



**University of Miskolc**  
**Faculty of Mechanical Engineering and Informatics**

**Java Web Application Development Technology**  
N13020008

---

## **Servlet technology**

**Tamás Tompa, PhD**  
assistant professor  
Department of Information Technology  
University of Miskolc



# What are Servlets?

---

- Servlets provide a **component-based, platform-independent method for building Web-based applications**
- Servlets **have access to the entire family of Java APIs**, including the JDBC API to access enterprise databases
- Using Servlets, you can **collect input from users through web page forms**, present records from a database or another source, and **create web pages dynamically**
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI)

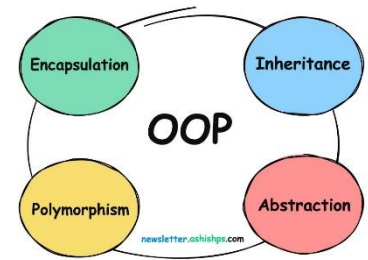


# What are Java Servlets?

---

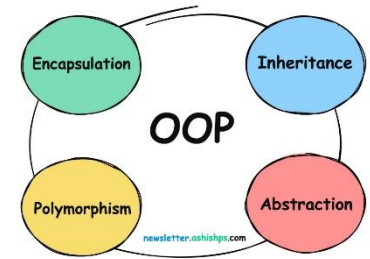
- **Java Servlets are**
  - **programs (classes)** that run on a Web or Application server
  - **act as a middle layer** between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server
- Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages
  - which are a standard part of the Java's enterprise edition
- **Tasks**
  - read the explicit data sent by the clients, read the implicit HTTP request data sent by the clients, process the data and generate the results, send the explicit data (i.e., the document), send the implicit HTTP response, and so on...

# OOP principles



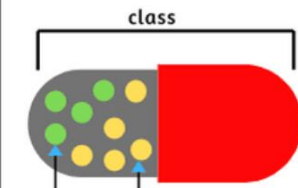
- Objects contain data, referred to as attributes or properties, and methods
- OOP allows objects to interact with each other using four basic principles:
  - **encapsulation**
    - data into a structured unit, along with the methods used to work with that data
  - **inheritance**
    - mechanism that allows a class to inherit properties and behaviors from another class
  - **abstraction**
    - used to hide unnecessary information and display only necessary information to the users interacting
  - **polymorphism**
    - allows a specific routine to use variables of different types at different times, gives a program the ability to redefine methods for derived classes

# OOP principles



```
class
{
    data members
    +
    methods (behavior)
}
```

ENCAPSULATION



Variables  
Methods

ENCAPSULATION

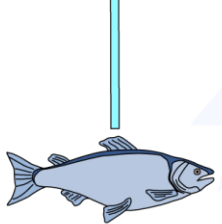
Animal



BASE CLASS  
CLASS: ANIMAL

METHOD

```
public procedure move()
print "I am going to move somehow"
```



DERIVED CLASS  
CLASS: FISH

METHODS  
OVERRIDE

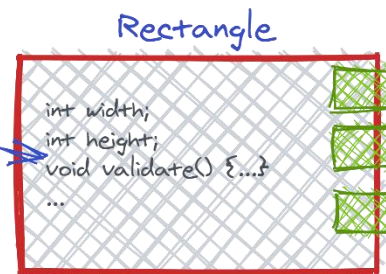
```
public procedure move()
print "I am a fish swimming"
```

DERIVED CLASS  
CLASS: BEAR

METHODS  
OVERRIDE

```
public procedure move()
print "I am a bear lumbering around"
```

Abstraction



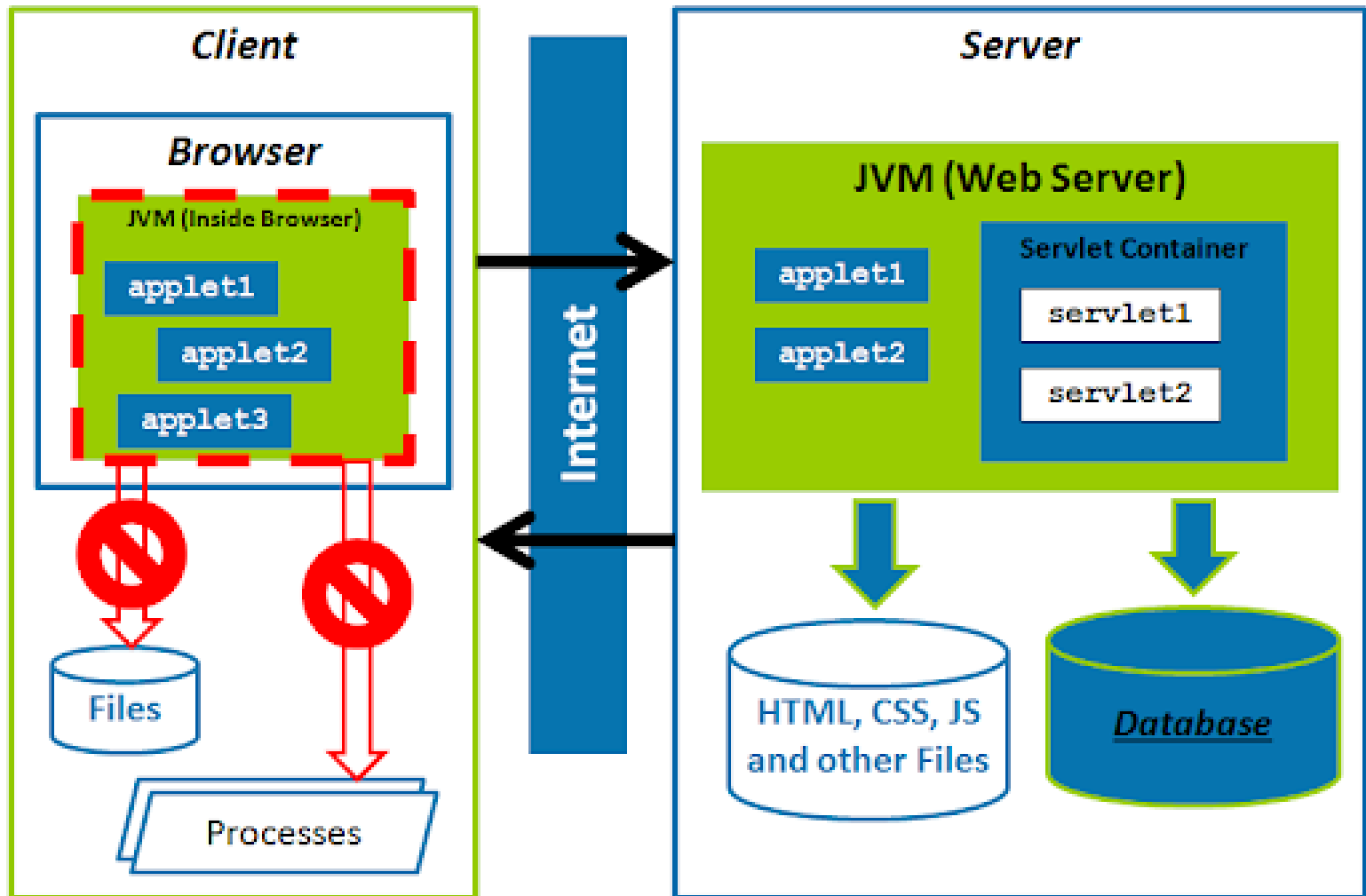
From outside we cannot see them

```
double getArea();
int getWidth();
int getHeight();
```

these are from outside we can only see



# Servlets Architecture



# Environment Setup

---

## ○ JDK (Java Development Kit)

- download an implementation of the Java Software Development Kit (SDK)
- setup PATH environment variable appropriately



## ○ Web Server – Tomcat



- download and install
- setup PATH environment variable appropriately
- these steps were introduced in the JSP course...

## ○ Eclipse IDE



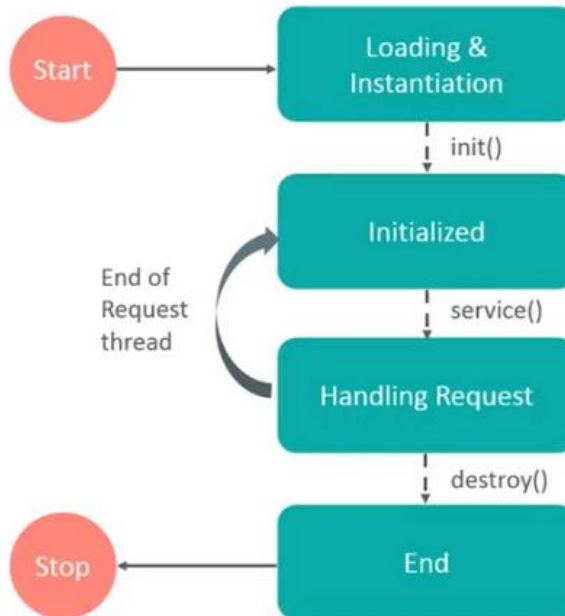
- <https://eclipseide.org/>
- Eclipse IDE for Java EE Developers
- [https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2024-09/R/eclipse-jee-2024-09-R-win32-x86\\_64.zip](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2024-09/R/eclipse-jee-2024-09-R-win32-x86_64.zip)

# Life Cycle

## ○ The life cycle of Servlets:

- The servlet is initialized by calling the **init()** method
- The servlet calls **service()** method to process a client's request
- The servlet is terminated by calling the **destroy()** method
- Finally, servlet is garbage collected by the garbage collector of the JVM

## Servlet Life Cycle







# Life Cycle

---

- **The life cycle of Servlets:**
  - The servlet is initialized by calling the **init()** method
    - called only once, when the servlet is created
    - servlet is normally created when a user first invokes a URL corresponding to the servlet
    - when a user invokes a servlet, a single instance of each servlet gets created
    - each user request resulting in a new thread that is handed off to doGet or doPost as appropriate

```
public void init() throws ServletException {  
    // Initialization code...  
}
```



# Life Cycle

---

- **The life cycle of Servlets:**
  - The servlet calls **service()** method to process a client's request
    - the main method to perform the actual task
    - servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client
    - each time the server receives a request for a servlet, the server spawns a new thread and calls service
    - service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException {
}
```



# Life Cycle

---

- **The life cycle of Servlets:**
  - The servlet calls **service()** method to process a client's request
    - A **GET** request results from a normal request for a **URL** or from an **HTML** form that has no **METHOD** specified and it should be handled by **doGet ()** method

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Servlet code
}
```



# Life Cycle

---

- **The life cycle of Servlets:**
  - The servlet calls **service()** method to process a client's request
    - A **POST** request results from an **HTML** form that specifically lists **POST** as the **METHOD** and it should be handled by **doPost()** method

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Servlet code
}
```

# Life Cycle

---

## ○ The life cycle of Servlets:

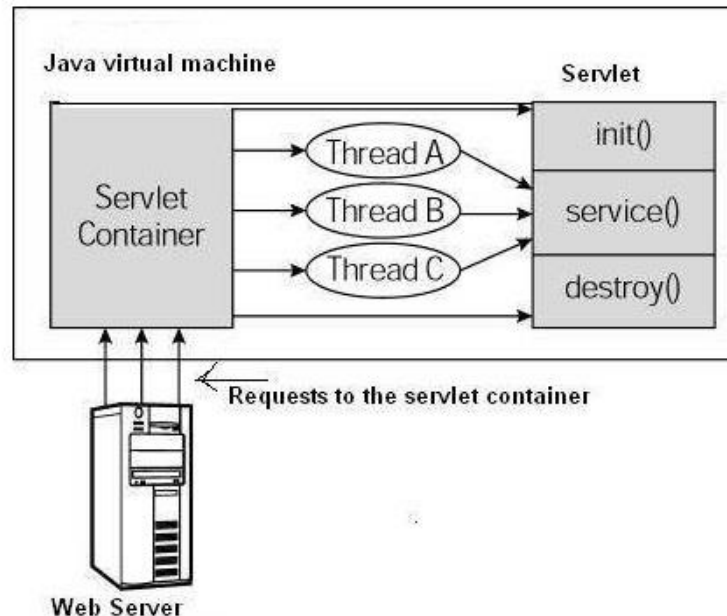
- The servlet is terminated by calling the **destroy()** method
  - called only once at the end of the life cycle of a servlet
  - it gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities
  - the servlet object is marked for garbage collection

```
public void destroy() {  
    // Finalization code...  
}
```

# Life Cycle

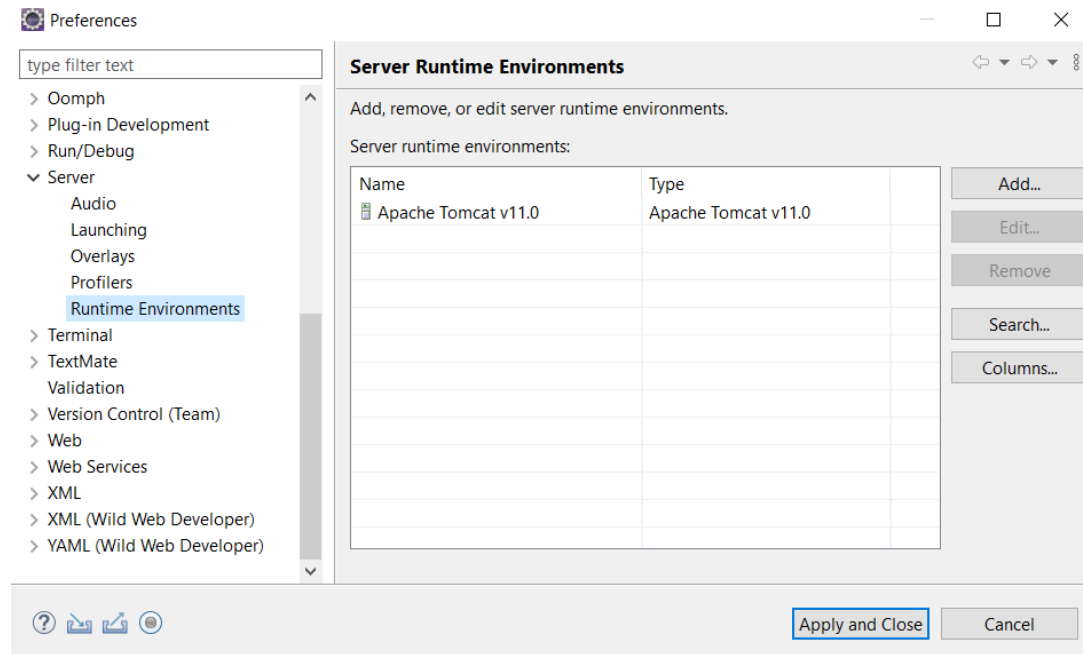
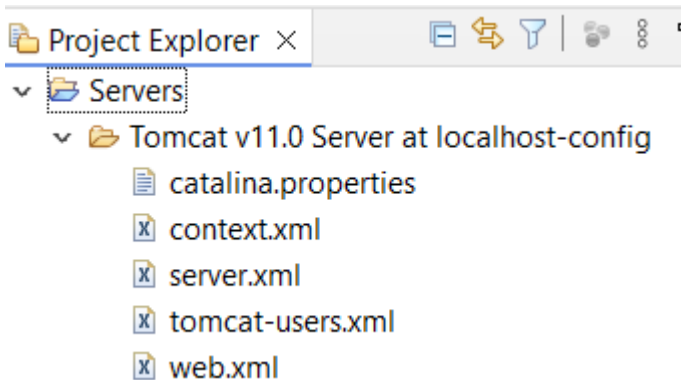
## ○ The life cycle of Servlets:

- First the HTTP requests coming to the server are delegated to the servlet container
- The servlet container loads the servlet before invoking the `service()` method
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the `service()` method of a single instance of the servlet



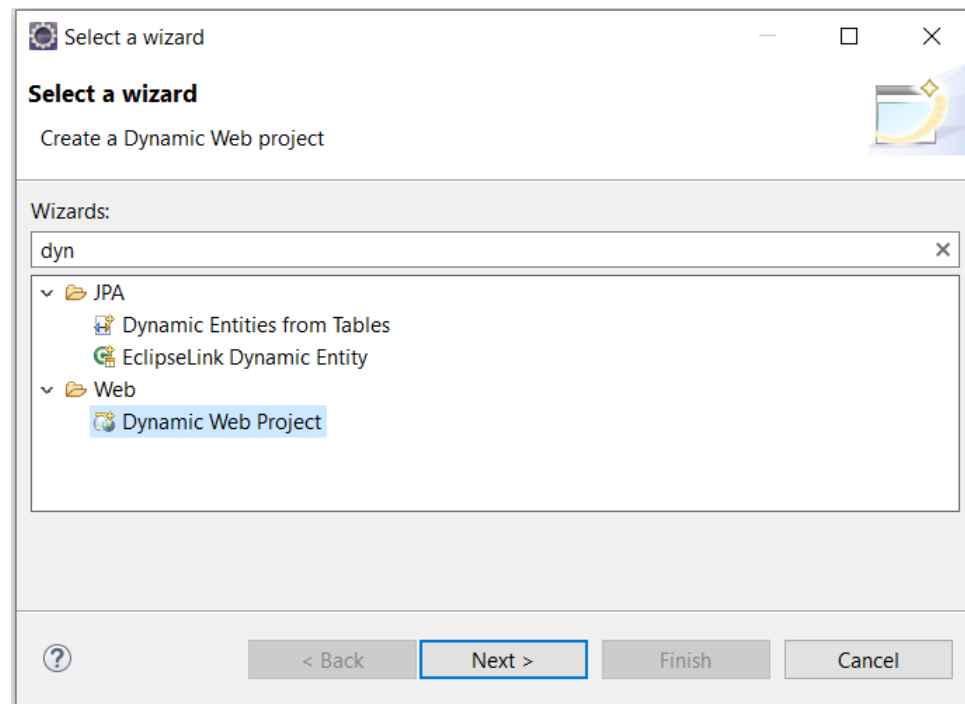
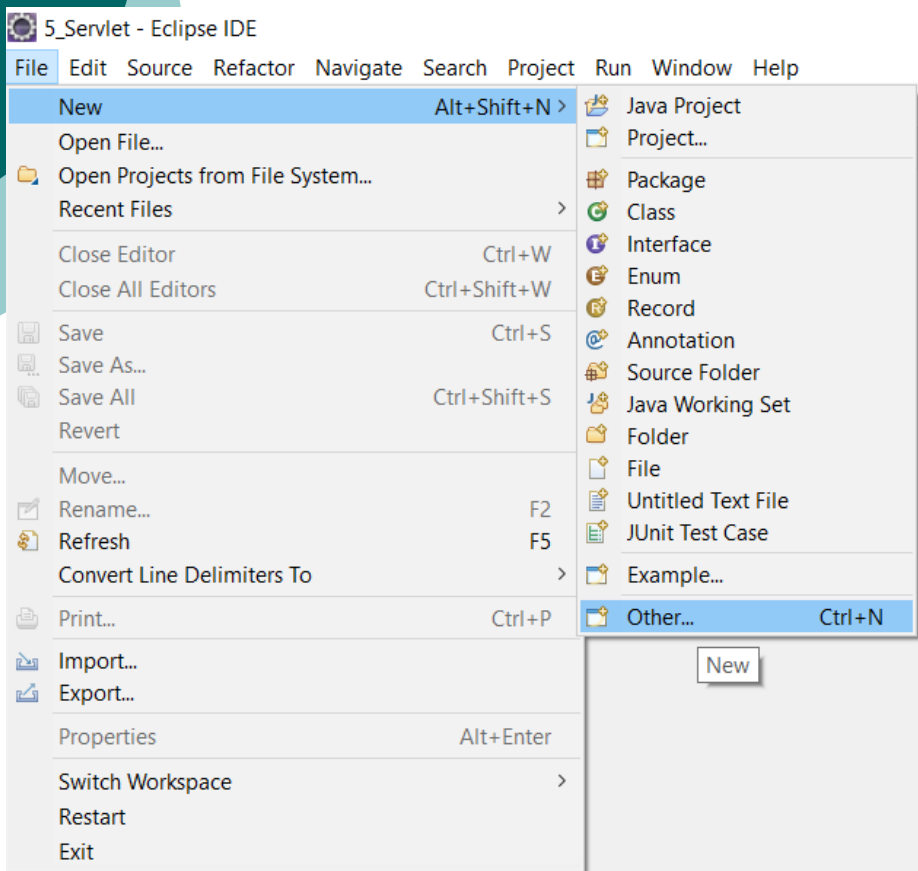
# Create new project in Eclipse

- Add the TomCat server to Eclipse:
  - Go to the “Window” menu -> “Preferences”
  - Expand “Server” -> “Runtime Environments”
  - Click on “Add...” to add a new server runtime environment
  - Select “Apache Tomcat” from the list of server types
  - Click “Next”
  - Browse and select the Tomcat installation directory
  - Click “Finish”



# Create new project in Eclipse

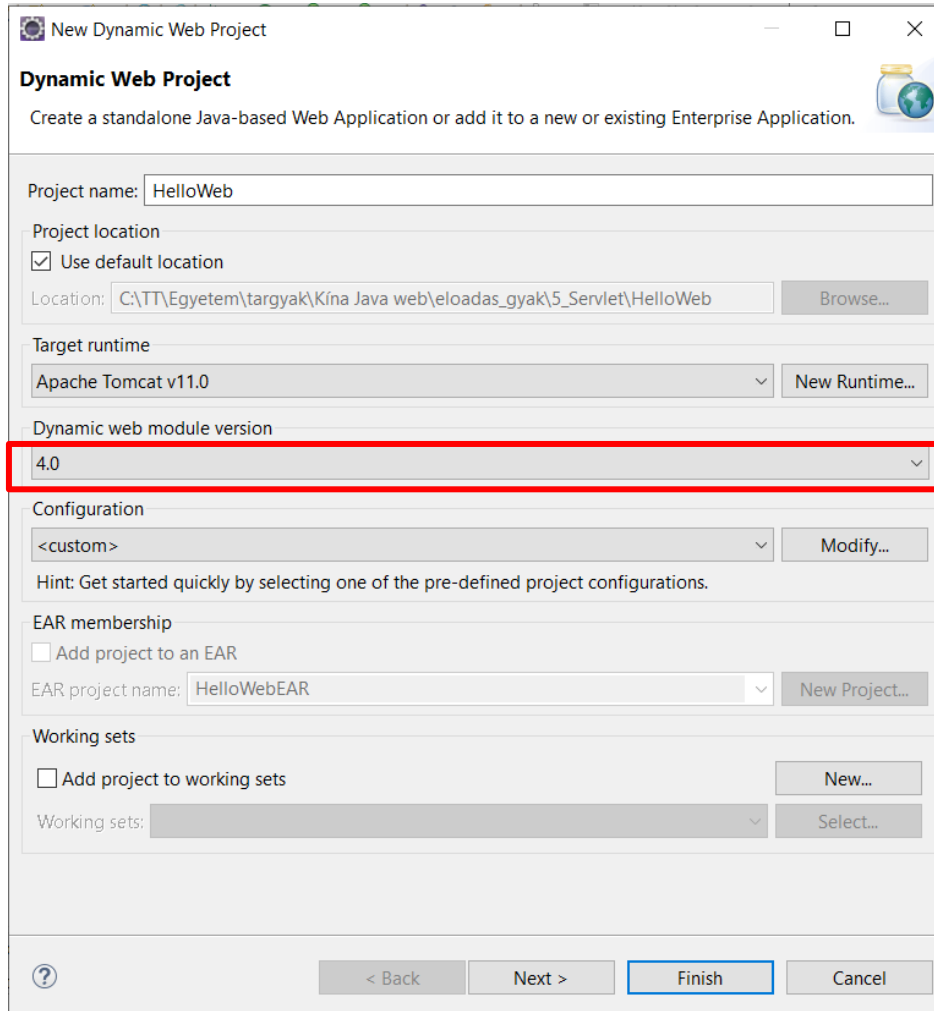
- Create the Dynamic Web project:





# Create new project in Eclipse

- Create the Dynamic Web project:



**New Dynamic Web Project**

**Dynamic Web Project**  
Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.

Project name: HelloWeb

Project location  
 Use default location  
Location: C:\TT\Egyetem\targyak\Kina Java web\eloadas\_gyak\5\_Servlet\HelloWeb Browse...

Target runtime  
Apache Tomcat v11.0 New Runtime...

Dynamic web module version  
**4.0**

Configuration  
<custom> Modify...  
Hint: Get started quickly by selecting one of the pre-defined project configurations.

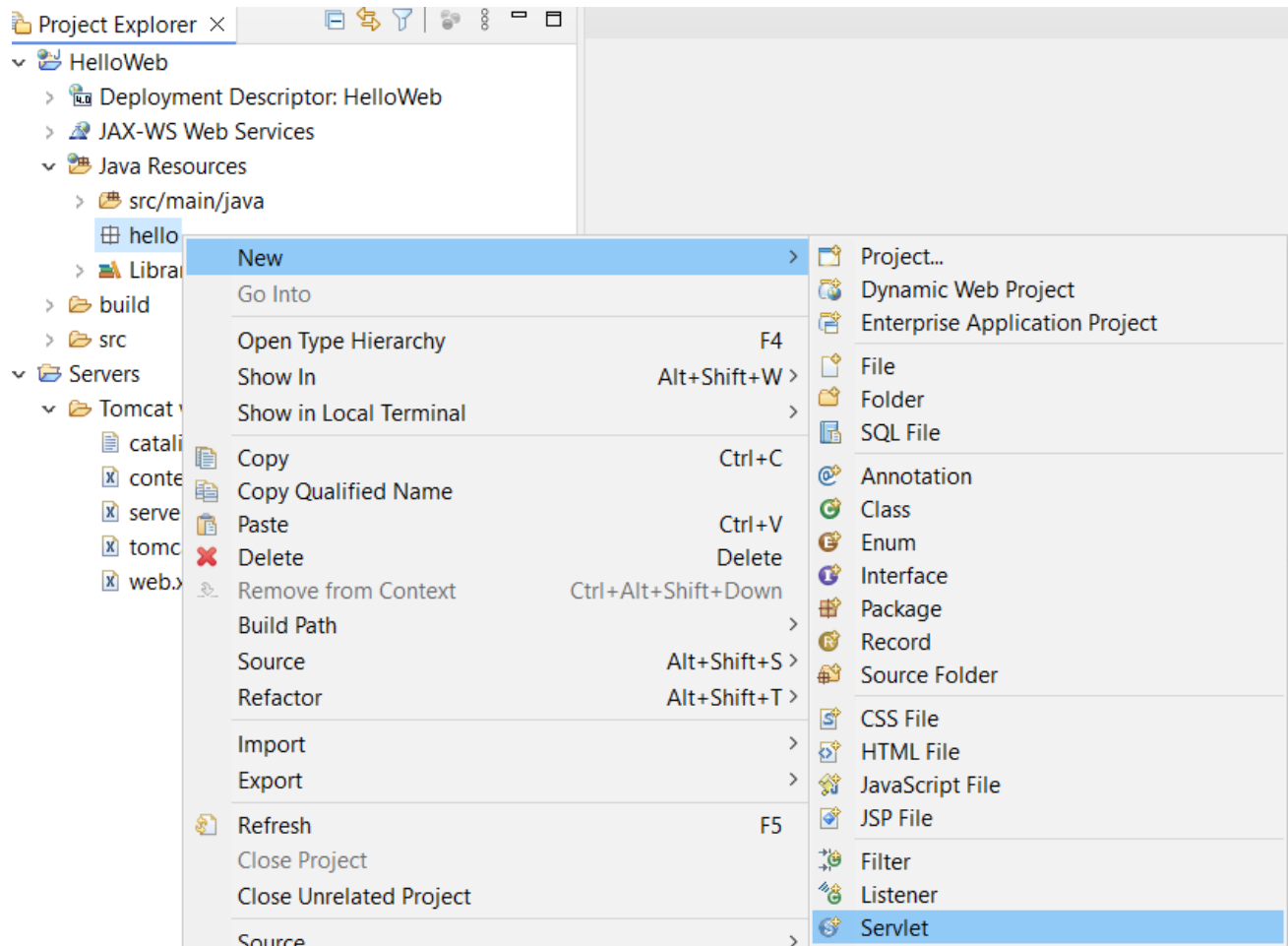
EAR membership  
 Add project to an EAR  
EAR project name: HelloWebEAR New Project...

Working sets  
 Add project to working sets New...  
Working sets: Select...

? < Back Next > Finish Cancel

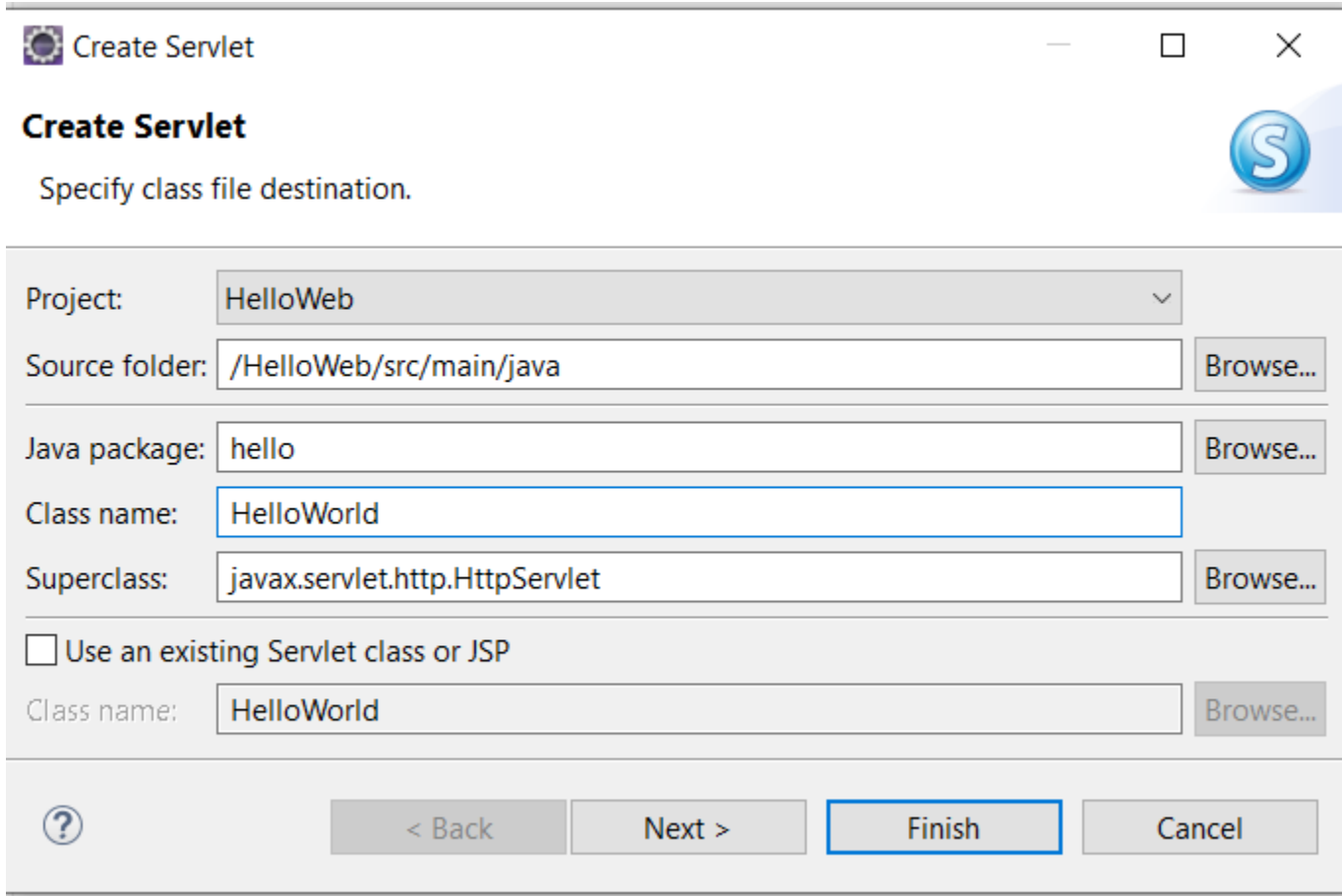
# Create new project in Eclipse

- Create the Dynamic Web project:



# Create new project in Eclipse

- Create the Dynamic Web project:



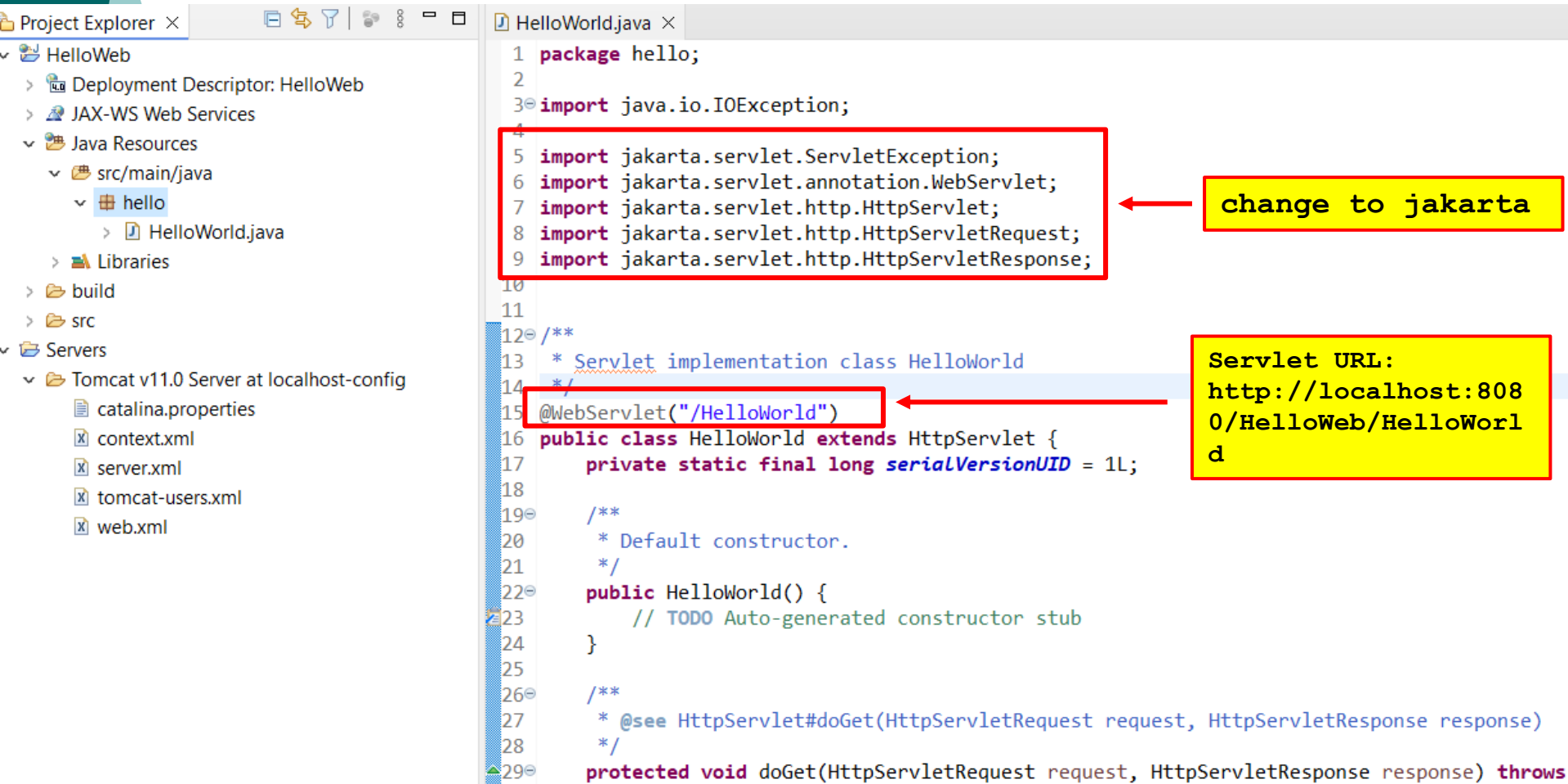
The screenshot shows the 'Create Servlet' dialog box in Eclipse. The dialog has a title bar with the text 'Create Servlet' and standard window controls. Below the title bar, the text 'Create Servlet' is displayed in bold, followed by the instruction 'Specify class file destination.' and a blue circular icon with a white 'S'. The dialog contains several input fields and buttons:

- Project:** A dropdown menu with 'HelloWeb' selected.
- Source folder:** A text field containing '/HelloWeb/src/main/java' and a 'Browse...' button to its right.
- Java package:** A text field containing 'hello' and a 'Browse...' button to its right.
- Class name:** A text field containing 'HelloWorld'.
- Superclass:** A text field containing 'javax.servlet.http.HttpServlet' and a 'Browse...' button to its right.
- Use an existing Servlet class or JSP
- Class name:** A text field containing 'HelloWorld' and a 'Browse...' button to its right.

At the bottom of the dialog, there is a row of buttons: a help icon (?), '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

# Create new project in Eclipse

## ○ Create the Dynamic Web project:



The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer showing a project named 'HelloWeb' with a sub-project 'hello' containing 'HelloWorld.java'. The main editor displays the code for 'HelloWorld.java'. The code is as follows:

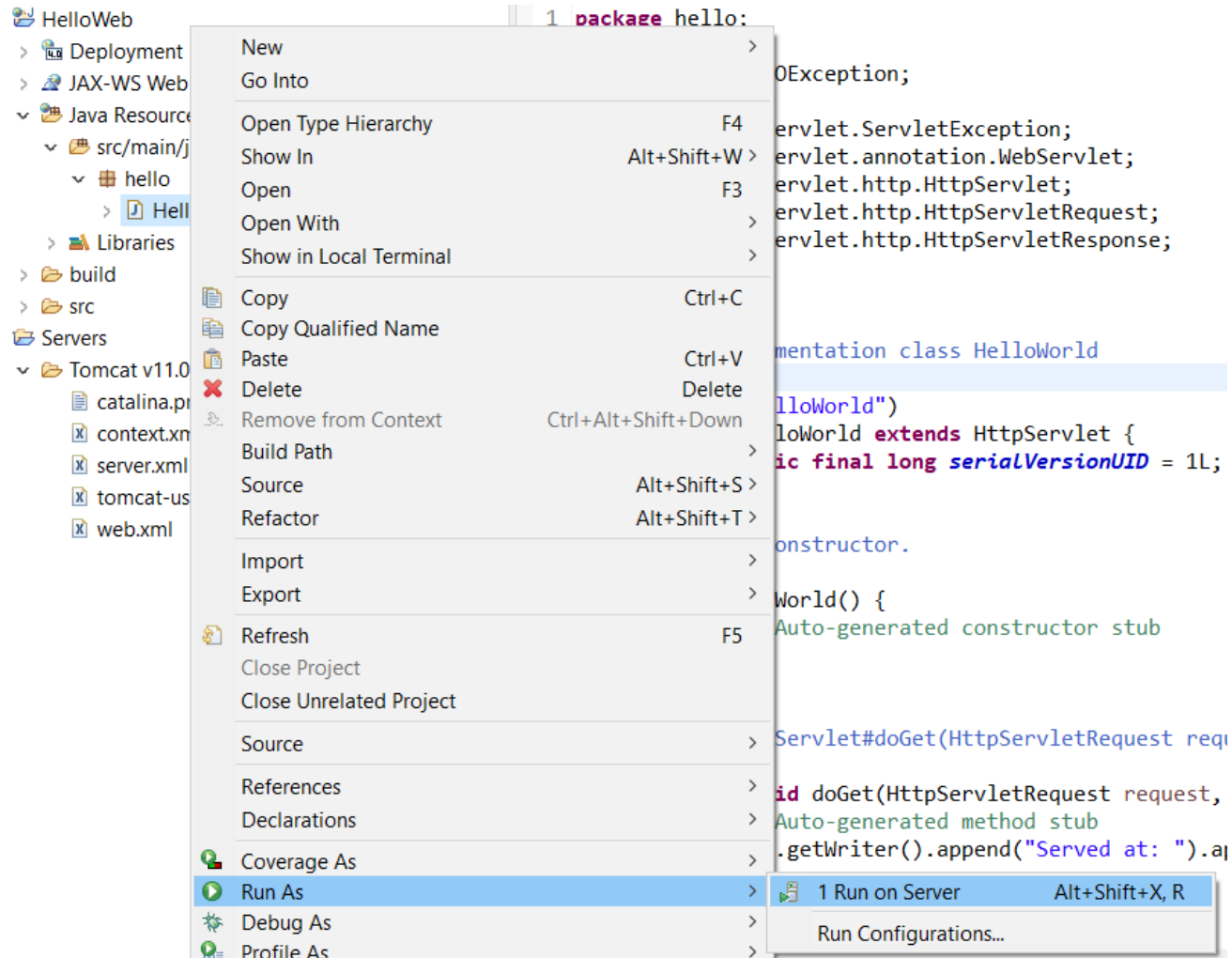
```
1 package hello;
2
3 import java.io.IOException;
4
5 import jakarta.servlet.ServletException;
6 import jakarta.servlet.annotation.WebServlet;
7 import jakarta.servlet.http.HttpServlet;
8 import jakarta.servlet.http.HttpServletRequest;
9 import jakarta.servlet.http.HttpServletResponse;
10
11
12 /**
13  * Servlet implementation class HelloWorld
14  */
15 @WebServlet("/HelloWorld")
16 public class HelloWorld extends HttpServlet {
17     private static final long serialVersionUID = 1L;
18
19     /**
20      * Default constructor.
21      */
22     public HelloWorld() {
23         // TODO Auto-generated constructor stub
24     }
25
26     /**
27      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
28      */
29     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
```

Annotations and imports are highlighted with red boxes and arrows:

- A red box highlights the import statements from line 5 to 9: `import jakarta.servlet.*`. An arrow points from a yellow box labeled "change to jakarta" to this box.
- A red box highlights the `@WebServlet("/HelloWorld")` annotation on line 15. An arrow points from a yellow box labeled "Servlet URL: http://localhost:8080/HelloWeb/HelloWorld" to this box.

# Create new project in Eclipse

## ○ Run the Dynamic Web project:



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer shows a project named 'HelloWeb' with a package structure: 'src/main/java/hello/HelloWorld.java'. A context menu is open over the 'HelloWorld' class, listing various actions. The 'Run As' option is highlighted in blue. A sub-menu is visible for 'Run As', showing '1 Run on Server' selected, with the keyboard shortcut 'Alt+Shift+X, R'. The main editor window shows the source code of the 'HelloWorld' class, which is a simple servlet implementation.

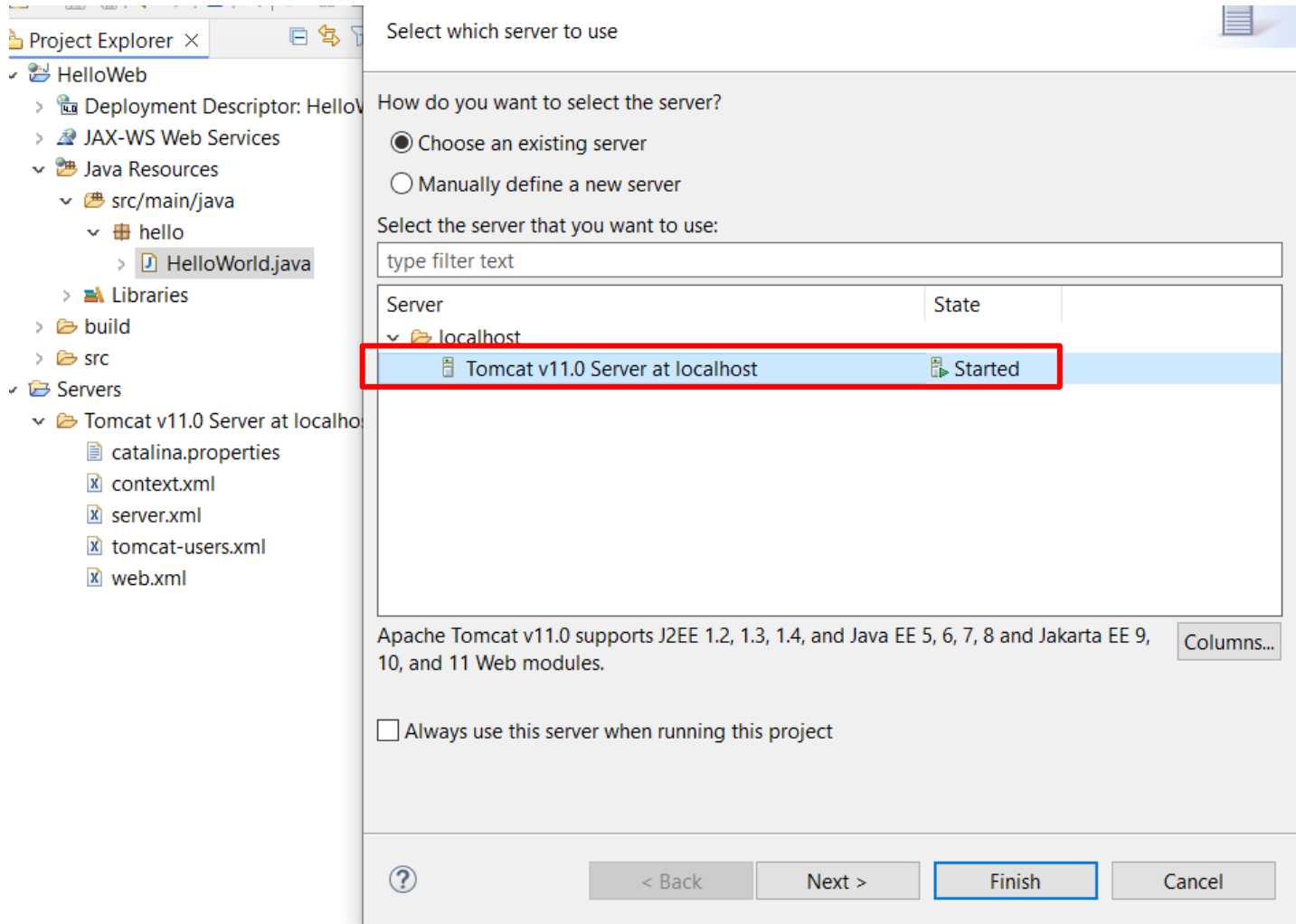
```

1 package hello:
2 import java.io.IOException;
3 import javax.servlet.ServletException;
4 import javax.servlet.annotation.WebServlet;
5 import javax.servlet.http.HttpServlet;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 // Commented out JSP page class loader (see https://bugs.eclipse.org/bugs/show_bug.cgi?id=18608)
10 // import org.eclipse.jdt.internal.compiler.tool.ToolClassLoader;
11
12 /**
13  * Servlet implementation class HelloWorld
14  */
15 @WebServlet("/helloWorld")
16 public class HelloWorld extends HttpServlet {
17     private static final long serialVersionUID = 1L;
18
19     // Constructors
20
21     /**
22      * Constructor for the servlet
23      */
24     public HelloWorld() {
25         // Auto-generated constructor stub
26     }
27
28     /**
29      * Servlet#get(HttpServletRequest request)
30      */
31     @Override
32     protected void doGet(HttpServletRequest request,
33                           HttpServletResponse response)
34         throws ServletException, IOException {
35         // Auto-generated method stub
36         response.getWriter().append("Served at: ").append(request.getDate());
37     }
38 }

```

# Create new project in Eclipse

## ○ Run the Dynamic Web project:



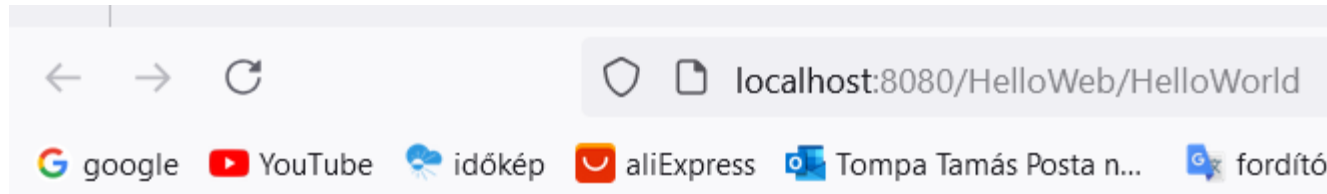
The screenshot shows the Eclipse IDE interface with the 'Select which server to use' dialog box open. The 'Project Explorer' on the left shows a project named 'HelloWeb' with a 'HelloWorld.java' file selected. The dialog box asks 'How do you want to select the server?' and offers two options: 'Choose an existing server' (selected) and 'Manually define a new server'. Below this, it says 'Select the server that you want to use:' and provides a search filter 'type filter text'. A table lists available servers:

Server	State
localhost	
Tomcat v11.0 Server at localhost	Started

The 'Tomcat v11.0 Server at localhost' entry is highlighted with a red box. Below the table, it states: 'Apache Tomcat v11.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, 8 and Jakarta EE 9, 10, and 11 Web modules.' There is a checkbox for 'Always use this server when running this project' which is currently unchecked. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

# Create new project in Eclipse

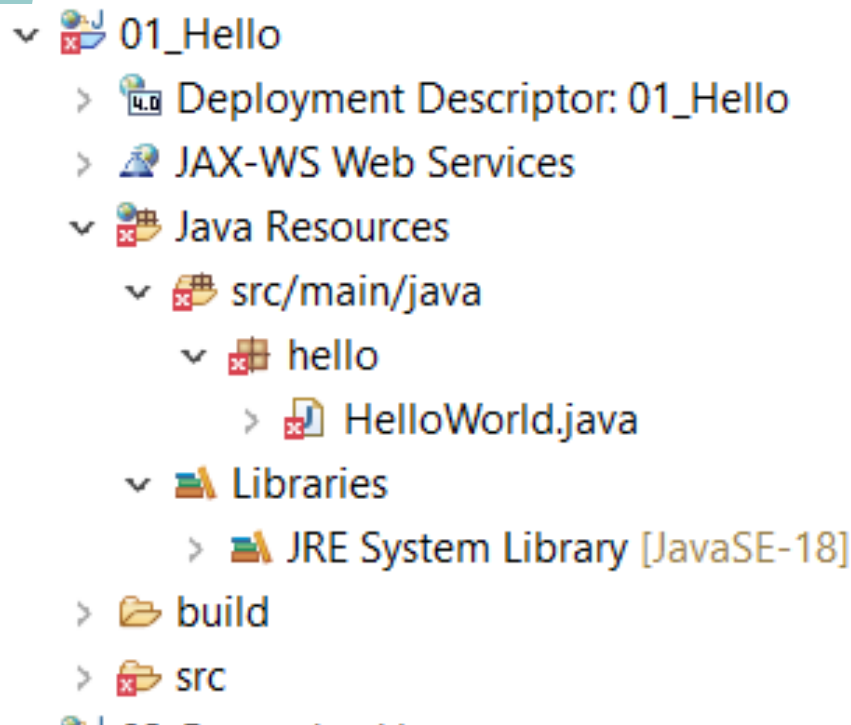
- Run the **Dynamic Web** project:



Served at: /HelloWeb

# Create new project in Eclipse

- Add server library to the project (if it is necessary):

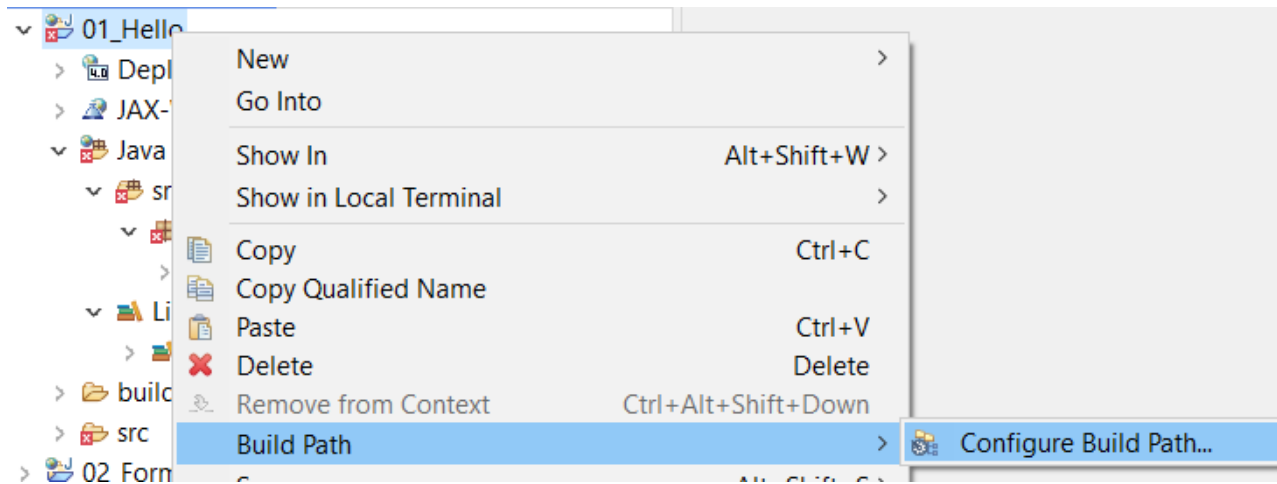


**Server runtime library is missing**



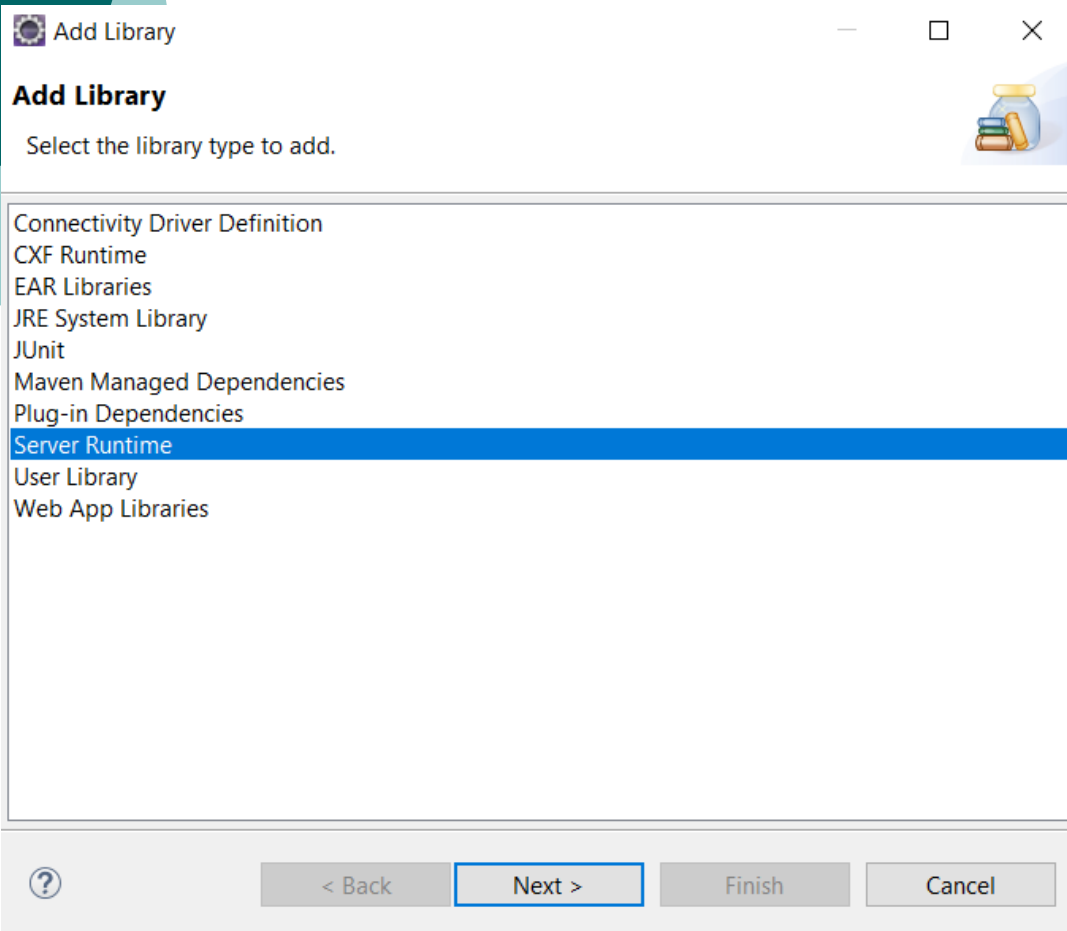
# Create new project in Eclipse


- Add server library to the project (if it is necessary):



# Create new project in Eclipse

- Add server library to the project (if it is necessary):




 Add Library

## Server Library

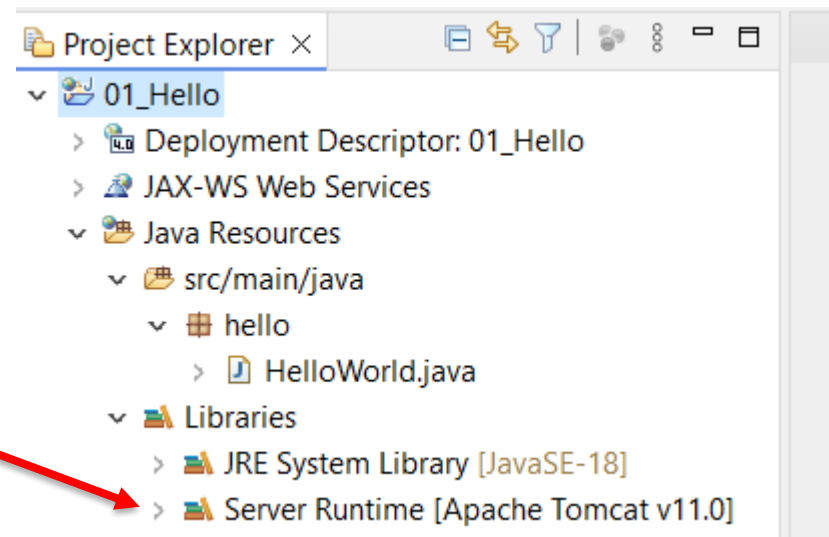
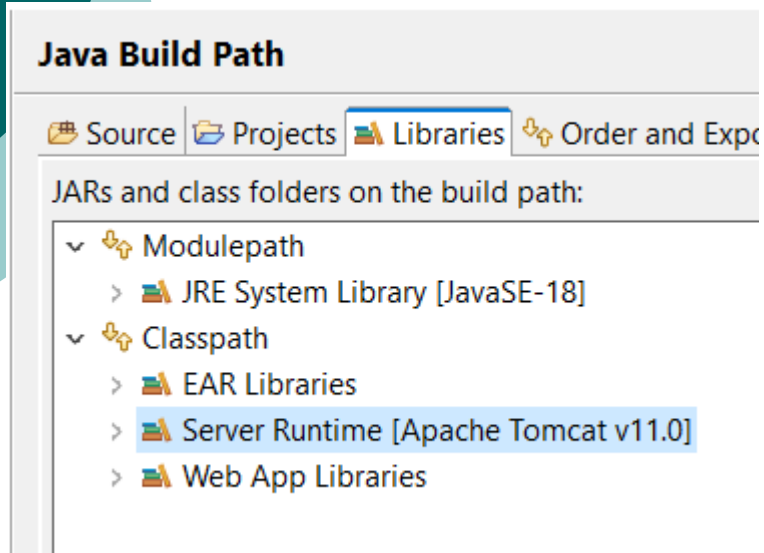
Select a server runtime for the project build path.

Runtime environments:

 Apache Tomcat v11.0

# Create new project in Eclipse

- Add server library to the project (if it is necessary):



**Server runtime library added successfully**



# Servlet deployment

---

- By default, a **servlet application** is located at the path **<Tomcat-installationdirectory>/webapps/ROOT** and the **class file** would reside in **<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes**
- If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in **WEB-INF/classes/com/myorg/MyServlet.class**



# Servlet deployment

---

- For now, let us copy HelloWorld.class into `<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes` and create following entries in web.xml file located in `<Tomcat-installation-directory>/webapps/ROOT/WEB-INF/`

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>

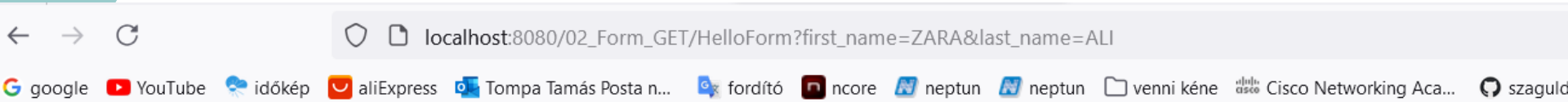
<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

- <http://localhost:8080/HelloWorld>

# Task2: Form data - GET

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- [http://localhost:8080/HelloForm?first\\_name=ZARA&last\\_name=ALI](http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI)



## Using GET Method to Read Form Data

- **First Name:** ZARA
- **Last Name:** ALI

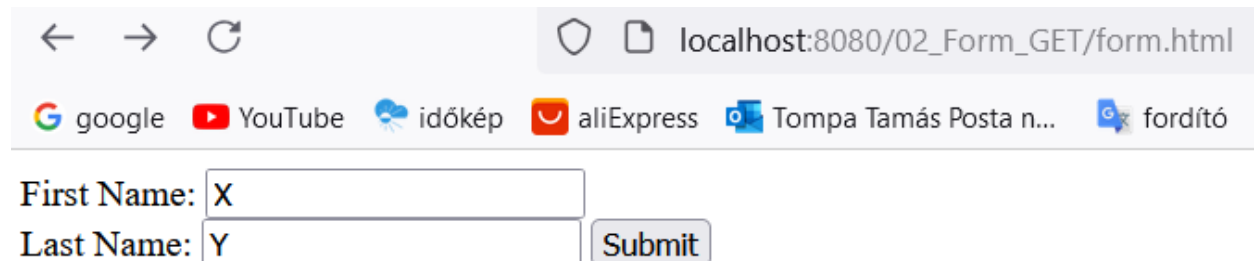
# Task2: Form data - GET

## ○ Add an HTML Form:

```
02_Form_GET
├── Deployment Descriptor: 02_Form_GET
├── JAX-WS Web Services
├── Java Resources
│   ├── src/main/java
│   │   └── form
│   │       └── HelloForm.java
│   └── Libraries
├── build
├── src
│   └── main
│       └── java
│           └── webapp
│               ├── META-INF
│               ├── WEB-INF
│               └── form.html
└── ...
```

```
<html>
<body>
  <form action = "HelloForm" method = "GET">
    First Name: <input type = "text" name = "first_name">
    <br />
    Last Name: <input type = "text" name = "last_name" />
    <input type = "submit" value = "Submit" />
  </form>
</body>
</html>
```

URL: [http://localhost:8080/02\\_Form\\_GET/form.html](http://localhost:8080/02_Form_GET/form.html)



← → ↻ localhost:8080/02\_Form\_GET/form.html

google YouTube időkép aliExpress Tompa Tamás Posta n... fordító

First Name:

Last Name:

# Task2: Form data - POST

---

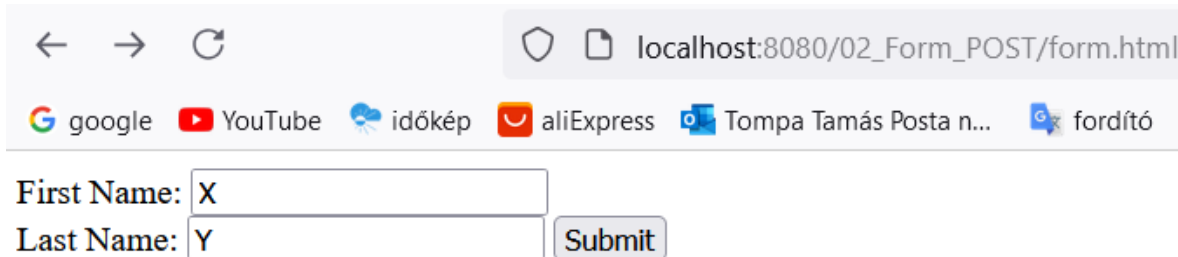
- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- Form code:

```
<html>
<body>
  <form action = "HelloForm" method = "POST">
    First Name: <input type = "text" name = "first_name">
    <br />
    Last Name: <input type = "text" name = "last_name" />
    <input type = "submit" value = "Submit" />
  </form>
</body>
</html>
```

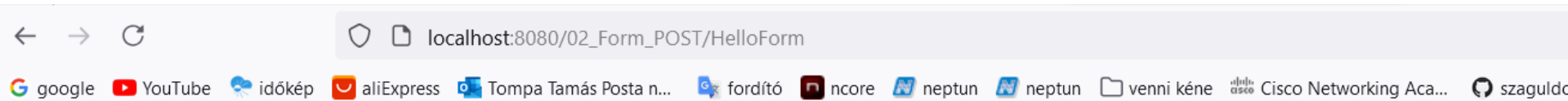


# Task2: Form data - POST

- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- URL: [http://localhost:8080/02\\_Form\\_POST/form.html](http://localhost:8080/02_Form_POST/form.html)



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/02_Form_POST/form.html`. Below the address bar, there are several search engines and services: google, YouTube, időkép, aliExpress, Tompa Tamás Posta n..., and fordító. The main content area of the browser displays a simple form with two input fields. The first field is labeled "First Name:" and contains the text "X". The second field is labeled "Last Name:" and contains the text "Y". To the right of the second field is a button labeled "Submit".



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/02_Form_POST/HelloForm`. Below the address bar, there are several search engines and services: google, YouTube, időkép, aliExpress, Tompa Tamás Posta n..., fordító, ncore, neptun, neptun, venni kéne, Cisco Networking Aca..., and szaguld. The main content area of the browser displays a simple form with two input fields. The first field is labeled "First Name:" and contains the text "X". The second field is labeled "Last Name:" and contains the text "Y". To the right of the second field is a button labeled "Submit".

## Using POST Method to Read Form Data

- First Name: X
- Last Name: Y

# Task3: Checkbox

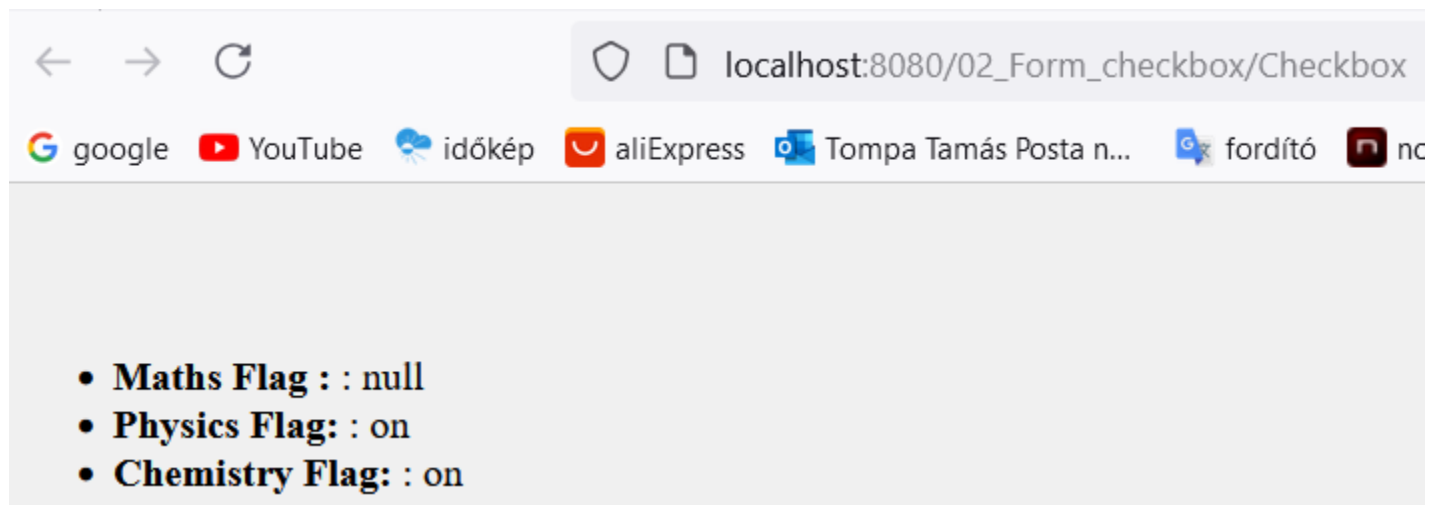
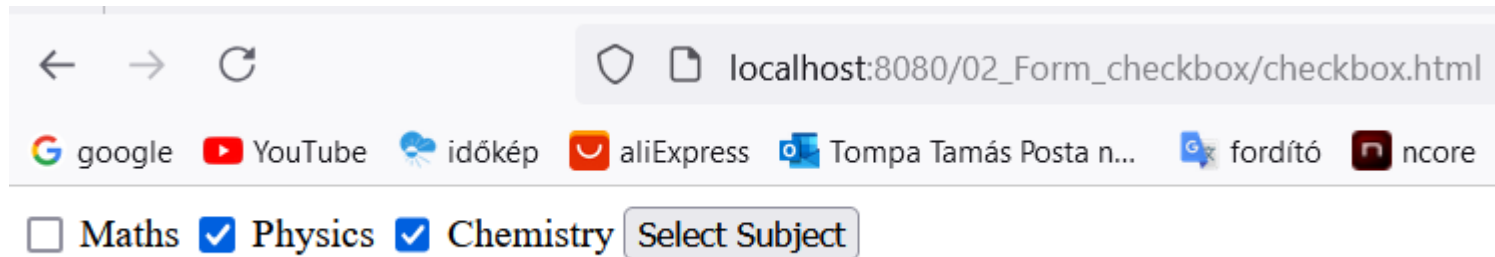
---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- URL: [http://localhost:8080/02\\_Form\\_checkbox/checkbox.html](http://localhost:8080/02_Form_checkbox/checkbox.html)
- Form code:

```
<html>
<body>
  <form action = "Checkbox" method = "POST" target = "_blank">
    <input type = "checkbox" name = "maths" checked = "checked" /> Maths
    <input type = "checkbox" name = "physics" /> Physics
    <input type = "checkbox" name = "chemistry" checked = "checked" /> Chemistry
    <input type = "submit" value = "Select Subject" />
  </form>
</body>
</html>
```

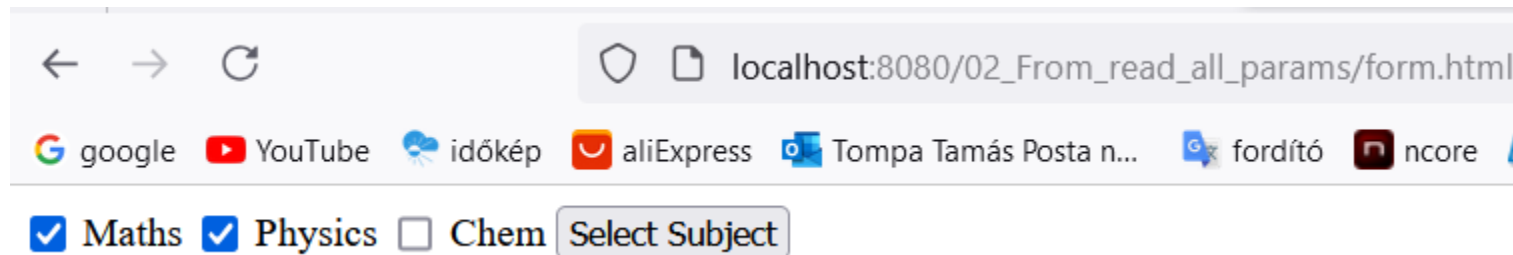
# Task3: Checkbox

- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- URL: [http://localhost:8080/02\\_Form\\_checkbox/checkbox.html](http://localhost:8080/02_Form_checkbox/checkbox.html)



# Task4: Read all params of a form

- Source code: <https://www.tutorialspoint.com/servlets/servlets-form-data.htm>
- URL: [http://localhost:8080/02\\_From\\_read\\_all\\_params/form.html](http://localhost:8080/02_From_read_all_params/form.html)



## Reading All Form Parameters

Param Name	Param Value(s)
maths	on
physics	on



# Task5: HTTP Header Request

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-client-request.htm>
- **getHeaderNames ()**
  - **HttpServletRequest** to read the **HTTP** header information
  - **returns an Enumeration** that contains the header information associated with the current HTTP request
- **HTTP** header is used to pass additional information between the client (such as a browser) and the server during an **HTTP** request or response
  - **It contains metadata about the communication:**
    - **Content-Type:** Specifies the media type of the resource (e.g., text/html, application/json)
    - **Content-Length:** Indicates the size of the resource in bytes
    - **User-Agent:** Identifies the client making the request
    - **Authorization:** Contains credentials for authenticating the client
    - **Cache-Control:** Defines caching policies for the response

# Task5: HTTP Header Request

- Source code: <https://www.tutorialspoint.com/servlets/servlets-client-request.htm>
- `getHeaderNames ()`

## HTTP Header Request Example

Header Name	Header Value(s)
host	localhost:8080
user-agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
accept-language	hu-HU,hu;q=0.8,en-US;q=0.5,en;q=0.3
accept-encoding	gzip, deflate, br, zstd
connection	keep-alive
upgrade-insecure-requests	1
sec-fetch-dest	document
sec-fetch-mode	navigate
sec-fetch-site	none
sec-fetch-user	?1
priority	u=0, i



# Task6: Server HTTP Response

---

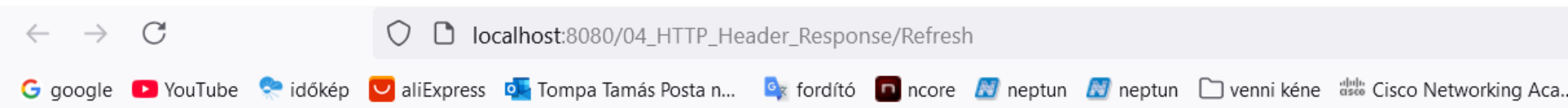
- Source code: <https://www.tutorialspoint.com/servlets/servlets-server-response.htm>
- server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document
- **setContentType ()**
  - This method is used to **set the MIME** (Multipurpose Internet Mail Extensions) type of the HTTP response
  - It **informs the client** (like a browser) **about the type of data** it will receive, such as text/html, application/json, or text/plain
  - `response.setContentType("text/html");`
- **setIntHeader ()**
  - This method sets an **HTTP response header with an integer value**
  - It's **used to set or overwrite specific headers**, such as setting a status code or defining a header like Content-Length
  - `response.setIntHeader("Content-Length", 1024);`

# Task6: Server HTTP Response

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-server-response.htm>
- server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document

```
// Set refresh, autoload time as 5 seconds  
response.setIntHeader("Refresh", 5);
```



## Auto Refresh Header Setting

Current Time is: 1:53:31 PM





# Task7: Http Status Codes

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-http-status-codes.htm>
- Methods to Set HTTP Status Code
  - **public void setStatus ( int statusCode )**
    - sets an arbitrary status code. The setStatus method takes an int (the status code) as an argument. If your response includes a special status code and a document, be sure to call setStatus before actually returning any of the content with the PrintWriter
  - **public void sendRedirect(String url)**
    - generates a 302 response along with a Location header giving the URL of the new document
  - **public void sendError(int code, String message)**
    - sends a status code (usually 404) along with a short message that is automatically formatted inside an HTML document and sent to the client.



# Task7: Http Status Codes

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-http-status-codes.htm>

```
// Set error code and reason.  
response.sendError(407, "Need authentication!!!");
```

## HTTP Status 407 – Proxy Authentication Required

**Type** Status Report

**Message** Need authentication!!!

**Description** This status code is similar to 401 (Unauthorized), but it indicates that the client needs to authenticate itself in order to use a proxy.

**Apache Tomcat/11.0.0**



# Task8: Writing filters

---

- **Servlet Filters are Java classes**
  - **To intercept requests from a client before they access** a resource at back end
  - **To manipulate responses from server before they are sent back** to the client
  - Types: Authentication Filters, Data compression Filters, Encryption Filters, Filters that trigger resource access events, Image Conversion Filters, etc.
- **Servlet Filter Methods**
  - `public void doFilter (ServletRequest, ServletResponse, FilterChain)`
  - `public void init(FilterConfig filterConfig)`
  - `public void destroy()`



# Task8: Writing filters

---

## ○ Servlet Filter Methods

- **public void doFilter (ServletRequest, ServletResponse, FilterChain)**
  - called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain
- **public void init(FilterConfig filterConfig)**
  - called by the web container to indicate to a filter that it is being placed into service
- **public void destroy()**
  - called by the web container to indicate to a filter that it is being taken out of service
- **Filter mapping have to define in the web.xml file of the project**
  - automatically runs on every HTTP request when the specified URL pattern (/\*) is called

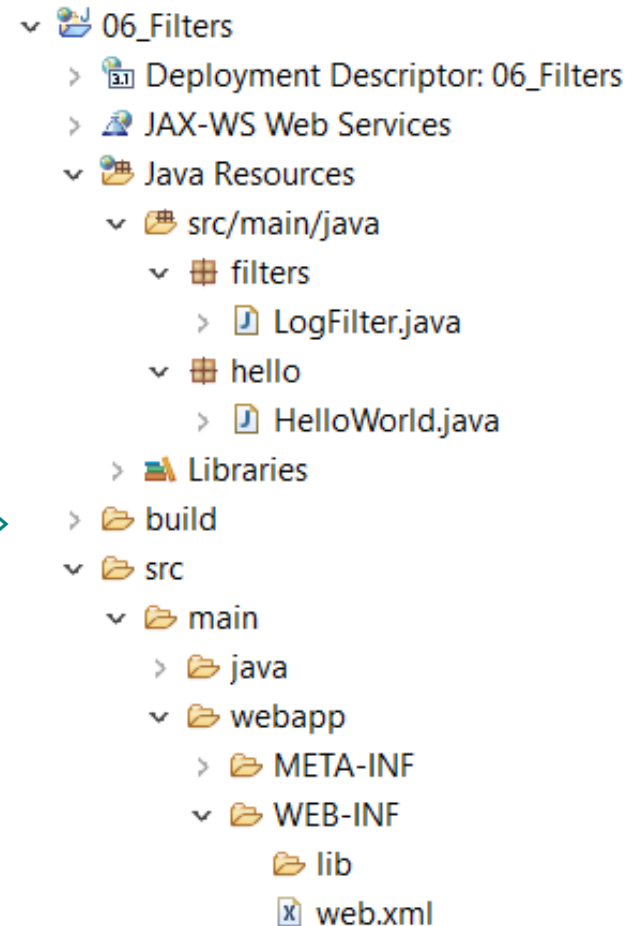
# Task8: Writing filters

- Source code: <https://www.tutorialspoint.com/servlets/servlets-writing-filters.htm>
- Copy the HelloWorld class from the 01\_Hello

## Servlet Filter Mapping in the web.xml:

```
<filter>
<filter-name>LogFilter</filter-name>
<filter-class>filters.LogFilter</filter-class>
<init-param>
<param-name>test-param</param-name>
<param-value>Initialization Parameter</param-value>
</init-param>
</filter>
```

```
<filter-mapping>
<filter-name>LogFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```





# Task9: Exception Handling

---

- Source code: <https://www.tutorialspoint.com/servlets/servlets-exception-handling.htm>
- When a servlet throws an exception, the web container searches the configurations in web.xml
- You would have to use the error-page element in web.xml to specify the invocation of servlets in response to certain exceptions or HTTP status codes
  - The servlet ErrorHandler is defined in usual way as any other servlet and configured in web.xml
  - If there is any error with status code either 404 (Not Found) or 403 (Forbidden ), then ErrorHandler servlet would be called
  - If the web application throws either ServletException or IOException, then the web container invokes the /ErrorHandler servlet
  - You can define different Error Handlers to handle different type of errors or exceptions. Above example is very much generic and hope it serve the purpose to explain you the basic concept



# Task10: Cookies Handling

- Source code: <https://www.tutorialspoint.com/servlets/servlets-cookies-handling.htm>
- **Cookies are text files stored on the client computer** and they are kept for various information tracking purpose
- **Small piece of data that a server sends to a user's web browser and is stored on the user's device**
- **There are three steps involved in identifying returning users:**
  - **server script sends a set of cookies to the browser.** For example name, age, or identification number etc.
  - **browser stores this information on local machine for future use**
  - **when next time browser sends any request to web server then it sends those cookies information to the server** and server uses that information to identify the user
- Cookies are used to store information such as user preferences, session data, or tracking information, enabling a website to remember the user across visits or track behavior for personalized experiences



# Task10: Cookies Handling -setting

---

- Cookies are usually set in an HTTP header

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38
GMT;
path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

- Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date
- If the browser is configured to store cookies, it will then keep this information until the expiry date

```
Cookie cookie = new Cookie("key", "value");
```





Servlets

# Task10: Cookies Handling -setting

← → ↻ localhost:8080/07\_Cookies\_handling\_setting/HelloForm.html

google YouTube időkép aliExpress Tompa Tamás Posta n... fordító ncore neptun

First Name:

Last Name:

← → ↻ localhost:8080/07\_Cookies\_handling\_setting/HelloForm?first\_name=Tamas&last\_name=Tompa

google YouTube időkép aliExpress Tompa Tamás Posta n... fordító ncore neptun neptun venni kéne Cisco Networking

## Setting Cookies Example

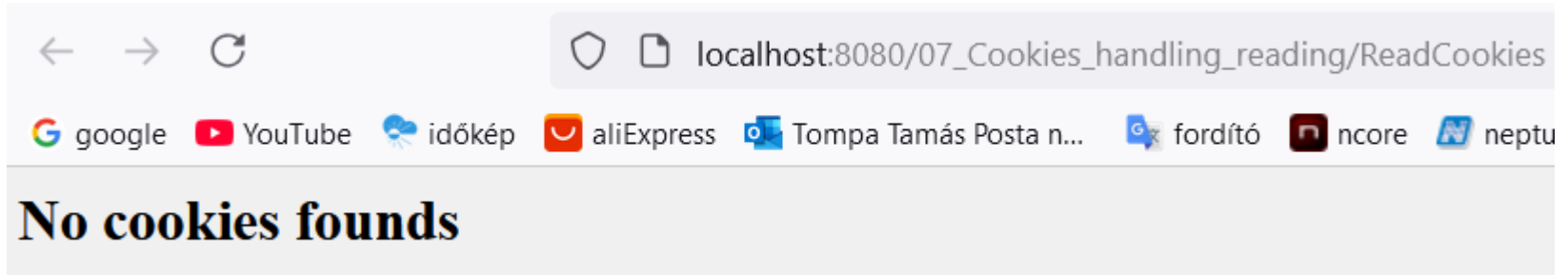
- **First Name:** Tamas
- **Last Name:** Tompa



Servlets

# Task10: Cookies Handling -reading

- Source code: <https://www.tutorialspoint.com/servlets/servlets-cookies-handling.htm>





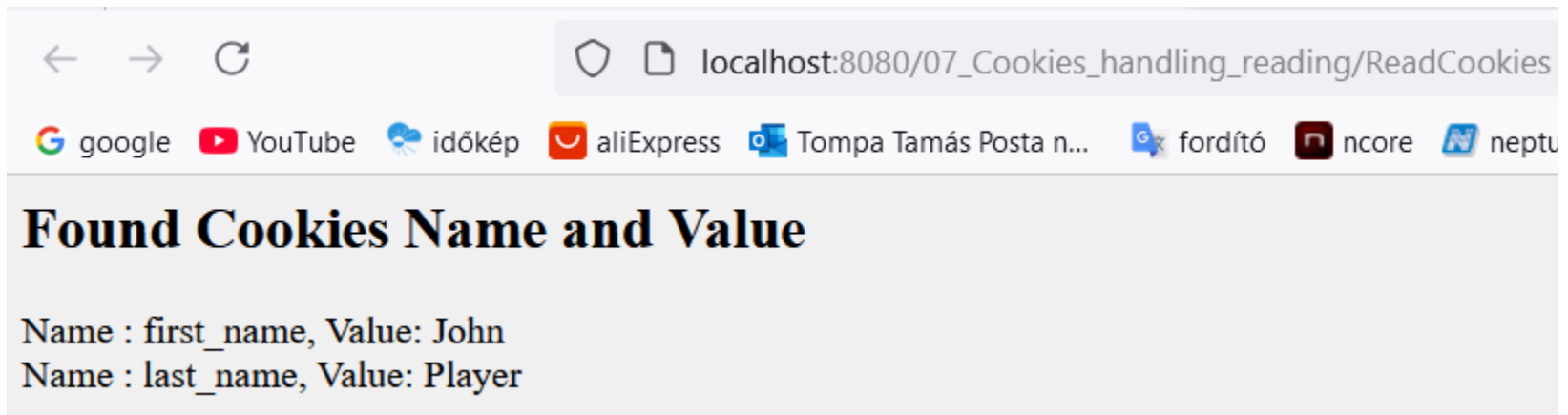
# Task10: Cookies Handling -reading

- Source code: <https://www.tutorialspoint.com/servlets/servlets-cookies-handling.htm>

```
// Create cookies
Cookie firstName = new Cookie("first_name", "John");
Cookie lastName = new Cookie("last_name", "Player");

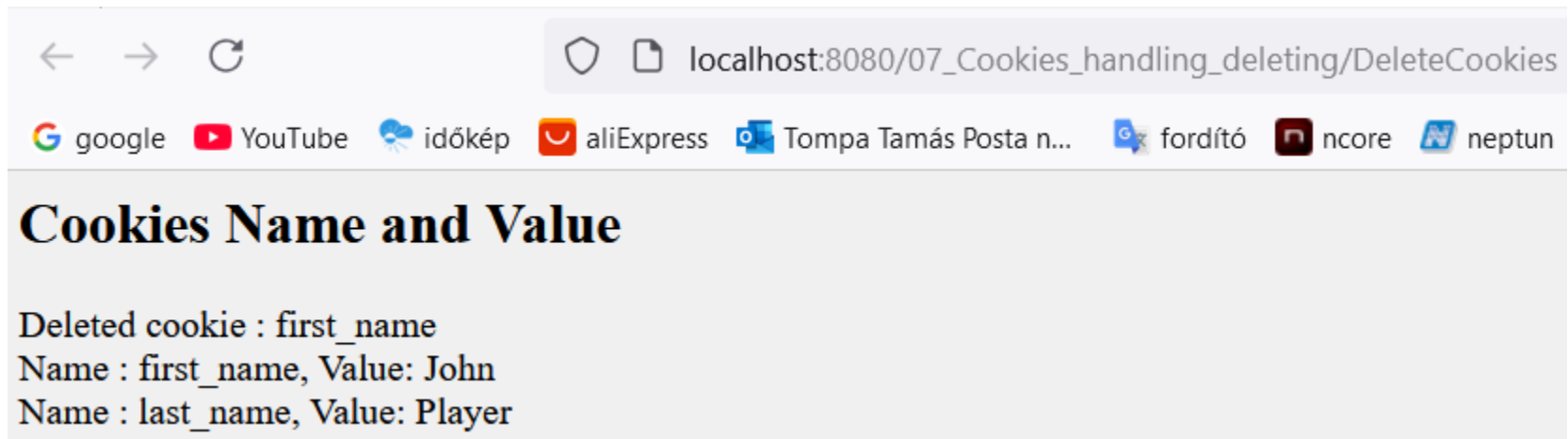
// Set cookies to expire in 24 hours
firstName.setMaxAge(60 * 60 * 24);
lastName.setMaxAge(60 * 60 * 24);

// Add cookies to response
response.addCookie(firstName);
response.addCookie(lastName);
```



# Task10: Cookies Handling -delete

- Source code: <https://www.tutorialspoint.com/servlets/servlets-cookies-handling.htm>
- **Read an already existing cookie** and store it in Cookie object
  - `cookies = request.getCookies();`
- **Set cookie age as zero** using `setMaxAge()` method to delete an existing cookie
  - `cookie.setMaxAge(0);`
- **Add this cookie back into response header**
  - `response.addCookie(cookie);`



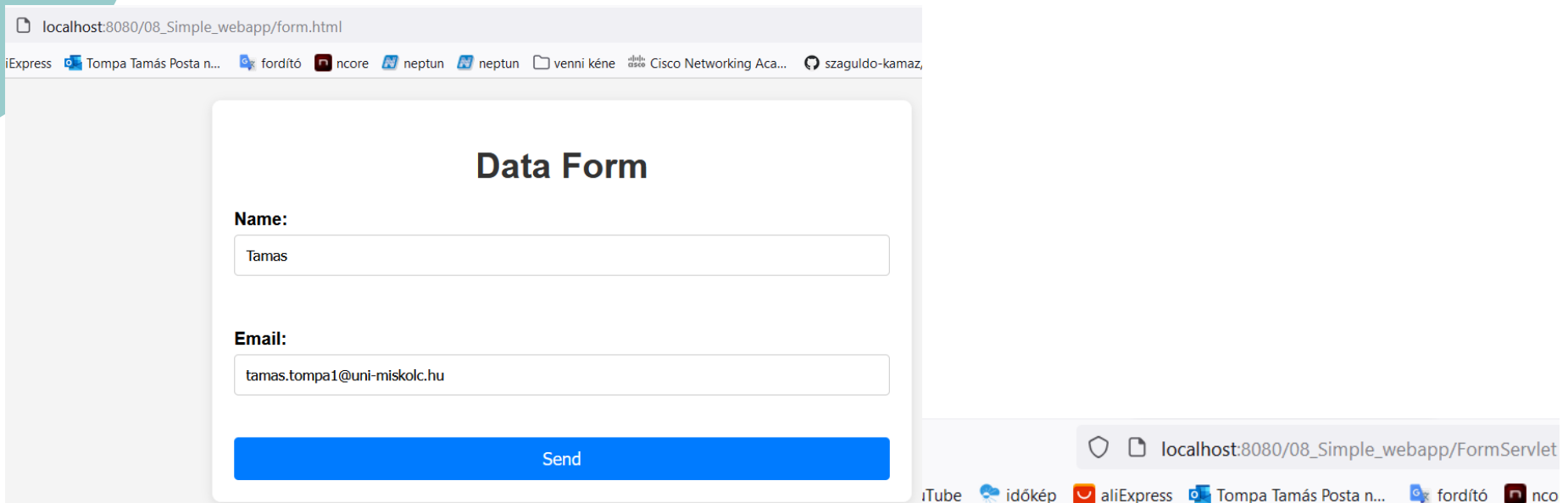
The screenshot shows a web browser window with the address bar displaying `localhost:8080/07_Cookies_handling_deleting/DeleteCookies`. Below the address bar, there are several search and utility icons including Google, YouTube, időkép, aliExpress, Tompa Tamás Posta n..., fordító, ncore, and neptun. The main content area of the browser displays the following text:

## Cookies Name and Value

Deleted cookie : first\_name  
Name : first\_name, Value: John  
Name : last\_name, Value: Player

# Task11: Create a simple webapp

- Create a simple web application that contains
  - a form where the user can enter their name and email address
  - the servlet processes the form and displays the entered data on a new page



The screenshot shows a web browser window with two tabs. The first tab is titled 'localhost:8080/08\_Simple\_webapp/form.html' and displays a form titled 'Data Form'. The form has two input fields: 'Name:' with the value 'Tamas' and 'Email:' with the value 'tamas.tompa1@uni-miskolc.hu'. A blue 'Send' button is at the bottom. The second tab is titled 'localhost:8080/08\_Simple\_webapp/FormServlet' and displays the output of the form, showing the entered data.

**Data:**

Name: Tamas

Email: tomas.tompa1@uni-miskolc.hu



---

Thank you for your attention!

*thank you* 😊