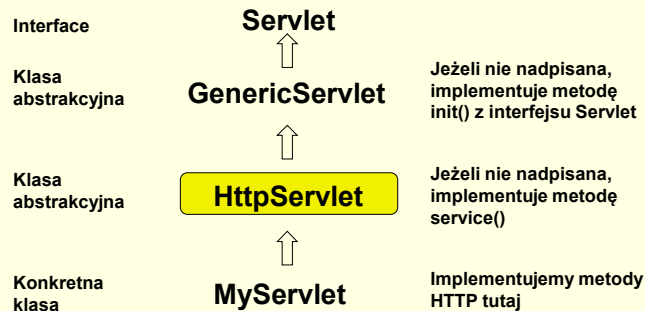
Metoda	Opis
<code>public void init(ServletConfig config)</code>	Inicjalizuje servlet. Wywoływana przez kontener tylko raz (metoda cyklu życia).
<code>public void service(ServletRequest request, ServletResponse response)</code>	Generuje odpowiedź na pojawiające się żądanie. Jest wywoływana przy każdym żądaniu przez web kontener.
<code>public void destroy()</code>	Jest wywoływana tylko raz. Oznacza, że servlet jest niszczone.
<code>public ServletConfig getServletConfig()</code>	zwraca obiekt ServletConfig.
<code>public String getServletInfo()</code>	Zwraca informacje o serwlecie takie jak autor, typ licencji, wersja ...

## GenericServlet

Klasa `GenericServlet` implementuje interfejsy `Servlet`, `ServletConfig` i `Serializable`. Klasa `GenericServlet` może obsługiwać dowolny typ żądania – jest niezależna od protokołu

<code>void</code>	<code>destroy()</code> - Wywoływana przez kontener przed usunięciem serwletu.
<code>String</code>	<code>getInitParameter(String name)</code>
<code>Enumeration</code>	<code>getInitParameterNames()</code>
<code>ServletConfig</code>	<code>getServletConfig()</code> - Zwraca obiekt ServletConfig.
<code>ServletContext</code>	<code>getServletContext()</code> - Zwraca referencję do ServletContext, w którym serwlet działa.
<code>String</code>	<code>getServletInfo()</code> - Informacje o serwlecie.
<code>String</code>	<code>getServletName()</code>
<code>void</code>	<code>init()</code>
<code>void</code>	<code>init(ServletConfig config)</code>
<code>void</code>	<code>log(String msg)</code> - Zapisuje określoną wiadomość do pliku logów serwletów, poprzedzonego nazwą serwletu.
<code>void</code>	<code>log(String message, Throwable t)</code>
<code>abstract void</code>	<code>service(ServletRequest req, ServletResponse res)</code> - Wywoływana przez kontener w celu wygenerowania odpowiedzi na żądanie.

## Serwlety



45

## ServletConfig

Obiekt klasy `ServletConfig` jest tworzony przez kontener web dla każdego serwletu. Obiekt ten jest wykorzystywany przez kontener do przekazania informacji podczas inicjalizacji serwletu (informacje z `web.xml`).

Jeżeli plik `web.xml` ulegnie zmianie, nie musimy na nowo kompilować serwletu.

Jak uzyskać obiekt `ServletConfig`:

1. `getServletConfig()` – metoda interfejsu `Servlet`
2. `init(ServletConfig config)` – przy inicjalizacji

47

## Serwlety

Serwlety (omawiana na wykładzie) dziedziczą po klasie `HttpServlet`.

Interfejs `Servlet` – określa metody, które muszą implementować wszystkie serwlety (nie koniecznie HTTP). Dotyczą one głównie cyklu życia serwletu (`init`, `service`, `destroy`).

Abstrakcyjna klasa `GenericServlet` – Podstawowa implementacja interfejsów `Servlet` i `ServletConfig` (daje dostęp do ustawień i parametrów serwletu).

`HttpServlet` udostępnia metody `do*` (`doGet`, `doPost` ...).

46

## ServletConfig

Interfejs `ServletConfig` udostępnia metody:

Interfejs ServletConfig	
<code>java.lang.String</code>	<code>getInitParameter (java.lang.String name)</code> Zwraca <code>String</code> zawierający wartość parametru określonego przez <code>name</code> lub <code>null</code> w przypadku, gdy parametr nie istnieje.
<code>java.util.Enumeration</code>	<code>getInitParameterNames ()</code> Zwraca nazwy parametrów inicjalizacyjnych serwletu.
<code>ServletContext</code>	<code>getServletContext ()</code> Zwraca referencję do obiektu <code>ServletContext</code> , w którym nastąpiło wywołanie.
<code>java.lang.String</code>	<code>getServletName ()</code> Zwraca nazwę Serwletu.

## Serwlety

### Parametry serwletów:

Parametryzacja kodu w pliku **web.xml** (deskryptor wdrożenia) powoduje, że w przypadku potrzeby zmiany pewnych wartości dla aplikacji web'owej, nie trzeba jej na nowo kompilować serwletów.

Parametry serwletów można określić za pomocą znacznika **<init-param>** w następujący sposób:

```
<init-param>
  <param-name> nazwa</param-name>
  <param-value>wartość</param-value>
</init-param>
```

Można dodawać dowolnie dużo **init-param**

49

## Serwlety

METODY	kiedy wywoływana	Po co	Czy można nadpisać
<b>init()</b> (metoda ma dostęp do obiektów <b>Context</b> i <b>Config</b> w przeciwieństwie do konstruktora)	Kontener wywołuje metodę zanim serwlet może obsłużyć jakiegokolwiek żądanie	Aby zainicjalizować serwlet przed obsługą żądań użytkownika	TAK
<b>service()</b>	Kiedy przychodzi nowe żądanie do serwletu	Aby określić którą metodę HTTP wywołać	Nie powinno się
<b>doGet()</b> lub <b>doPost()</b>	Metoda <b>service()</b> wywołuje jedną z metod, w zależności od metody HTTP z żądania	Aby obsłużyć logikę biznesową	Zawsze

```
<servlet>
  <servlet-name>ServInit</servlet-name>
  <servlet-class>pl.pwsz.ServInit</servlet-class>
  <init-param>
    <param-name>user</param-name>
    <param-value>Ed</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
    <param-value>xxxx</param-value>
  </init-param>
</servlet>
```

```
18-Oct-2015 21:26:20.622 INFO [main] org.apache.coyote.AbstractProtocol.start Starting Pr
18-Oct-2015 21:26:20.634 INFO [main] org.apache.catalina.startup.Catalina.start Server st
18-Oct-2015 21:26:24.148 INFO [http-nio-8084-exec-5] org.apache.catalina.startup.HostConf
18-Oct-2015 21:26:24.320 INFO [http-nio-8084-exec-11] org.apache.catalina.startup.HostConf
18-Oct-2015 21:26:24.324 WARNING [http-nio-8084-exec-11] org.apache.catalina.startup.SetC
18-Oct-2015 21:26:24.529 INFO [http-nio-8084-exec-11] org.apache.jasper.servlet.TldScanne
18-Oct-2015 21:26:24.536 INFO [http-nio-8084-exec-11] org.apache.catalina.startup.HostCon
paramName: password, warto?: xxxx
paramName: user, warto?: Ed
    String paramName=paramNames.nextElement();
    String value=servletConfig.getInitParameter(paramName);
    System.out.println("paramName: "+paramName+", wartość: "+value);
  }
}

public void destroy() {
    System.out.println("KONIEC ŻYCIA SERWLETU");
}
```

## Serwlety

Protokół HTTP definiuje szereg metod: GET, POST, PUT, OPTIONS, TRACE, DELETE, CONNECT.

W praktyce wykorzystywane są najczęściej **GET** i **POST**.

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Handles the HTTP <code>POST</code> method.
 *
 * @param request HttpServletRequest request
 * @param response HttpServletResponse response
 * @throws ServletException if a Servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
```

52

## Serwlety

HTTP request określa, czy wywołać doGet() czy doPost():

	GET (doGet())	POST (doPost())
HTTP Request	żądanie zawiera wyłącznie linię żądania i nagłówki HTTP	dodatkowo zawiera HTTP body
Przekazywanie parametrów	Elementy formularza przekazywane są poprzez dodanie do końca URL	Elementy formularza są przekazane wewnątrz HTTP body
Wielkość	ograniczona (dane ograniczone w zależności od kontenera)	Możliwe jest przesyłanie wielkiej ilości danych
Użycie	Zazwyczaj w celu załadowania pewnych informacji z hosta	Zazwyczaj w celu przetwarzania przesłanych danych

## ServletContext

Obiekt utworzony przez Web Kontener w czasie „odpalania” projektu.



Istnieje wyłącznie jeden ServletContext na całą aplikację webową.



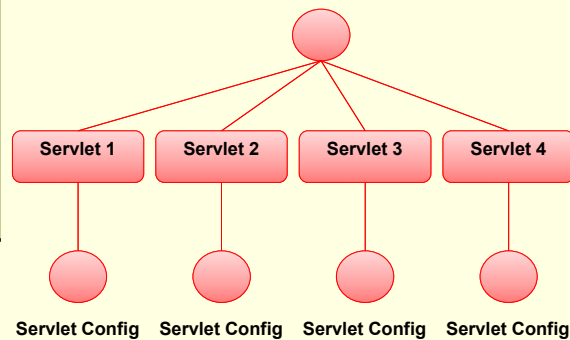
Obiekt ServletContext może zostać użyty do odczytu konfiguracji z pliku web.xml. Jeżeli jakaś informacja jest dzielona między servletami – najbardziej optymalnie dostarczyć ją w znacznikach <context-param> w web.xml

Różne aplikacje Webowe posiadają inny kontekst

55

## Serwlety

Servlet Context



54

## Serwlety

getServletContext() – kontekst serwletu.

Każdy parametr kontekstu jest widoczny we wszystkich serwletach.

```

<web-app>
  <context-param>
    <param-name>jeden</param-name>
    <param-value>1</param-value>
  </context-param>
  ...
</web-app>
  
```

56

## ServletContext

Interface ServletConfig zawiera:

ServletContext getServletContext () – metoda zwraca referencję do obiektu ServletContext, w którym element wywołujący metodę się wykonuje

GenericServlet:

ServletContext getServletContext() – zwraca referencję do obiektu ServletContext, w którym servlet działa

```
//Dostęp poprzez obiekt ServletConfig:
ServletContext app = getServletConfig().getServletContext();

//Dostęp bezpośredni (z serwetu):
ServletContext app=getServletContext();
```

## Serwlety

Atrybuty:

Główny sposób komunikacji między serwetami, kontenerem, sesją,

- W przypadku parametrów zarówno klucz jak i wartość była ciągiem znaków, dla atrybutów wartość może być obiektem,
- Parametry z założenia są tylko do odczytu (od wersji 3.0 nieco inaczej), atrybuty – zarówno do odczytu jak i zapisu,
- Atrybuty posiadają zasięg widoczności – atrybut w zasięgu danego żądania (request) nie będzie widoczny w innych zasięgach.

59

## Serwlety

Pobieranie parametru kontekstu:

```
this.getServletContext().getInitParameter („nazwa”);
```

Od wersji JavaServlet 3.0 API istnieje możliwość dynamicznego ustawiania parametrów kontekstu:

```
setInitParameter().
```

58

## Serwlety

Czym jest atrybut?

Różnica między atrybutem a parametrem

	Atrybuty	Parametry
Typy	Context, Request, Session	Context, Servlet
Metody ustawiania	setAttribute(String, Object)	nie można ustawić parametrów dynamicznie (wyłącznie raz w web.xml)
Zwracany typ	Object	String
Metoda do pobierania	getAttribute(String)	getInitParameter(String)

## Serwlety

Zakres	Parametry		Atrybuty	
	Zapis	Odczyt	Zapis	Odczyt
Żądanie	Nie	Tak	Tak	Tak
Serwlet	Nie	Tak	Brak	Brak
Sesja	Brak	Brak	Tak	Tak
Kontekst aplikacji	Tak (od wersji 3.0)	Tak	Tak	Tak

61

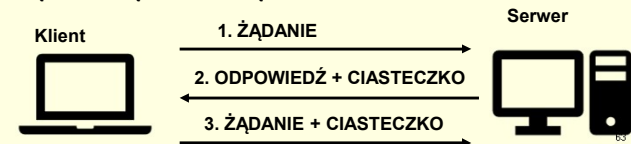
## Ciasteczka



Ciasteczko jest niewielką porcją informacji przekazywaną pomiędzy kolejnymi żądaniami klienta

**Posiada swoją nazwę, pojedynczą wartość, przypisaną domenę, maksymalny wiek i numer wersji.**

Każde żądanie HTTP obsługiwane jest jak nowe. Wysłanie do klienta ciasteczka w odpowiedzi spowoduje, że w kolejnych żądaniach będzie ono dołączane.



## Serwlety

Różnica między kontekstem servletów (Servlet Context) a konfiguracją parametrów początkowych (Init parameters)

	Parametry początkowe kontekstu	Parametry początkowe servletu
Zasięg	Zasięg wewnątrz kontenera WEB	Specyficzne dla servletu
kod	<code>getServletContext()</code>	<code>getServletConfig()</code>
Deskryptor wdrożenia	wewnątrz <code>&lt;web-app&gt;</code> ale nie wewnątrz <code>&lt;servlet&gt;</code>	wewnątrz określonego <code>&lt;servlet&gt;</code>

62

## Ciasteczka



2 rodzaje ciasteczek:

- tymczasowe - ważne w pojedynczej sesji. Jest usuwane za każdym razem, gdy użytkownik zamknie przeglądarkę.
- stałe - nie jest usuwane w przypadku, gdy klient zamknie przeglądarkę.

Klasa: `javax.servlet.http.Cookie` - wybrane metody:

Metoda	Opis
<code>void setMaxAge(int)</code>	Ustawia maksymalny okres istnienia ciasteczka (w s)
<code>String getName()</code>	Nazwa
<code>String getValue()</code>	Wartość
<code>void setName(String)</code>	Ustawia nazwę
<code>void setValue(String)</code>	Ustawia wartość
<code>Cookie(String nazwa, String wartość)</code>	Konstruktor



## Ciasteczka



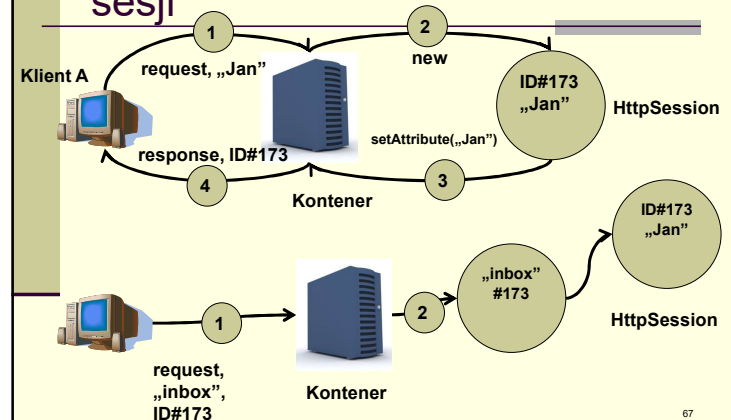
```
HttpServletResponse:
public void addCookie(Cookie c) - dodaje ciasteczko do
odpowiedzi,
HttpServletRequest:
public Cookie[] getCookies() - pobiera wszystkie
ciasteczka z żądania
```

Usuwanie ciasteczka z przeglądarki:

```
Cookie c = new Cookie("nazwa", "");
c.setMaxAge(0);
response.addCookie(c);
```

65

## Zarządzanie sesją – śledzenie sesji



67

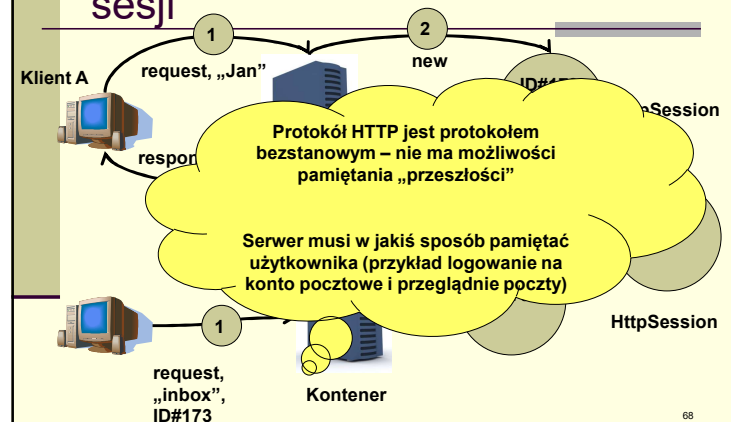
## Ciasteczka



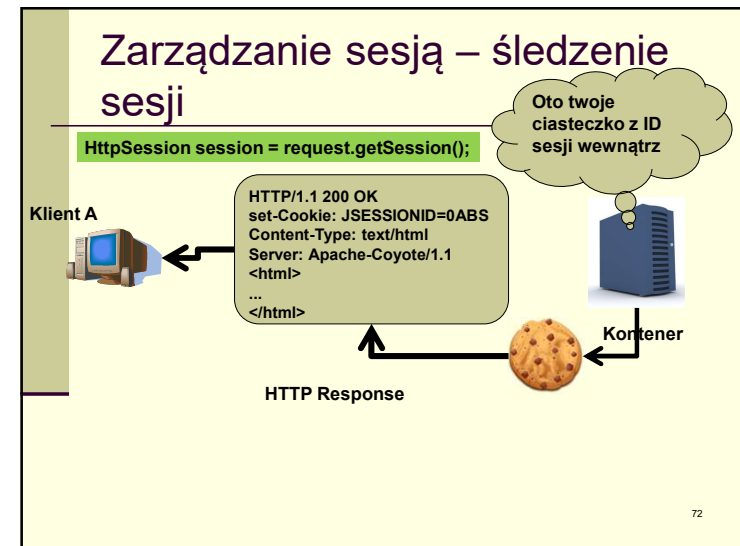
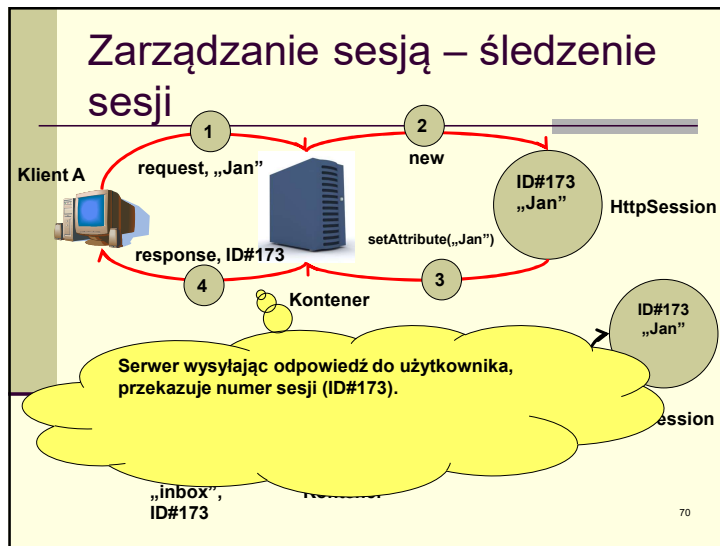
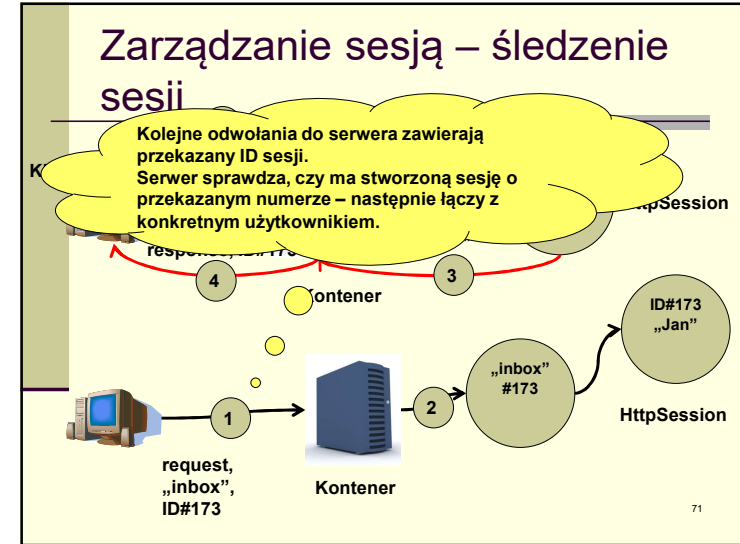
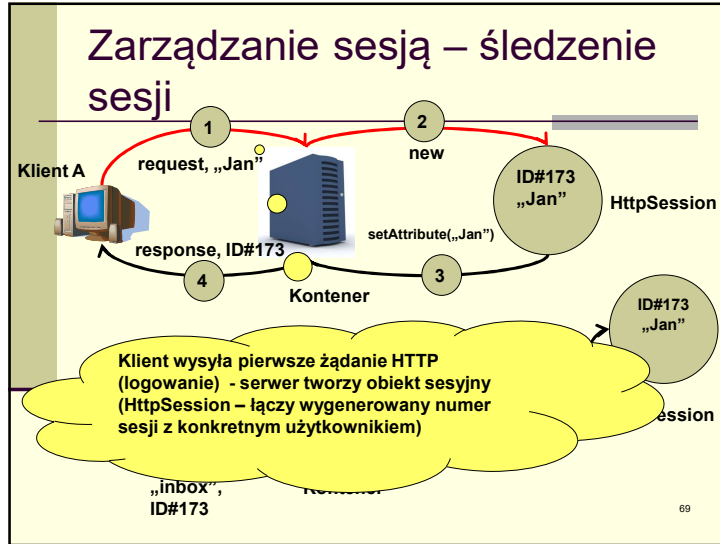
```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    Cookie OstatniaWizyta = null;
    for(Cookie c : request.getCookies())
        if(c.getName().equals("bylemtu"))
        {
            OstatniaWizyta=c; break;
        }
    if(OstatniaWizyta!= null)
        out.println("Ostatnie odwiedziny: "+OstatniaWizyta.getValue())
    else
        out.println("Pierwsza wizyta");
    OstatniaWizyta=new Cookie("bylemtu", new Date().toString());
    response.addCookie(OstatniaWizyta);
} finally {
    out.close();
}
```

66

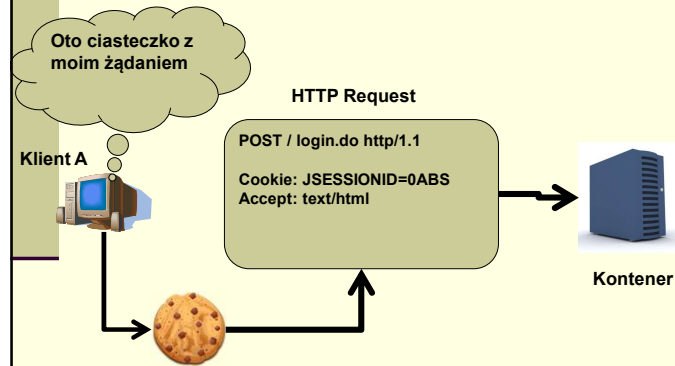
## Zarządzanie sesją – śledzenie sesji



68



## Zarządzanie sesją – śledzenie sesji



73

## Sesje

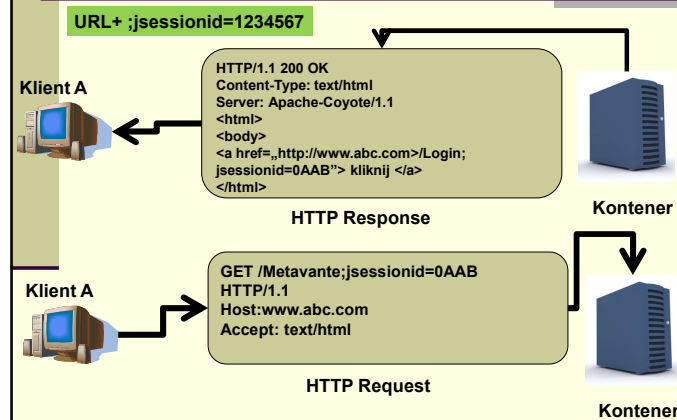
Servlet Container wykorzystuje interfejs `HttpSession` do utworzenia sesji pomiędzy klientem HTTP a serwerem HTTP

Sesja trwa przez określony czas niezależnie od liczby połączeń lub żądań ze strony użytkownika. Sesja zazwyczaj jest związana z jednym użytkownikiem, który może odwiedzić stronę wiele razy.

Serwer może utrzymywać sesję poprzez zastosowanie ciasteczek lub parametrów URL.

75

## Zarządzanie sesją – śledzenie sesji



74

## Sesje

Obiekt `HttpSession` może zostać wykorzystany w celu wykonania dwóch zadań:

- powiązania obiektów z sesją, pozwalając na zachowanie informacji o użytkowniku pomimo wielu połączeń

- Odczyt i manipulacja informacją na temat sesji - między innymi identyfikatorem, czasem utworzenia, czasem ostatniego odwołania

76

## Sesje

Jak uzyskać obiekt sesji

Interfejs `HttpServletRequest` dostarcza dwie metody:

`HttpSession getSession()`

`HttpSession getSession(boolean czyTworzyć)` – zwraca obiekt sesji lub tworzy nowy (dla `true`), zwraca `null` gdy sesji nie ma (dla `false`).

`getSession == getSession(true)`.

Metoda:

`boolean isNew()` – zwraca `true`, jeśli obiekt sesji został utworzony podczas tego żądania.

77

## Sesje

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
    if(request.getSession(false)==null)
    {
        out.println("Pierwsze odwiedziny");
        HttpSession a=request.getSession();
        if(a.isNew()==true)
            out.println("Otwarta pierwsza sesja");
    }
    else
    {
        out.println("Kolejne odwiedziny");
        request.getSession().invalidate();
    }
}
}
```

79

## Sesje

Najczęściej wykorzystywane metody interfejsu `HttpSession`:

### Metody

`long getCreationTime()` – Zwraca czas, kiedy sesja została stworzona (w milisekundach od 01.01.1970r)

`java.lang.String getId()` – Zwraca tekst zawierający unikalny identyfikator powiązany z sesją

`long getLastAccessedTime()` – zwraca ostatni czas, kiedy klient wysłał żądanie powiązane z sesją (w milisekundach od 1.1.1970)

`int getMaxInactiveInterval()` – zwraca maksymalny interwał czasowy pomiędzy kolejnymi operacjami użytkownika

`void invalidate()` – kończy sesję

## Sesje

Możliwe jest ustawienie atrybutu sesji w jednym serwlecie i odczytanie ustawionej wartości w zasięgu sesji w innym serwlecie.

Aby ustawić atrybut w zasięgu sesji wykorzystywana jest metoda `setAttribute()` interfejsu `HttpSession`.

Odczyt atrybutu – za pomocą metody `getAttribute`

80

## Sesje

Object `getAttribute(String nazwa)` – zwraca atrybut sesji o podanej nazwie (lub null),  
 void `setAttribute(String nazwa, Object wartość)` – dodaje obiekt do sesji (lub zastępuje istniejący).

Maksymalny czas bezczynności można ustawić w pliku web.xml:

```
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<welcome-file-list>
```

Czas wyrażony w minutach

Lub metodą:  
 void `setMaxInactiveInterval(int czas)` – czas w sekundach, dla wartości 0 sesja nigdy nie wygasa.

81

## Sesje

### Synchronizacja

Atrybuty żądania – żądanie jest realizowane przez jednego użytkownika, ponadto żądanie nie wiąże się z innymi żądaniami (nawet tego samego użytkownika) – **problem synchronizacji nie występuje** – mechanizm komunikacji z technologią JSP.

Atrybuty sesji – sesja jest powiązana z użytkownikiem (problem, gdy aplikacja zostanie uruchomiona w dwóch zakładkach i użytkownik jednocześnie uruchamia te same lub inne serwlety).  
**Należy wykorzystać synchronizację** – blokowany jest obiekt sesji, do którego uzyskuje się dostęp za pomocą obiektu żądania.

83

## Sesje

java.util Enumeration<java.lang.String>  
`getAttributeNames()` – zwraca kolekcję String'ów zawierającą atrybuty sesji.

void `removeAttribute(java.lang.String name)` – usuwa atrybut z sesji

```
HttpSession sesja= request.getSession();
if(sesja.getAttribute("ile")==null)
sesja.setAttribute("ile", 1);
else
{
  int licznik=Integer.parseInt(sesja.getAttribute("ile").toString());
  sesja.setAttribute("ile", licznik+1);
}
out.println(sesja.getAttribute("ile").toString());
```

## Sesje

### Synchronizacja:

```
HttpSession sesja=request.getSession();
synchronized(sesja){
  if(sesja.getAttribute("ile")==null)
    sesja.setAttribute("ile", 1);
  else
  {
    int licznik=Integer.parseInt(sesja.getAttribute("ile").toString());
    sesja.setAttribute("ile", licznik+1);
  }
}
out.println(sesja.getAttribute("ile").toString());
```

84

## Serwlety

Atrybuty kontekstu aplikacji – konieczna synchronizacja.

```
ServletContext x=this.getServletContext();
synchronized(x)
{
    ...
}
```

85

## Serwlety

Odczyt danych z formularza z wykorzystaniem serwletu

Metoda	Opis
Object getParameter(String nazwa)	Pobiera parametr określony nazwą z żądania
Enumeration<String> getParameterNames()	Pobiera nazwy wszystkich parametrów
Map getParameterMap()	Zwraca Mapę parametrów z żądania (java.util.Map)
String[] getParameterValues(String name)	Zwraca tablicę Stringów zawierającą wszystkie wartości związane z parametrem (lub null w przypadku, gdy parametr nie istnieje).

87

## HttpServlet

Metody do... .

86

```
Strona startowa:

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <form action="dane1" method="GET">
      Imię: <input type="text" name="imie"/>
      <br/>
      Nazwisko: <input type="text" name="nazwisko"/>
      <input type="submit" value="Wyślij"/>
    </form>
  </body>
</html>
```

```

serwlet dostępny pod adresem serwer/dane1 - metoda doGet:
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    ... // --> na następnych slajdach
}

```

```

let dostępny pod adresem serwer/dane1 - metoda doGet:

out.println("Sposób 2:<br/>");
Map<String,String[]> paramMap=request.getParameterMap();
Set<String> paramNamesSet= paramMap.keySet();
out.println("<ul>");
for(String paramName: paramNamesSet) {
    out.println("<li>"+paramName+":   ");
    String[] paramValue= paramMap.get(paramName);
    for(int i=0; i<paramValue.length; i++)
        out.println(paramValue[i]+"</li>");
}
out.println("</ul>");

```

```

serwlet dostępny pod adresem serwer/dane1 - metoda doGet:

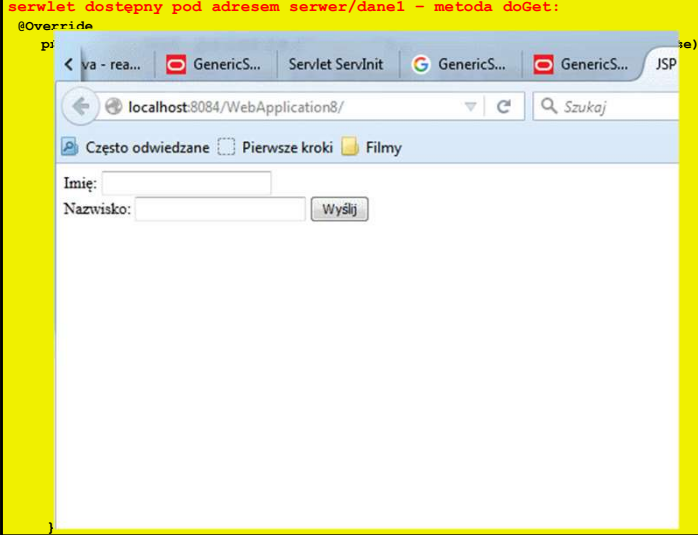
Enumeration<String> paramNames=request.getParameterNames();
out.println("Sposób 1:<br/>");
out.println("<ul>");
while(paramNames.hasMoreElements()) {
    String paramName= paramNames.nextElement();
    out.print("<li>"+paramName+":   ");
    String paramValue=request.getParameter(paramName);
    out.println(paramValue+"</li>");
}
out.println("</ul>");

```

```

serwlet dostępny pod adresem serwer/dane1 - metoda doGet:
@Override

```



The screenshot shows a web browser window with the address bar displaying 'localhost:8084/WebApplication8/'. The page contains a form with two input fields: 'Imię:' and 'Nazwisko:'. Below these fields is a button labeled 'Wyslij'. Above the form, there is a search bar with the text 'Szukaj' and a magnifying glass icon. The browser's tab bar shows several tabs, including 'GenericS...', 'Servlet ServInit', and 'GenericS...'. The page title is partially visible as 'JSP'.

Strona startowa:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <form action="dane2" method="GET">
      <input type="checkbox" name="ulubione" value="Java"/> Java
      <input type="checkbox" name="ulubione" value="PUM"/> PUM
      <input type="checkbox" name="ulubione" value="C#"/> C#
      <input type="checkbox" name="ulubione" value="JEE"/> JEE
      <input type="submit" value="Wybierz to co lubisz w zyciu"/>
    </form>
  </body>
</html>
```

Obsługa Wszystkich parametrów:

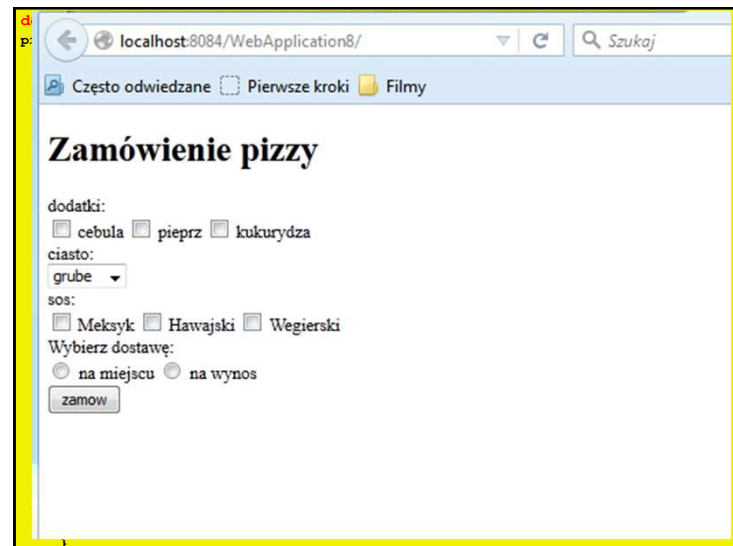
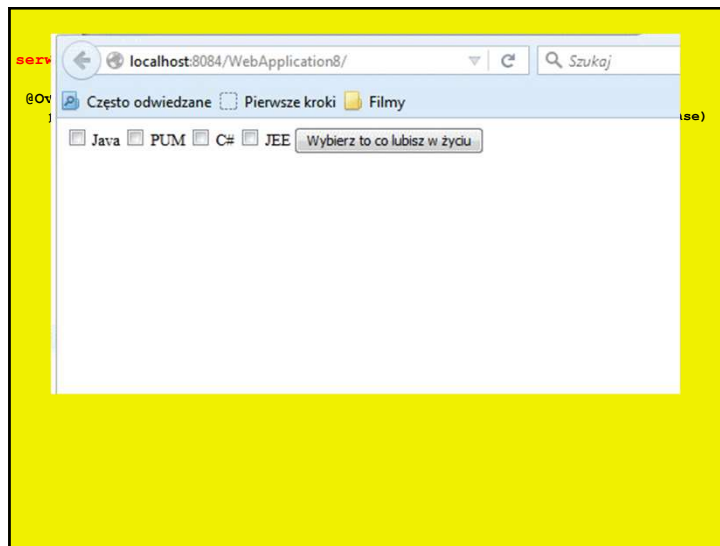
```
<h1>Zamówienie pizzy</h1>
<form action="dane3" method="GET">
  dodatki:<br/>
  <input type="checkbox" name="dodatek" value="cebula"/> cebula
  <input type="checkbox" name="dodatek" value="pieprz"/> pieprz
  <input type="checkbox" name="dodatek" value="kukurydza"/> kukurydza

  <br/>ciasto:<br/>
  <select name="ciasto">
    <option value="grube">grube</option>
    <option value="cienkie">cienkie</option>
  </select>

  <br/>sos:<br/>
  <input type="checkbox" name="sos" value="Meksyk"/> Meksyk
  <input type="checkbox" name="sos" value="Hawajski"/> Hawajski
  <input type="checkbox" name="sos" value="Wegierski"/> Wegierski

  <br/>
  Wybierz dostawę:
  <input type="radio" name="dostawa" value="tu"> na miejscu </input>
  <input type="radio" name="dostawa" value="tam"> na wynos </input>
  <br/>

  <input type="submit" value="zamow"/>
</form>
```





Zakładając tą samą stronę z formularzem - co zrobi serwet?:

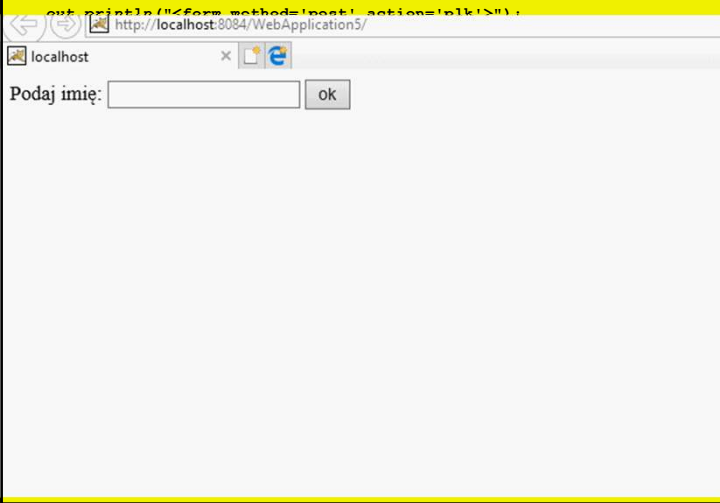
```
protected void doGet(HttpServletRequest request,...){
    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    out.println("<h1> zamówienie </h1>");
    ;
    {
        }
    out.println("</tr></table>");
}
```

To samo tylko wykorzystujemy mapę

```
out.println("</form>");
} else{
    out.println("<form method='post' action='plk'>");
    out.println("Podaj imię: <input type='text' name='imie'>");
    out.println("<input type='submit' value='ok' />");
    out.println("</form>");
}
}
```

```
Cookie imie = null;
if(request.getCookies()!=null)
for(Cookie c : request.getCookies()) if(c.getName().equals("imie"))
{
    imie=c; break;
}
if(imie!= null){
    HttpSession sesja=request.getSession();
    if(sesja.getAttribute("pkt")==null) sesja.setAttribute("pkt", 0);
    int pkt = (int) sesja.getAttribute("pkt");
    if(request.getParameter("wybor")!=null){
        int los=Math.abs((new Random()).nextInt())%2;
        out.println("Wylosowano: "+los+"<br>");
        int wybor=Integer.parseInt(request.getParameter("wybor"));
        if(los==wybor) pkt++;
        sesja.setAttribute("pkt", pkt);
    }
    out.println("Użytkownik: "+imie.getValue()+"<br>");
    out.println("Punkty: "+pkt+"<br>");
    out.println("<form method='post' action='plk'>");
    out.println("Wybierz 1 lub 0: <input type='text' name='wybor'>");
    out.println("<input type='submit' value='ok' />");
    out.println("</form>");
} else{
    if(request.getParameter("imie")!=null){
        Cookie cookie = new Cookie("imie", request.getParameter("imie"));
        response.addCookie(cookie);
        out.println("<form method='post' action='plk'>");
        out.println("<h1>Naciśnij OK by zagrać</h1><br>");
        out.println("<input type='submit' value='ok' />");
    }
}
```

```
out.println("</form>");
} else{
    out.println("<form method='post' action='plk'>");
    out.println("<h1>Naciśnij OK by zagrać</h1><br>");
    out.println("<input type='submit' value='ok' />");
}
```



Nagłówki żądania HTTP:

```
<h1>Zamówienie pizzy</h1>
<form action="Naglowek11" method="POST">
  dodatki:<br/>
  <input type="checkbox" name="dodatek" value="cebula" />
  <input type="checkbox" name="dodatek" value="pieprz" />
  <input type="checkbox" name="dodatek" value="kukurydza" />
  <br/>
  ciasto:
  <br/>
  <select name="ciasto">
  ...
```

**Analiza naglowkow**

Nazwa	wartosc
host	localhost:8084
user-agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	pl,en-US;q=0.7,en;q=0.3
accept-encoding	gzip, deflate
referer	http://localhost:8084/WebApplication8/
cookie	JSESSIONID=7EBA18450175CB3A10165361910192B9; csrfToken=8QYN0de1WbNsTmfqILrN5MayCP35yu3
connection	keep-alive
content-type	application/x-www-form-urlencoded
content-length	82

Nagłówki żądania HTTP:

#Override

**Analiza naglowkow**

Nazwa	wartosc
host	localhost:8084
user-agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:41.0) Gecko/20100101 Firefox/41.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	pl,en-US;q=0.7,en;q=0.3
accept-encoding	gzip, deflate
cookie	JSESSIONID=7EBA18450175CB3A10165361910192B9; csrfToken=8QYN0de1WbNsTmfqILrN5MayCP35yu3
connection	keep-alive

## Wysyłanie plików

W przypadku żądań typu multipart/form-data obiekt żądania udostępnia metody:

**Collection<Part> getParts()** - metoda zwraca kolekcję kolejnych części żądania (plików),

**Part getPart(String name)** - metoda zwraca część o określonej nazwie (powiązane z nazwą pola formularza).

Należy skonfigurować serwet obsługujący żądanie w pliku web.xml:

```
<servlet>
  <multipart-config>
    <max-file-size>xxx</max-file-size>
    <max-request-size>xxx</max-request-size>
  </multipart-config>
</servlet>
```

104

```

Plik web.xml:
<servlet>
<servlet-name>plik</servlet-name>
<servlet-class>wyklad.plik</servlet-class>
<multipart-config>
<max-file-size>10000000</max-file-size>
<max-request-size>50000000</max-request-size>
</multipart-config>
</servlet>
Plik index.jsp:
Wybierz plik do przesłania: <br />
<form action = "plik" method = "post" enctype = "multipart/form-data">
<input type = "file" name = "file" />
<br />
<input type = "submit" value = "Wyślij" />
</form>
Plik plik:
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
String appPath = request.getServletContext().getRealPath("");
request.getPart("file").write(appPath+"obrazek.jpg");
PrintWriter out = response.getWriter();
out.println("<img src=\"obrazek.jpg\">");
out.println(appPath);
}

```

## Wysyłanie plików

W celu załadowania większej liczby plików

Nazwy plików są przekazane w nagłówku content-disposition  
żądania w postaci:

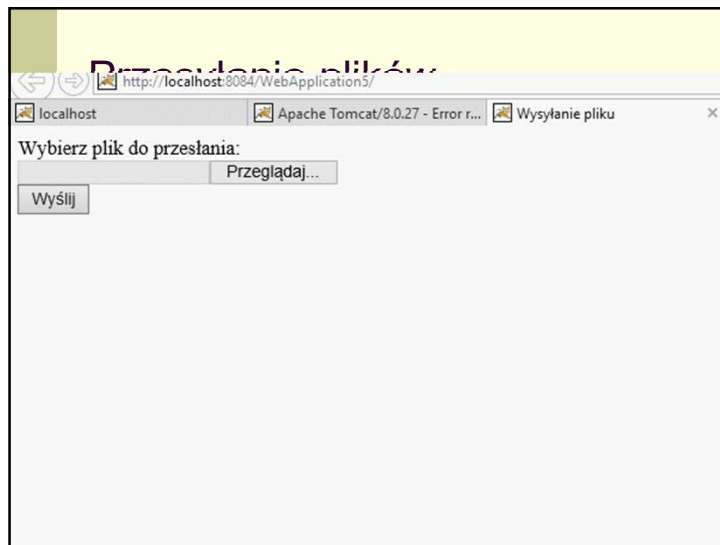
form-data; name="nazwa\_z\_formularza"; filename="plik.JPG"

```

private String nazwa(Part part) {
String dane = part.getHeader("content-disposition");
String[] elem = dane.split(";");
for (String s : elem) {
if (s.trim().startsWith("filename")) {
return s.substring(s.indexOf("=") + 2, s.length()-1);
}
}
return "";
}

```

107



```

<input type = "file" name = "file" multiple/>

protected void processRequest(. . .) . . . {
List<String> list = new ArrayList<>();
String appPath = request.getServletContext().getRealPath("");
for (Part part : request.getParts()) {
String nazwaPliku = nazwa(part);
nazwaPliku = new File(nazwaPliku).getName();
part.write(appPath + File.separator + nazwaPliku);
list.add(nazwaPliku);
}
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.println("<h2>Przesłane pliki:</h2>");
out.println("<ol>");
for (String x: list) out.println("<li>"+x+"</li>");
out.println("</ol>");
}

private String nazwa(Part part) {
String dane = part.getHeader("content-disposition");
String[] elem = dane.split(";");
for (String s : elem) {
if (s.trim().startsWith("filename")) {
return s.substring(s.indexOf("=") + 2, s.length()-1);
}
}
return "";
}

```

```
... <input type = "file" name = "file" multiple/>
protected void processRequest(. . .) . . . {
    List<String> list = new ArrayList<>();
    ;
}
pri
Str
String[] elem = dane.split(";");
for (String s : elem) {
    if (s.trim().startsWith("filename")) {
        return s.substring(s.indexOf("=") + 2, s.length()-1);
    }
}
return "";
```

## Przesłane pliki:

1. jee.JPG
2. poprawka1\_2017\_klucz.docx
3. poprawka1\_2017\_TEST.docx