educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 1 - Java Analyst

Módulo 10

educación IT
CODE YOUR FUTURE

Java EE

Introducción

Java proporciona una **plataforma de desarrollo empresarial** robusta y tiene una gran presencia en el mercado laboral.

Java Enterprise Edition (Java EE) se basa en la especificación Java SE (utiliza Java SE y añaden más funcionalidades).

En la infraestructura de *Java EE*, se suman **reglas en dos niveles**:

- En la **capa de la aplicación**, para gestionar la lógica empresarial dinámica y el flujo de tareas.
- En la **capa de presentación**, para personalizar el flujo de páginas y el flujo de trabajo y para construir páginas personalizadas basándose en el estado de la sesión.

Proporciona una **plataforma para construir aplicaciones** *transaccionales*, *seguras*, *interoperables y distribuidas* con los siguientes servicios:

Básicos

- Creación de páginas web con `Servlets` y *Java Server Pages* (JSP).
- Acceso a bases de datos relacionales con Java *Database Connectivity* (JDBC).
- Sistema de mapeo objeto-relacional con *Java Persistence API* (JPA).

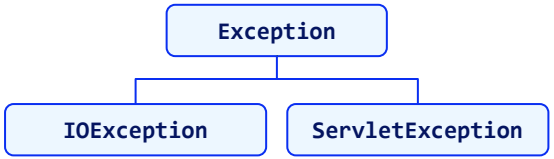
Avanzados

- Gestión de componentes con *Enterprise Java Beans* (EJB).
- Interfaz de usuario con *Java Server Faces* (JSF).
- Gestionar transacciones con *Java Transaction API* (JTA).
- Envío y recepción de mensajes con Java *Message Service* (JMS).

Excepciones

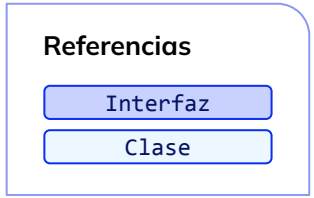
La excepción **`IOException`** se encargará de **lanzar errores cuando exista un problema al tratar de leer un fichero**. Puede ocurrir porque se encuentra bloqueado o porque se interrumpió la comunicación.

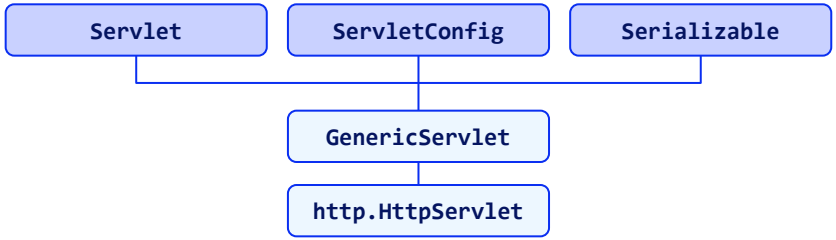
La excepción **`ServletException`** se encargará de lanzar un error si encuentra un problema.



Paquete javax.servlet

Contiene una serie de clases e interfaces que describen y definen los contratos entre una clase de **`servlet`** y el entorno de ejecución proporcionado para una instancia de dicha clase por un contenedor de `servlet`.





Interfaz Servlet

Define métodos que deben implementar todos los `servlets`.

Interfaz ServletConfig

Es utilizada por un contenedor de `servlet` para pasar información a un servlet durante la inicialización.

Interfaz Serializable

Se encarga de transformar los objetos de Java en bytes para transmitirlos por la red o persistirlos.

Clase abstracta GenericServlet

Define un `servlet` genérico independiente del protocolo.

Clase abstracta HttpServlets

Provee métodos para responder a solicitudes HTTP en el contexto de una aplicación web.

Servlets

Para poder manejar verbos *HTTP* debemos crear una clase que extienda directamente de `HttpServlets` y sobreescribir los métodos que vayamos a permitir en nuestra clase.

Servlets

Tipo	Método	Descripción
`protected void`	**`doDelete(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes `DELETE`.
`protected void`	**`doGet(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes `GET`.
`protected void`	**`doHead(HttpServletRequest req, HttpServletResponse resp)`**	Recibe una solicitud `HTTP HEAD` de la solicitud.
`protected void`	**`doOptions(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes de `OPTIONS`.
`protected void`	**`doPost(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes `POST`.
`protected void`	**`doPut(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes `PUT`.
`protected void`	**`doTrace(HttpServletRequest req, HttpServletResponse resp)`**	Permite a través de este servicio que el `servlet` maneje solicitudes `TRACE`.

HttpServletRequest

Esta interfaz nos provee información de la solicitud para `servlets` HTTP.

Tipo	Método	Descripción
`Object`	**`getAttribute(String name)`**	Devuelve el valor del atributo nombrado como un `Object`, o `null` si no existe ningún atributo del nombre dado.
`String`	**`getParameter(String name)`**	Devuelve el valor de un parámetro de solicitud como a `String`, o `null` si el parámetro no existe.
`RequestDispatcher`	**`getRequestDispatcher(String path)`**	Devuelve un `RequestDispatcher` objeto que actúa como contenedor del recurso ubicado en la ruta dada.
`void`	**`setAttribute(String name, Object o)`**	Almacena un atributo en esta solicitud.
`HttpSession`	**`getSession()`**	Devuelve la sesión actual asociada con esta solicitud, o si la solicitud no tiene una sesión, crea una.

HttpServletResponse

Esta interfaz proporciona una funcionalidad específica de HTTP al enviar respuestas a los clientes.

Tipo	Método	Descripción
`void`	**`sendRedirect(String location)`**	Envía una respuesta de redireccionamiento temporal al cliente utilizando la URL de ubicación de redireccionamiento especificada y borra el `búfer`.
`PrintWriter`	**`getWriter()`**	Devuelve un `PrintWriter` objeto que puede enviar texto de caracteres al cliente.

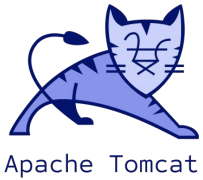
Contenedor Web Java

Java EE también define un modelo de contenedor, que aloja y gestiona instancias de componentes de aplicaciones *Java EE*.

Los contenedores están a su vez alojados en servidores *Java EE* y se encargarán de redireccionar las peticiones a un objeto `Servlet` en específico.

Existen **dos tipos de servidores**:

- **Servidores de aplicaciones**: Contienen todos los servicios definidos en el estándar *Java EE*.
- **Servidores web**: Contienen los servicios esenciales de *Java EE* (`servlets` y JSP).



Descriptores de implementación

Describen las clases, recursos y configuración de la aplicación y la forma en que los usa el servidor web para entregar solicitudes web. Cuando el servidor web recibe una solicitud para la aplicación, usa el descriptor de implementación a fin de asignar la URL de la solicitud al código que debe manejarla.

El descriptor de implementación es un archivo llamado **`web.xml`**. Se encuentra en el `WAR` de la app en el directorio **`WEB-INF/`**. El archivo es un archivo `XML` cuyo elemento raíz es **`<web-app>`**.

El directorio `WEB-INF` contiene todas las cosas relacionadas con la aplicación que no están en la raíz del documento de la aplicación.

El `WEB-INF` no forma parte del árbol de documentos públicos de la aplicación, por lo que ningún archivo contenido en él puede ser servido directamente a un cliente por el contenedor, sin embargo, el contenido es visible para el código de `Servlet`.

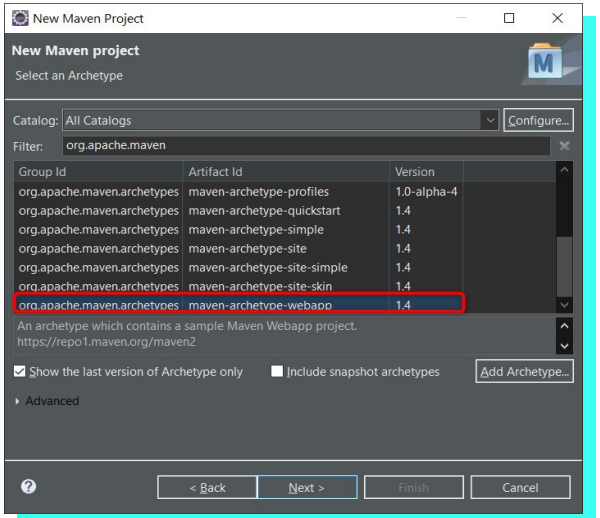
Ejemplo

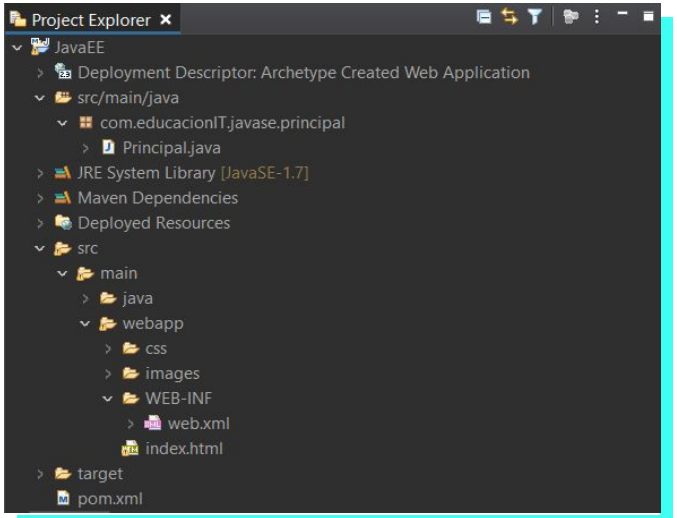
```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
    <servlet-name>Principal</servlet-name>
    <display-name>Principal</display-name>
    <description></description>
    <servlet-class>com.educacionIT.javase.principal.Principal</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Principal</servlet-name>
    <url-pattern>/Principal</url-pattern>
  </servlet-mapping>
</web-app>
```

Arquetipo de Maven Webapp

`Maven-archetype-webapp` es un arquetipo que genera un proyecto de aplicación web.





Librerías adicionales

Al trabajar con Java EE resulta necesario agregar un **`jar`** al proyecto que contiene las clases necesarias que se utilizan para **"dialogar" con los verbos HTTP.**

Recordemos que los proyectos están creados con un arquetipo de Maven y a través del archivo **`pom.xml`** podemos **gestionar las librerías** que necesitamos.

```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

MVNrepository.com

Seleccionamos la versión, copiamos la dependencia y la pegamos en el pom.

¡Sigamos trabajando!