A Java platform comprises the **JVM** together with supporting **class libraries**.

**Java 2 Standard Edition** (**J2SE**)
- (1999) provides core libraries for data structures, XML parsing, security, internationalization, db connectivity, RMI

**Java 2 Platform, Enterprise Edition** (**J2EE**)
- provides more class libraries for servlets, JSPs, Enterprise Java Beans, advanced XML

**Java Platform, Enterprise Edition** (**Java EE**)
- When Java Platform 5.0 was released (2004) the ‘2’ was dropped from these titles.
A Java platform comprises the JVM together with supporting class libraries.

Java Micro Edition (Java ME)
- comprises the necessary core libraries and tools for writing Java for embedded systems and other small footprint platforms, along with some specialised libraries for specific types of device such as mobile phones.
What is a Java Web application?
A **Java web application** generates **interactive web pages** containing various types of markup language (HTML, XML, and so on) and **dynamic content**.

It is typically comprised of web components such as:

- JavaServer Pages (JSP)
- Servlets
- JavaBeans

To **modify** and temporarily **store data**, **interact with databases** and **web services**, and **render content** in response to **client requests**.

https://grizzly.dev.java.net/
What is the Java Enterprise Edition?
Java EE (Enterprise Edition) is a widely used platform containing a set of coordinated technologies that significantly reduce the cost and complexity of:

- developing
- deploying and
- managing

multitier, server-centric applications. Java EE builds upon the Java SE platform and provides a set of APIs (application programming interfaces) for developing and running portable, robust, scalable, reliable and secure server-side applications.

Java EE 6 is supported only by the GlassFish server v3.x.

http://netbeans.org/kb/trails/java-ee.html
The Java EE platform uses a simplified programming model. **XML deployment descriptors** are **optional**. Instead, a developer can simply enter the information as an **annotation** directly into a Java source file, and the **Java EE server** will configure the component at deployment and runtime.

With **annotations**, you put the specification information in your code next to the program element affected.

an architecture for implementing services as multitier applications that deliver the scalability, accessibility, and manageability needed by enterprise-level applications.

With this structure you can more easily change one of the tiers without compromising your entire application.

- **Business and presentation logic** - to be implemented by the developer
- **Standard system services** – to be provided by the Java EE platform

What is a Java Servlet?
Java Servlets

- Servlets are **Java classes** that dynamically process **requests** and construct **responses**.
- Server side replacement for CGI
- Extensions to Java enabled web-servers
- Inherently **multi-threaded**.
- One thread per request.
- Very efficient.
- Platform independent.
How do Servlets work?

- **Servlets** run inside a **Web Container** - the component of the web server that runs and interacts with Servlets.

- **Servlet** is running on the server listening for requests.

- When a **request** comes in, a **new thread** is generated by the **web container**.
What is a Java EE Container?
Java EE Containers

Java EE containers

• are the **interface** between a **Java component** and the **low-level platform-specific functionality** *(i.e. transaction and state management, multithreading, resource pooling, etc.)* that supports the component.

• provide for the separation of **business logic** from **resource** and **lifecycle management**.

• this allows developers to focus on writing business logic rather than writing **enterprise infrastructure**.

The **Java EE platform** uses "**containers**" to simplify development.

http://www.oracle.com/technetwork/java/javaee/javaee-faq-jsp-135209.html#diff
When a request comes in:

- A **Servlet** needs to be instantiated and create a new thread to handle the request.
- Call the Servlet's `doPost()` or `doGet()` method and pass the **HTTP request** and **HTTP response** objects.
- Get the request and the response to the Servlet.
- Manage the life, death and resources of the Servlet.

*All of the above are the tasks of the **web container**.*
Java EE Containers

From Bodoff et. al. 2005
Recall: (PHP-MySQL)  **Server**: response

- Webserver supports HTTP.
Historically (Java Web App)

Server: response

- Webserver supports HTTP.

It’s the Container that gives the Servlet the HTTP request and response, and it’s the Container that calls the Servlet’s methods (e.g. doPost() or doGet()).
• Webserver supports HTTP.

It’s the **Container** that gives the **Servlet** the **HTTP** request **and response**, and it’s the **Container** that calls the **Servlet’s** methods (e.g. `doPost()` or `doGet()`)

Historically (Java Web App)

**Server: response**

- **Web server** supports **HTTP**.
- **Servlet** (Java code)
- **Container** that gives the **Servlet** the **HTTP** request **and response**, and it’s the **Container** that calls the **Servlet’s** methods (e.g. `doPost()` or `doGet()`)
- **Client**
- **Web browser**
- **Operating System**
- **Internet**
- **TCP/IP**
- **Operating System**

**GET...**

```html
<html>
<head>
</head>
<body>
...
</body>
</html>
```
Webserver supports HTTP.

Grizzly is now the **HTTP front end** of the application server.

It’s the **Container** that gives the **Servlet** the **HTTP request and response**, and it’s the **Container** that calls the **Servlet’s methods** (e.g. `doPost()` or `doGet()`).
Java Servlets simplify web development by providing infrastructure for component, communication, and session management in a web container that is integrated with a web server.

- Writing Servlets is like writing Java codes that place an HTML page inside a Java class (this is the worst part of Servlets!)

- (Historically!) requires a deployment descriptor (DD). This is in the form of an XML file.

- Servlets do not have a main() method.
- Servlets are under the control of another Java application called a Container.

http://www.oracle.com/technetwork/java/javaee/javaee-faq-jsp-135209.html#diff
manage the data flow between the following:

<table>
<thead>
<tr>
<th>Client/Database</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>application client or applet</td>
<td>components running on the Java EE server</td>
</tr>
<tr>
<td>database</td>
<td>Server components</td>
</tr>
</tbody>
</table>

JavaBeans components are not considered Java EE components by the Java EE specification.

JavaBeans components have properties and have get and set methods for accessing the properties.
**Enterprise JavaBeans** container handles:

- distributed communication
- threading
- scaling
- transaction management, etc.

has a new packaging! (see figure)
create a simple web application using NetBeans IDE
deploy it to a server, and
view its presentation in a browser
NetBeans

• A 3rd party **Java Integrated Development Environment** (IDE)
  
  • Comes with **Java EE class libraries**
  • bundled with GlassFish Sever Open Source Edition
  • Can deploy servlets, JSPs, and web services

Class libraries for Servlets, JSPs, Enterprise Java Beans, advanced XML
Example: NetBeans Project

A Quick Tour of the IDE (v.6.9)

JSP, Java Bean, User-defined Java Class & Package, Get Method, User Interface
Main interface, Html with form Invokes `response.jsp` through form `action`.

Class `NameHandler` containing user data

Generates the server’s response
Defines a `JavaBean` to connect the class `NameHandler` to the user’s input via a form text field (name).
Creating a new **Web Application**

**New Project / Java Web**

![Screenshot of a new project creation interface](image)

---

**Description:**

*This feature is not yet enabled. Press Next to activate it.*

**Creates an empty Web application** in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.
Creating a new **Web Application**

**Specify Project Name**

[Image of the New Web Application window with the project name set to "HelloWeb"]
Creating a new **Web Application**

**GlassFish Server**

![Image of GlassFish Server configuration](image-url)
GlassFish is an open source application server project led by Sun Microsystems for the Java EE platform. The proprietary version is called Oracle GlassFish Enterprise Server. GlassFish is free software.

It uses a derivative of Apache Tomcat as the servlet container for serving Web content, with an added component called Grizzly which uses Java NIO for scalability and speed.

Before the advent of the Java New I/O API (NIO), thread management issues made it impossible for a server to scale to thousands of users.

Sun is the original creator of Tomcat.
GlassFish is an open source (full) application server project led by Sun Microsystems for the Java EE platform. The proprietary version is called Oracle GlassFish Enterprise Server. GlassFish is free software.

It uses a derivative of Apache Tomcat as the servlet container for serving Web content, with an added component called Grizzly which uses Java NIO for scalability and speed.

On 25 March 2010, soon after the acquisition of Sun Microsystems, Oracle issued a Roadmap for versions 3.0.1, 3.1, 3.2 and 4.0 with themes revolving around clustering, virtualization and integration with Coherence and other Oracle technologies.

http://en.wikipedia.org/wiki/GlassFish
Historically, if you wanted to get good HTTP performance from Tomcat, you really needed to have an Apache web server to sit in front of Tomcat which involved more setting up and extra administrative work.

Since GlassFish v1 (May 2006), Grizzly is the HTTP frontend of the application server.

It's a 100% Java NIO framework that provides the same performance as Apache, only it's written in Java and integrated straight into the application server.
Other Java web application-capable Servers

- **Blazix** from Desiderata Software (1.5 Megabytes, JSP, Servlets and EJBs)
- **TomCat** from Apache (Approx 6 Megabytes)
- **WebLogic** from BEA Systems (Approx 40 Megabytes, JSP, Servlets and EJBs)
- **WebSphere** from IBM (Approx 100 Megabytes, JSP, Servlets and EJBs)

http://www.jsptut.com/Getfamiliar.jsp
Commercial Deployment

• **Oracle GlassFish Server**
  - delivers a flexible, lightweight and extensible Java EE 6 platform. It provides a small footprint, fully featured Java EE application server that is completely supported for commercial deployment and is available as a standalone offering.

• **Oracle WebLogic Server**
  - designed to run the broader portfolio of Oracle Fusion Middleware and large-scale enterprise applications.
  - industry's most comprehensive Java platform for developing, deploying, and integrating enterprise applications.

Creating a new Web Application

JSP File

```html
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```
Creating a new Web Application

Sample Run

Hello World!
HelloWeb: Directories and Files
Adding a Java source package and a source file

NameHandler.java
Java Package

Right-click Source Packages

http://en.wikipedia.org/wiki/GlassFish
Java Package

• a mechanism for organizing **Java classes** into **namespaces**
• can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time.

Add a Java Class, specify Package name

http://en.wikipedia.org/wiki/GlassFish
Java Package

Add a Java Class

```java
package org.mypackage.hello;

/**
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

public class NameHandler {
    
}
```

http://en.wikipedia.org/wiki/GlassFish
• Declare a String variable inside the class declaration.

```java
String name;
```

• Add a constructor to the class:

```java
public NameHandler()
```

• Add the following line in the NameHandler() constructor:

```java
name = null;
```
Generating **Getter and Setter Methods**

Right-click **name field** in the Source editor

Selection: **Name Field / Refactor / Encapsulate Fields**
Generating **Getter and Setter Methods**

Notice that Fields' **Visibility** is by default set to **private**, and Accessors' **Visibility** to public, indicating that the **access modifier** for class variable declaration will be specified as **private**, whereas **getter** and **setter** methods will be generated with **public** and **private** modifiers, respectively.
Generating Getter and Setter Methods

Select the **Refactor** button.
Notice that the variable declaration has changed.
• set to **private**

Get and set functions with implementation have been added as well.
• access modifier: **public**
Editing the Default JSP file

Adding and Customising a Form, input text field, submit button
Invoke the **palette**: from the menu, select (Window/Palette): or press Ctrl+Shift+8

**Inserting a Form**

- Expand HTML Forms
expand HTML Forms and drag a **Form item** to a point after the `<h1>` tags in the Source Editor.

The Insert Form dialog box displays.
Specify the following values:

Action: response.jsp
Method: GET
Encoding: application/x-www-form-urlencoded
Name: Name Input Form

Click OK.
An HTML form is automatically added to the index.jsp file.

```html
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <form name="Name Input Form" action="response.jsp">
    
    </form>
  </body>
</html>
```
Drag a Text Input item to a point just before the </form> tag, then specify the following values:

- **Name**: name
- **Type**: text
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <form name="Name Input Form" action="response.jsp">
      <input type="text" name="name" value="" />
    </form>
  </body>
</html>
Drag a Button item to a point just before the </form> tag. Specify the following values:

- **Label**: OK
- **Type**: submit

Click OK. An HTML button is added between the <form> tags.
Adding some extra labels, tidying up your code

Type **Enter your name:** just before the first `<input>` tag, then change the default Hello World! text between the `<h1>` tags to **Entry Form**.

Right-click within the Source Editor and choose Format (Alt-Shift-F) to tidy the format of your code.
index.jsp: Source Generated

```html
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h1>Hello World!</h1>
<form name="Name Input Form" action="response.jsp">
  Enter your name: <input type="text" name="name" value="" />
  <input type="submit" value="Ok" />
</form>
</body>
</html>
```

We would like to pass this to our server.
Creating a JSP file that generates the server's response

response.jsp
In the Projects window, right-click the **HelloWeb** project node and choose **New > JSP**. The New JSP File wizard opens.

Name the file **response**, and **click Finish**.

Notice that a response.jsp file node displays in the Projects window beneath index.jsp, and the new file opens in the **Source Editor**.
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
In the **Palette** to the right of the Source Editor, expand **JSP** and drag a **Use Bean** item to a point just below the `<body>` tag in the Source Editor.

The **Insert Use Bean dialog** opens. Specify the values shown in the following figure.

The **class** `NameHandler` belongs to the **package** we have set earlier.
Notice that the `<jsp:useBean>` tag is added beneath the `<body>` tag.
Drag a Set Bean Property item from the Palette to a point just before the <h1> tag and click OK.

In the <jsp:setProperty> tag that appears, delete the empty value attribute and edit as follows. Delete the value = "" attribute if the IDE created it! Otherwise, it overwrites the value for name that you pass in index.jsp.
Drag a **Set Bean Property item** from the Palette to a point just before the `<h1>` tag and click OK.

In the `<jsp:setProperty>` tag that appears, delete the empty value attribute and edit as follows. Delete the `value = ""` attribute if the IDE created it! Otherwise, it overwrites the value for name that you pass in index.jsp.
Adding a Get Bean property item

Drag a **Get Bean Property** item from the **Palette** and drop it after the comma between the `<h1>` tags.
Specify the following values in the Insert Get Bean Property dialog:

- **Bean Name:** mybean
- **Property Name:** name

Insert a **Get Bean Property** item here!
the user input coming from index.jsp becomes a name/value pair that is passed to the request object.

When you set a property using the `<jsp:setProperty>` tag, you can specify the value according to the name of a property contained in the request object.

Therefore, by setting property to name, you can retrieve the value specified by user input.
Sample Run

Hello World!

Enter your name: napoleon

Response from the JSP file

Hello, napoleon!
Sample Run

Index.jsp
Main interface, Html with form
Invokes response.jsp through form action.

NameHandler.java
Class NameHandler
containing user data, get and set methods

response.jsp
Generates the server’s response
Defines a JavaBean to connect the class NameHandler to the user’s input via a form text field (name).
Main interface, Html with form invokes `response.jsp` through form action.

Class `NameHandler` containing user data, get and set methods.

Generates the server's response.

Defines a JavaBean to connect the class `NameHandler` to the user's input via a form text field (name).

The Java EE specification defines how the web application can be archived into a **web application archive (WAR)**

- **WAR files** are
  - Java archives with a `.war extension`
  - Packaged using the same specification as zip files
  - Understood by all Java EE compliant application servers

- WAR files can be directly deployed in servlet containers such as Tomcat
To make a WAR for your NetBeans project, right click on the project node and select **Build Project**.

The WAR file will be placed in the “dist” sub-directory of your project folder.
Java EE 6

http://download.oracle.com/javaee/6/tutorial/doc/

Recommended Directory Structure for Projects


NetBeans

http://netbeans.org/kb/docs/web/quickstart-webapps.html

Simple Database Example

http://netbeans.org/kb/docs/web/mysql-webapp.html

E-Commerce Example

http://netbeans.org/kb/docs/javaee/ecommerce/design.html
http://netbeans.org/kb/docs/javaee/ecommerce/data-model.html#createERDiagram
Welcome to the online home of the Affable Bean Green Grocer.

Our unique home delivery service brings you fresh organic produce, dairy, meats, breads and other delicious and healthy items direct to your doorstep.
The Affable Bean

Bakery

Dairy

Meats

Bakery

Fruit & Veg

Sunflower Seed Loaf
600g
€1.89
ADD TO CART

Sesame Seed Bagel
4 Bagels
€1.19
ADD TO CART

Pumpkin Seed Bun
4 Buns
€1.15
ADD TO CART

Chocolate Cookies
contain peanuts
(3 cookies)
€2.39
ADD TO CART
Your shopping cart contains 4 items.

- clear cart
- continue shopping
- proceed to checkout

**subtotal: € 6.87**

<table>
<thead>
<tr>
<th>product</th>
<th>name</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Image: sesame seed bagel</a></td>
<td>sesame seed bagel</td>
<td>€ 1.19 (€ 1.19 / unit)</td>
<td>1</td>
</tr>
<tr>
<td><a href="#">Image: chocolate cookies</a></td>
<td>chocolate cookies</td>
<td>€ 2.39 (€ 2.39 / unit)</td>
<td>1</td>
</tr>
<tr>
<td><a href="#">Image: corn on the cob</a></td>
<td>corn on the cob</td>
<td>€ 1.59 (€ 1.59 / unit)</td>
<td>1</td>
</tr>
<tr>
<td><a href="#">Image: milk</a></td>
<td>milk</td>
<td>€ 1.70 (€ 1.70 / unit)</td>
<td>1</td>
</tr>
</tbody>
</table>
Model-View-Controller Paradigm

**Model**
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes

**View**
- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view

**Controller**
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

= Method Invocations
= Events