**CLASE 12**

**Qué es JSP?**

Significa Java Server Pages. Se utiliza como la capa de presentación de una aplicación Web.

Es como a una pagina Web al cual puedo agregarle código Java.

**Relación con Servlets**

Los servlets fueron la primer propuesta de Sun Microsystems (ahora Oracle) para programación Web. Los servlets podían responder con HTML pero resultaba muy incómodo. En java 1.2 surgen los JSP y pueden trabajar en conjunto con los servlets.

Los archivos JSP se convierten en clases (servlets). El Servlet Container se encarga de realizar la conversión del JSP a una clase, y su posterior compilación.

**JSP Scripts**

**Scriptlets**

Son bloques de código encerrados por <% %>. JSP se construye a través de scriptlets.

Es posible escribir los atributos y métodos que formaran parte del Servlet a partir del archivo JSP. El cual se va embebiendo con HTML.

**Directiva page**

Entre sus atributos más conocidos se encuentran: import, pageEncoding, isErrorPage, errorPage, contentType, buffer, autoFlush, isThreadSafe, session, etc.

Ej:

<% @page import="java.util.\*,java.text.\*" %>

**Directiva Include**

Incluye el contenido de un archivo dentro de otro. Ej, HTML, JSP, otros.

Ej:

[code]

<HTML>

<BODY>

A continuación se incluye el archive hola.jsp<BR>

<iframe>

<% @include file="hola.jsp" %>

</iframe>

</BODY>

</HTML>

[/code]

**PATRON MODELO – CONTROLADOR – VISTA**

Es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

**Modelo:** Son todas aquellas clases Java que representan el negocio. Ej: Persona, Empleado, etc

**Vista:**  Interfaz de usuario. Paginas JSP. HTML, CSS y JS

**Controlador:** Intermediario entre **Vista** y **Modelo.** Recibe los request de la vista y conoce que clase del Modelo puede responder. Con el response del Modelo decide a que JSP debe dirigirse con dicho response.

HTML => JSP => SERVLET => MODELO (CLASE)=>BBDD

El **modelo** representa todas las clases de Java que son mi modelo de Negocio, Ej: autos, modelos, facturación, empleados, etc.

La **vista** implica todo lo que es la parte visual=> HTML o JSP.

Y el **controlador** serian los servlets. **Reciben las peticiones de la vista y saben cuales son las clases que satisfacen ese request.**

Ej: Si quiero dar de alta un empleado, seguro voy a tener un HTML con un formulario de Alta y se va a comunicar con un Servlet AltaEmpleado quien se va a comunicar con EmpleadoDAO para dar de alta el empleado en la Base. Luego dará una respuesta si el Empleado fue insertado correctamente o algún tipo de error. El modelo le responderá al Servlet y el Servlet responderá a la Vista.

**Mostramos como se crearon los diferentes Archivos en el Servidor Tomcat:**

1. Vamos a **localhost:8080.**
2. hostManager.
3. Estado del Servidor=> Listar Aplicaciones.
4. Vemos todas las páginas que fuimos ejecutando hacia el servidor.

EJERCICIO **CLASE\_12\_WEB\_1 => ECLIPSE**

1. Crear un Proyecto Maven Web en Eclipse => <https://decodigo.com/java-crear-proyectos-maven-en-netbeans-y-eclipse>

Arquetipo: org.apache.maven.archetypes => Maven-archetype-webapp

1. Windows=>show view=>Navigator=>settings=> org.eclipse.wst.common.project.facet.core.xml => cambiamos a la versión 1.8 de java y también la versión de servlet que vamos a utilizar a la 4.0

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

<faceted-project>

<fixed facet=*"wst.jsdt.web"*/>

<installed facet=*"wst.jsdt.web"* version=*"3.0"*/>

<installed facet=*"java"* version=*"1.8"*/>

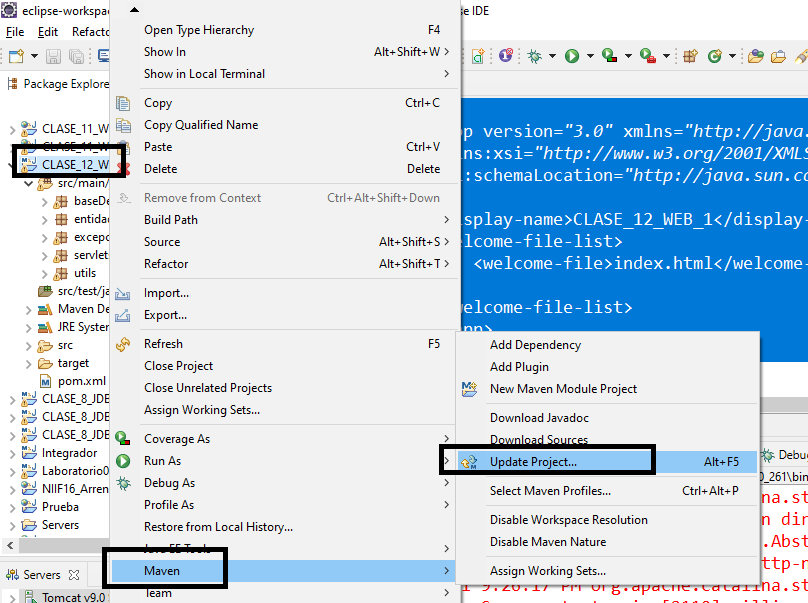
<installed facet=*"jst.jaxrs"* version=*"3.0"*/>

<installed facet=*"jst.jsf"* version=*"2.2"*/>

<installed facet=*"jst.web"* version=*"4.0"*/>

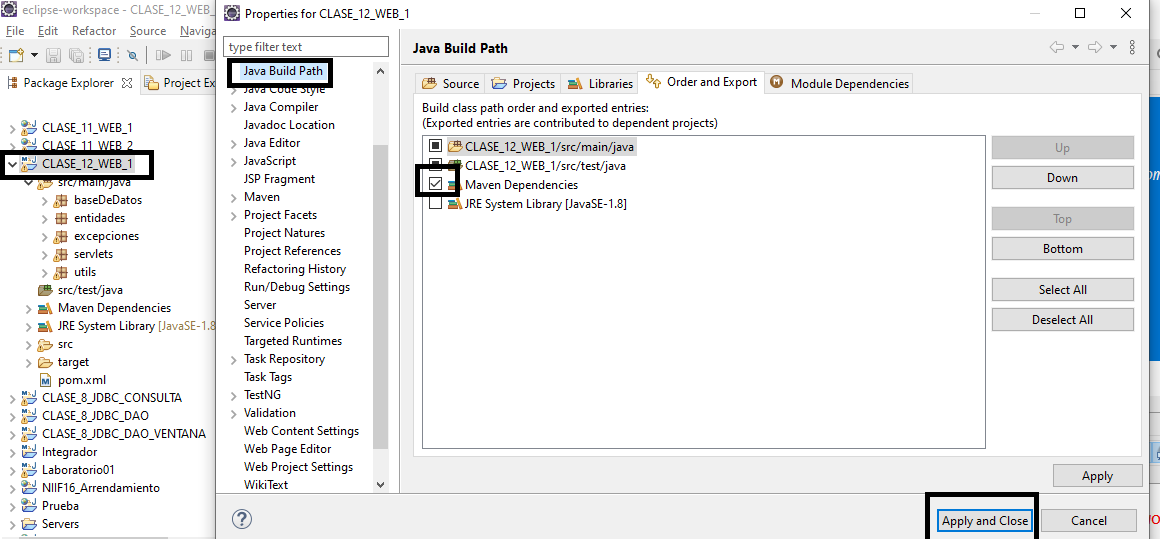
</faceted-project>

1. Guardamos=>Boton derecho sobre el proyecto => Maven => Update Project



1. Configuramos el **build path** para Maven:

Tildamos la opción **Maven Dependencies**.



1. **pom.xml** (agregamos dependencias).

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

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>CLASE\_12\_WEB\_1</groupId>

<artifactId>CLASE\_12\_WEB\_1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>CLASE\_12\_WEB\_1 Maven Webapp</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.19</version>

</dependency>

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-web-api</artifactId>

<version>8.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<finalName>CLASE\_12\_WEB\_1</finalName>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.1.0</version>

</plugin>

<!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.1</version>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.2.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.5.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.8.2</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

1. **web.xml**:

<web-app version=*"3.0"* 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\_3\_0.xsd"*>

<display-name>CLASE\_12\_WEB\_1</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

1. sobre el proyecto=> botón derecho => run as => Run on server
2. Codigo versión 1



1. Como harían para cuando Modifica solucionar el escenario en donde el usuario no ingrese “Descripcion del producto” o “Nuevo precio” o ambos:

***Ejemplo 1:*** Código: **Clase\_11\_b**



JSP es un complemento para los servlets y se utiliza en la capa de presentación (capa visual).

En los ejemplos pasados vimos como generar HTML a partir de Strings que intervienen en el request-response. Esto es algo inviable cuando se trata de páginas sumamente complejas. Porque el contenido y el formato cambia todo el tiempo.

Para el ejercicio tomar el tutorial.jsp del alumni para obtener el servlet nada mas.

Luego realizar los siguiente pasos:

1. Crear un Java Web Project desde cero.
2. Crear el Servlet llamado “PersonaServlet” y copiarle el siguiente código al método **doPost()**:

//se obtiene el nombre y apellido con el request.getParameter()

String nombre = request.getParameter("nombre");

String apellido = request.getParameter("apellido");

//esto es nuevo. Un request.setAttribute() permite

//darle un valor agregado a la proxima pagina. A traves

//del seteo de un atributo

request.setAttribute("nombreCompleto", nombre + ", " + apellido);

//el setAttribute() trabaja igual que un mapa

//setAttribute(key, value)

//Le envio el request a mi JSP el cual ya tiene un valor agregado.

//Desde el JSP voy a tener que hacer un getAttribute de "nombreCompleto"

//para obtener el valor de nombre y de apellido

request.getRequestDispatcher("bienvenido.jsp").forward(request, response);

1. En **index.html** colocar lo siguiente:

<!DOCTYPE html>

<html>

<head>

<title>Ingreso Persona</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

</head>

<body>

<section>

<h2 style="color:blue">www.educacionIT.com.ar</h2>

<hr>

</section>

<h1>Bienvenido</h1>

<br>

<h3>Ingrese los datos de la persona: </h3>

<br>

<form action="PersonaServlet" method="post">

Nombre: <input type="text" name="nombre" /><br>

Apellido: <input type="text" name="apellido" /><br>

Domicilio: <input type="text" name="domicilio" /><br>

Pais: <input type="text" name="pais" /><br>

<input type="submit" value="Enviar">

</form>

</body>

</html>

1. Crear un archivo JSP. Nos paramos sobre “Web Pages”=> New => JSP file. Llamado “**bienvenido**”.

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@page import="java.util.Date"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="windows-1252">

<title>JSP Page</title>

</head>

<body>

Bienvenido <%=request.getAttribute("nombreCompleto")%>

<!--Hace un getAttribute de la key-->

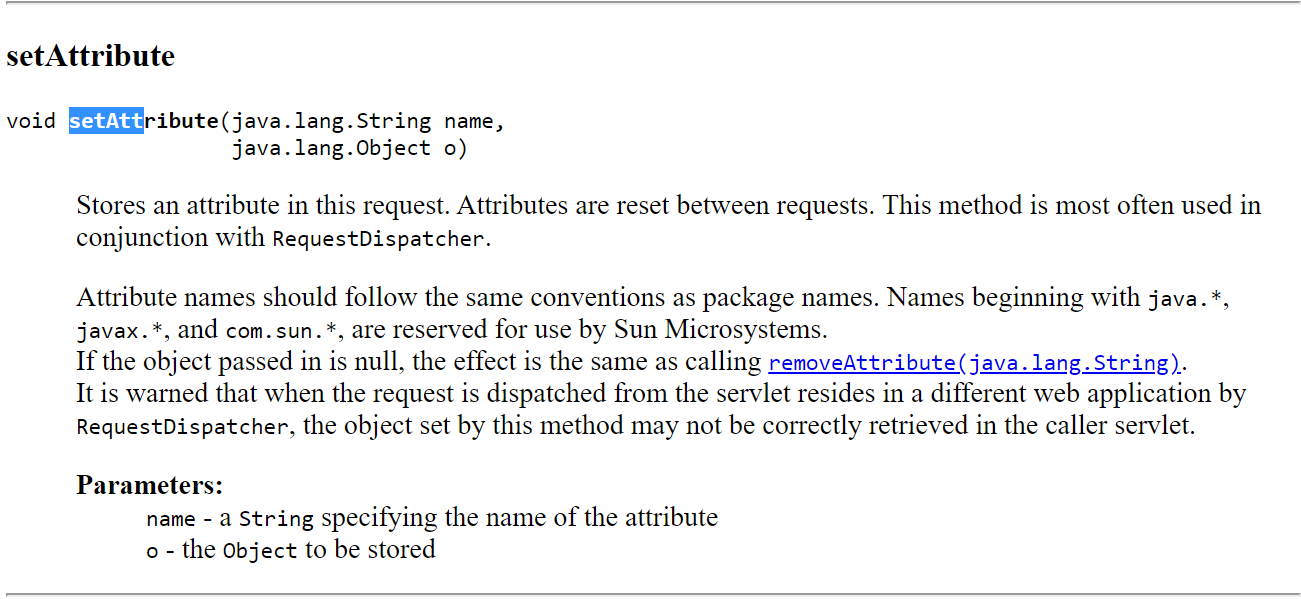
<br>

Son las <%=new Date()%>

</body>

</html>

Set Attribute=> Interfaz **ServletRequest**



**SCRIPTLESS:**

Hace referencia al código encerrados por <% %>. Me permiten la creación de paginas dinámicas. Por ejemplo: eBay y MercadoLibre. Donde tengo varios productos, lleno formularios, clickeo varios links.

Los JSP son ideales para este tipo de paginas donde el contenido depende de la interacción del usuario y la pagina cambia constantemente.

**Tipos de Declaraciones en Scriptless**

## **<%=Se va a mostrar al usuario(expresión); %>**

## **<%! declaración de variable o método; %>**

## **<%@ Directivas page(importar paquete) e include(incluir otra pagina); %>**

***Ejemplo 2:*** Código: **Clase\_11\_c**

****

Tipo de Scriptless **Expresion**

1. Creamos un “Java Web Project”
2. En “WebContent” creamos un JSP file llamado **Inicial**.
3. Ingresamos el siguiente código:

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Inicial</title>

</head>

<body>

<%= new java.util.Date() %>

<!--Este tipo de scriptless se llama expresión.

puedo crear también una instancia de persona=> new Persona();

El “=” es algo asi como el SOUT cuando estoy en JEE. Permite mostrarle algo al usuario

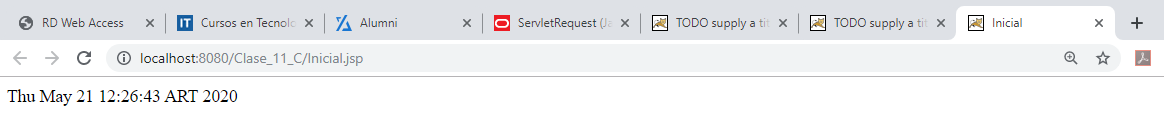
-->

</body>

</html>

1. Luego nos posicionamos sobre el archivo creado, botón derecho, Run File.

Vemos como sale la fecha actual en base al código Java escrito.



***Ejemplo 3:*** Código: **Clase\_11\_d**

**Tipo de ScriptLess “Declaracion” y “Directiva”**

1. Creamos un nuevo Proyecto Java Web.
2. Posicionados sobre **Web Pages** crear un archivo JSP llamado **Inicial\_1**.
3. Colocamos el siguiente código:

<%@page import="java.util.Date"%>

<!--Este tipo de Scriptless se llama DIRECTIVA. Me permite hacer

configuraciones. En este caso importamos el paquete java.util.Date

Esto evita que tenga que usar el Scriptless de Expresion para incluir

el paquete de esa otra forma.

-->

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Inicial</title>

</head>

<body>

<p>Primera forma de obtener la fecha y hora actual </p>

<%= new java.util.Date() %>

<%!

Date fechaActual = new Date();

Date obtenerFecha(){

return fechaActual;

}

%>

<br>

<p>Segunda forma de obtener la fecha y hora actual </p>

<p> La fecha actual es: <%=obtenerFecha()%></p>

<!--Llamo al metodo obtenerFecha() desde un tag de HTML.

<!--Este tipo de Scriptless se llama Declaracion.

Esto significa que voy a poder crear Atributos y Metodos

dentro de mi JSP-->

</body>

</html>

Los scriptlets del tipo **declaración** son los que permiten incluir atributos y métodos dentro de un **JSP.** Conceptualmente no es del todo correcto incluir atributos y métodos dentro del JSP. Un JSP es una clase Java, internamente es un SERVLET. Al compilarlo Java lo convierte en SERVLET. No es recomendable hacer scriptlets del tipo declaración. Si se pueden plantear scriptlets del tipo expresión o directivas.

Ej de directivas: import, encoding, contentType, include (incluir una pagina dentro de otra).

Ej:

<@ include file=”otraPagina.jsp” %> => evita que tenga que hacer el header, main, section, acticle, footer, etc.

**JSP** = Java Standard + uso de objetos implicitos (out, response, request, etc)

Servlet es un objeto implícito que ya está instanciado y listo para ser utilizado

Por cada JSP hay un Servlet atrás, lo genera automáticamente el Tomcat.

**TOMCAT => HTML +JSP + SERVLET + CLASES**

**Esto da lugar a diferentes "tiempos":**

Tiempo de Construcción: cuando el programador codifica el archivo JSP

Tiempo de Traducción: cuando se ejecuta el archivo .jsp el tomcat parsea y traduce el jsp a un servlet.

Tiempo de Compilación: ya traducido el jsp al servlet, el tomcat lo compila.

Tiempo de Ejecución: si bien parece que ejecuta el jsp, esta ejecutando en servlet que respalda al jsp.

***Ejercicio Integrador:*** Código: **IntegradorClase11**

Consignas:



