



Certified



Corporation®

APLICACIONES WEB DINÁMICAS CON JAVA

Informática

Noviembre de 2021



INTRODUCCIÓN

[

]

APLICACIONES WEB DINÁMICAS CON JAVA

- Java Enterprise Edition o también conocido como Java EE, es un conjunto de estándares de tecnologías dedicadas al desarrollo de aplicaciones del lado del servidor. Esta plataforma consta de un conjunto de servicios, API y protocolos que proporcionan la funcionalidad necesaria para desarrollar aplicaciones basadas en web de varios niveles, es decir, se pueden desarrollar aplicaciones empresariales distribuidas, con arquitecturas multicapa, escritas en Java y que se ejecutan en un servidor de aplicaciones.
- Java es un lenguaje orientado a objetos de alto nivel y, sin duda, es uno de grandes lenguajes de programación de los que se disponen, con una alta demanda en el mercado laboral.
- El presente documento te permitirá internalizar la potencialidad del lenguaje, mostrando su estructura a través de ejemplos que te llevarán al más alto nivel en el uso de esta plataforma

JAVA SERVLET Y JSP

[

]

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción a los servlets

Competencias

- Conocer la teoría detrás de los servlets
- Conocer el ciclo de Vida de los servlets
- Aplicar ejemplo de uso del ciclo de vida

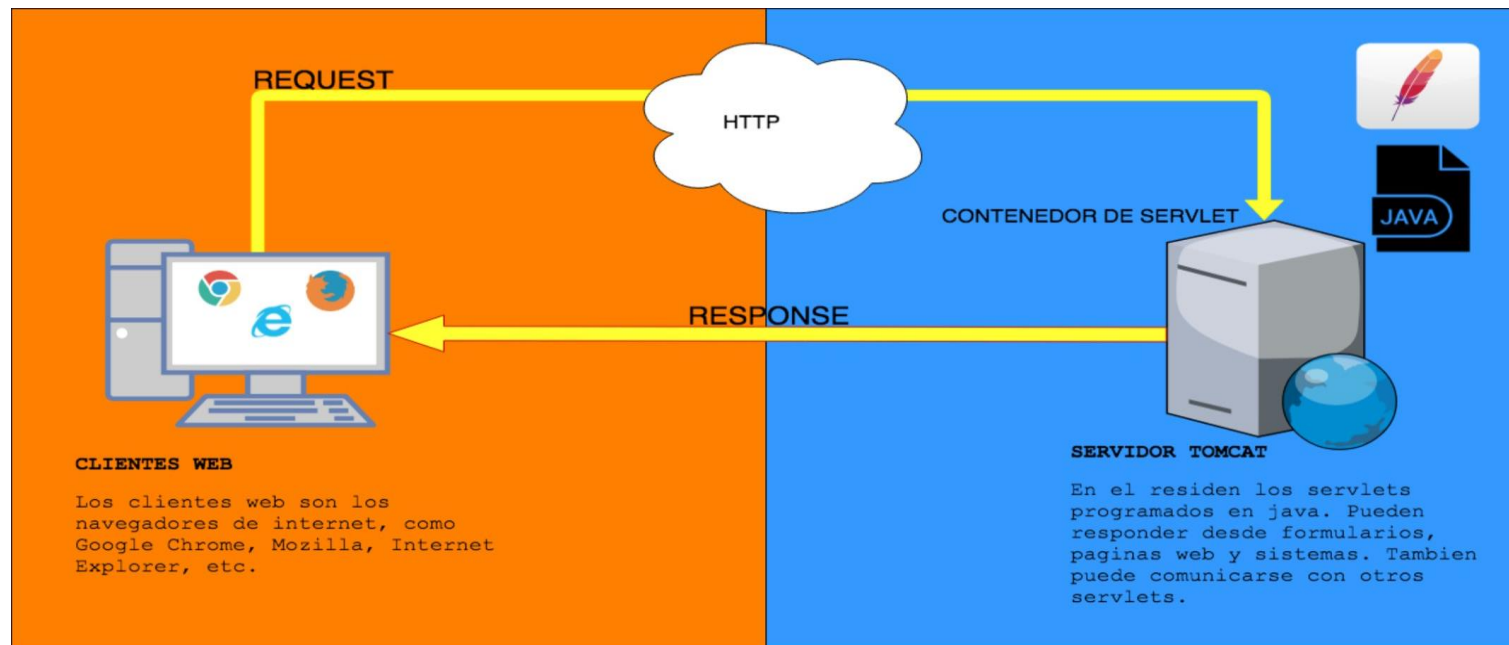
APLICACIONES WEB DINÁMICAS CON JAVA

Los servlets

- Un servlet no es más que una clase java que trabaja dentro de un contenedor de servlets como Apache Tomcat, el cual es el encargado de extender la funcionalidad de la clase y darle más "poder" por decirlo de una forma.
- El servlet fuera del contenedor Tomcat pierde todo su poder y se convierte en un archivo de texto más, con extensión java imposible de ejecutar, ya que esta clase no cuenta con un método "Main" que ejecute la rutina.
- La siguiente imagen muestra el flujo normal de petición entre clientes (navegador web) y servidor (contenedor de servlets, aquí están los servlets). Esta es la arquitectura típica de un sistema web, la cual es conocida como cliente-servidor

APLICACIONES WEB DINÁMICAS CON JAVA

Flujo cliente web a servlet



APLICACIONES WEB DINÁMICAS CON JAVA

Ciclo de vida de un Servlet

El ciclo de vida completo de un servlet es administrado por el contenedor de servlet y, antes de empezar a programarlos, hay que entender cómo funciona la creación y el funcionamiento de estos elementos de software.

El ciclo de vida se divide en 4 pasos bien definidos:

- Carga del servlet
- Inicialización del servlet
- Captura del request
- Destrucción el request

APLICACIONES WEB DINÁMICAS CON JAVA

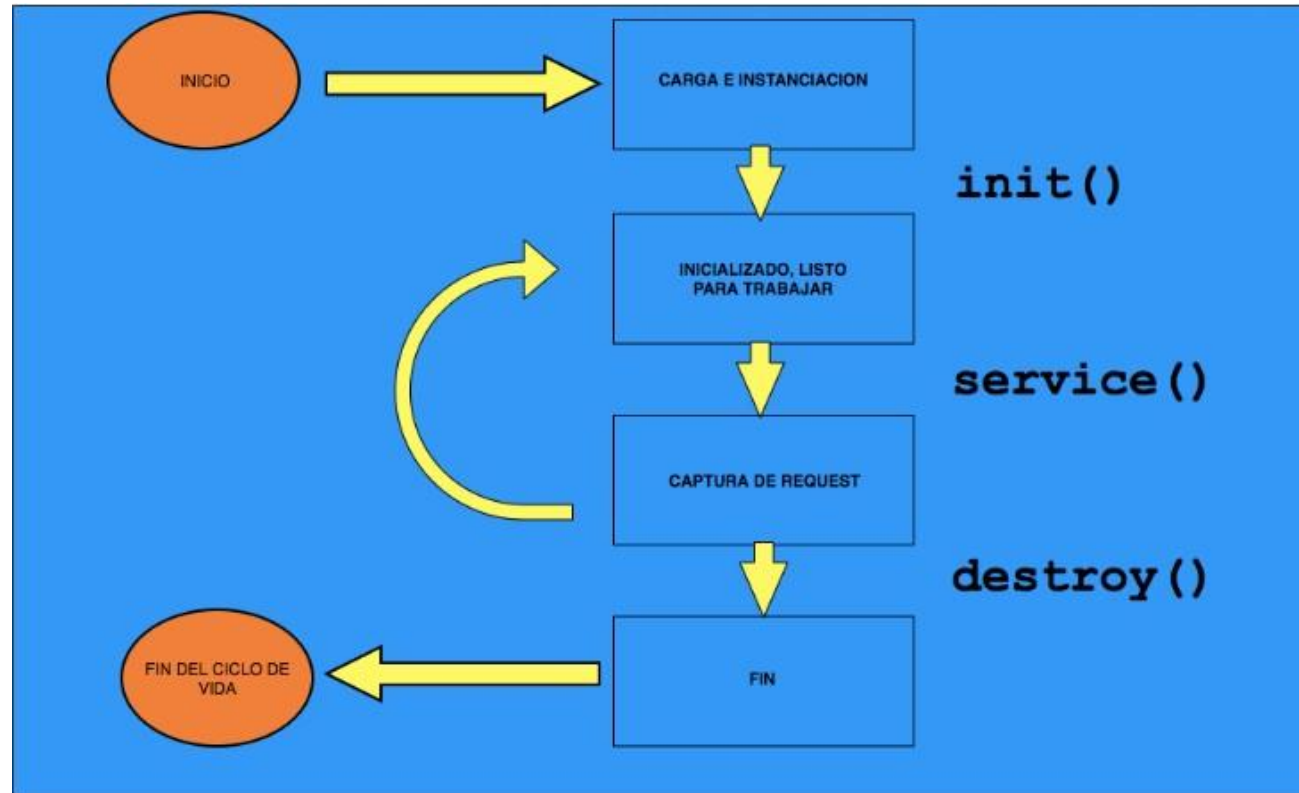
Carga del servlet

La primera etapa del ciclo de vida de un servlet parte con su inicialización gracias al servidor Tomcat. El servidor ejecuta dos pasos en esta etapa:

- **Loading:** Carga de la clase servlet
- **Instantiation:** Creación de una instancia del servlet. Para crear una nueva instancia del servlet, el contenedor utiliza un constructor sin parámetros.

APLICACIONES WEB DINÁMICAS CON JAVA

Ciclo de vida de un Servlet



APLICACIONES WEB DINÁMICAS CON JAVA

Inicialización de Servlets

- Después de que el servlet fue instanciado correctamente, el contenedor se encarga inicializarlo.
- Ésto lo realiza invocando al método “Servlet.init()”, que acepta como referencia del objeto “ServletConfig” como parámetro.

Captura del Request

- Después de la inicialización, la instancia del servlet está lista para atender los request del cliente.

APLICACIONES WEB DINÁMICAS CON JAVA

UnavailableException, e IOExceptionDestruir el Servlets

Destruir el Servlets

- Cuando el contenedor decide destruir el servlet, ejecuta las siguientes acciones:
- Permite que todos los subprocesos que se ejecutan actualmente en el método de servicio de la instancia de Servlet completen sus trabajos y sean liberados.
- Después de que los procesos en ejecución hayan completado sus trabajos, el contenedor Servlet
- Llamar al método: “destroy()”

APLICACIONES WEB DINÁMICAS CON JAVA

Creación del primer servlet

Competencias

- Entender cómo se compone un servlet.
- Compilación de un servlet
- Probar servlets en el contenedor
- Conocer los métodos de los servlets

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

En una arquitectura web, la comunicación entre cliente y servidor es crucial para un funcionamiento colaborativo, y dentro de la tecnología JEE un servlet es uno de los principales actores. Ya estudiamos que es un servlet y cuál es su ciclo de vida, por lo que ahora es tiempo de empezar a implementarlos para descubrir poco a poco cuáles son sus funciones y que nos permiten hacer.

Composición de un servlets

Los servlets se construyen basados en la herencia de su clase padre `Http Servlets`, clase pública y abstracta que permite que una clase herede los métodos y atributos que hacen que se convierta en un servlets. La clase que herede de esta clase abstracta, debe sobrescribir al menos:

- Método `doGet`
- Método `doPost`
- Método `doDelete`

APLICACIONES WEB DINÁMICAS CON JAVA

Creación de un servlet

- Creamos un nuevo proyecto de tipo Dinamic Web Project y nombramos al mismo como PrimerServlet.
- Avanzamos en el wizard con next y finish. Por el momento dejamos las opciones de web.xml por defecto.
- Al final Eclipse nos genera una estructura de Proyecto, la cual usaremos como base para nuestro trabajo.

Por el momento la carpeta importante para nosotros es la Java Resources, que contendrá el código fuente de nuestro programa. Se compone de:

- La carpeta src, que aloja las clases java y la estructura de paquetes de la aplicación.
 - La carpeta libraries, que mantiene las librerías utilitarias.
 - La carpeta build, que mantiene las clases autogeneradas.
- Luego de generar el Proyecto, vamos a crear nuestra primera clase. Para hacer esta labor, tenemos que seleccionar el nombre del proyecto con botón derecho, y presionar new, class.

APLICACIONES WEB DINÁMICAS CON JAVA

Heredando de HttpServlet

```
package com.desafiolatam.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/logout")
public class Deslogueo extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        // Se crea variable de sesion y se recibe desde el request
        HttpSession session = request.getSession();
        PrintWriter out = response.getWriter();
        //generacion de alerta que indica el cierre de sesion
        out.println("<script type='text/javascript'>");
        out.println("alert('A cerrado sesion correctamente');");
        out.println("</script>");
        //el metodo invalidate destruye la sesion
        session.invalidate();
        //el metodo sendRedirect redirige al formulario de ingreso
        response.sendRedirect("index.jsp");
        return;
    }
}
```


APLICACIONES WEB DINÁMICAS CON JAVA

Manejo de información entre servlet

Competencias:

- Entender el funcionamiento de métodos getParams
- Enviar parámetros desde el cliente
- Aplicar el envío de información entre servlets

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

- Los servlet no solamente se encargan de devolver recursos a los clientes que lo solicitan, también pueden comunicarse con otras clases servlets para compartir información y articular el flujo correcto de un sistema web. Esta capacidad es una gran razón del por qué esta tecnología es tan demandada y veremos la forma en la cual estas clases pueden comunicarse entre sí compartiendo sus variables y dando funcionalidad al sistema.

Creación de servlet de generación respuesta

- Los servlets tienen la capacidad de recibir parámetros desde los clientes, para luego procesarlos y generar salidas. El objeto request es el encargado de manejar tales parámetros y gracias a él podemos acceder a los datos enviados. Para entender el funcionamiento de esta característica vamos a implementar una nueva funcionalidad en nuestro primer ejemplo.
- El objetivo del ejercicio es permitir que el servlet obtenga dos parámetros enviados por nosotros y que luego nos devuelva un saludo con nuestro nombre, apellido y la fecha actual.

APLICACIONES WEB DINÁMICAS CON JAVA

Enviando parámetros a través de GeneradorRespuesta

```
@WebServlet("/generadorRespuesta")
public class GeneradorRespuesta extends HttpServlet {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {

        String nombre;
        String apellido;
        String fechaActual;
        PrintWriter printWriter = response.getWriter();
        SimpleDateFormat formato = new SimpleDateFormat("dd-MM-yyyy");
        fechaActual = formato.format(new Date());
        nombre = request.getParameter("nombre");
        apellido = request.getParameter("apellido");

        printWriter.println("<html>");
        printWriter.println("<body>");
        printWriter.println("Bienvenido/a " + nombre + " " + apellido);
        printWriter.println("La fecha es: " + fechaActual); printWriter.println("</body>");
        printWriter.println("</html>");

    }
}
```

El código mostrado corresponde a una nueva clase de nombre GeneradorRespuesta

APLICACIONES WEB DINÁMICAS CON JAVA

Creando el servlet:

- La clase debe extender de HttpServlet
- La clase debe implementar algún método del padre, en nuestro caso doGet
- La clase debe ir acompañada de la anotación

Si analizamos el código, la mayoría sigue igual, a excepción de las dos variables de tipo String que declaramos y su asignación. En cada variable String (nombre, apellido) estamos guardando el valor del request mediante:

```
nombre = request.getParameter("nombre");  
apellido = request.getParameter("apellido");
```

APLICACIONES WEB DINÁMICAS CON JAVA

La variable request, es el parmetro de entrada de nuestro método:

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

- En estas variables van los parámetros que el cliente envía al servlet.
- La variable request posee el método “getParameter()” y, además de él tiene otros métodos más, que veremos más adelante
- Con esas sentencias ya tenemos en nuestro poder el nombre y el apellido que vienen desde el cliente, y podemos hacer lo que queramos con ellas. En este caso solo vamos a mostrar un saludo con la fecha.



APLICACIONES WEB DINÁMICAS CON JAVA

Aplicar el envío de información entre Servlets

Si bien el flujo normal entre un cliente servidor es desde navegador a servidor, hay ocasiones en que se requiere que en la capa de negocio un servlet pueda comunicarse con otro servlet. Para lograr este objetivo, java nos provee de los métodos:

- **getServletContext:** método otorgado por HttpServletRequest, que lo que hace es obtener el contexto del servlet que queremos utilizar.
- **getRequestDispatcher:** luego de obtener el contexto, le enviamos la dirección del servlet.

```
getRequestDispatcher()
```

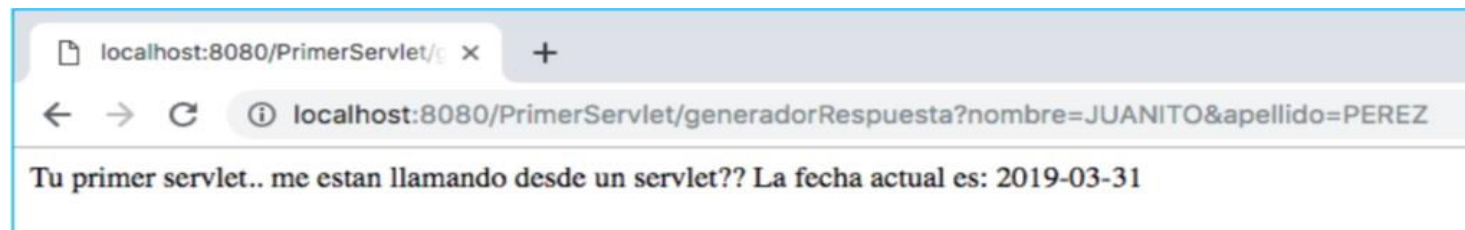
```
getServletContext()
```

APLICACIONES WEB DINÁMICAS CON JAVA

Envío de información entre Servlets

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
    PrintWriter writer = response.getWriter();  
    String formatoFecha = "yyyy-MM-dd";  
    SimpleDateFormat formato = new SimpleDateFormat(formatoFecha); String fechaActual = formato.format(new Date());  
  
    writer.println("<html><body>");  
    writer.println("Tu primer servlet ");  
    writer.println("La fecha actual es: "+fechaActual);writer.println("<body><html>");  
    request.getRequestDispatcher("/inicio").forward(request, response);  
}
```

Probando el Servlet .



APLICACIONES WEB DINÁMICAS CON JAVA

Introducción a los Formularios y JTSL con JSP:

Competencias:

- Entender la función de un formulario Introducción a la tecnología JSP y JSTL
- Instalación de librerías JSTL
- Aplicar envío de variables entre formularios

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

- Los formularios con JSP están hechos con html puro, lo que significa que no debemos saber nada extra para generar formularios. Lo que sí hay que tener en cuenta es el mecanismo para enviar información y recibirla entre distintos archivos jsp.

¿Qué es un formulario web?

- Pensemos un momento en una página web, con estilos y estructuras que permiten al usuario enterarse de los productos o servicios que una empresa puede ofrecer. Tal página es solo descriptiva ya que solamente se encarga de desplegar información y mostrarla de manera ordenada y atractiva.

APLICACIONES WEB DINÁMICAS CON JAVA

Los JSP y JSTL

- **JSP:** Java Server Page, más conocido simplemente como jsp, es una tecnología del stack Java JEE que se encarga de proveer herramientas para que los desarrolladores puedan construir páginas web generadas dinámicamente basadas en el lenguaje de marcado HTML, XML u otro tipo de documentos.
- Si analizamos la arquitectura, JSP está basada en el funcionamiento de los servlets, ya que, al ser ejecutados, las páginas JSP se convierten internamente en servlets en tiempo de ejecución, por lo tanto se podría decir que una página JSP es en realidad un servlet. Una página JSP se puede utilizar utilizando dos mecánicas:
- Como componente de vista de un diseño del lado del servidor, trabajando codo a codo con clases Java Bean como el modelo y con servlets como controlador. Posteriormente profundizaremos en la arquitectura cliente servidor. Como un componente independiente

APLICACIONES WEB DINÁMICAS CON JAVA

JSTL:

- Estos elementos tienen como misión proporcionar una manera fácil de mantener los componentes de una página JSP.
- El uso de estas etiquetas ha simplificado la tarea de los diseñadores para crear páginas web.
- Ahora simplemente usan la etiqueta relacionada con la tarea que deben hacer en una página JSP.

Las características más significativas de los JSTL son:

- JSTL también es JSP, siendo un conjunto complementario de este.
- Utiliza 4 librerías estándar: SQL, XML, CORE, INTERNALIZACIÓN JSTL define un nuevo lenguaje de expresiones llamado EL.
- Al usar una etiqueta JSTL, lo que hacemos es añadir una acción.
- Una etiqueta JSTL está delimitada por `${ }`

APLICACIONES WEB DINÁMICAS CON JAVA

Un ejemplo de JSTL:

```
primerUsoJstl.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3 <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
4 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
5 <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="UTF-8">
10 <title>Usando JSTL</title>
11 </head>
12 <body>
13 <p>Cadena de caracteres: <strong><c:out value="1+2+3"/></strong></p>
14 </body>
15 </html>
```

Resultado:

Usando JSTL

http://localhost:8080/JstlEjemplo1/primerUsoJstl.jsp

Cadena de caracteres: **1+2+3**

APLICACIONES WEB DINÁMICAS CON JAVA

Tags JSTL

- Para referenciar la librería JSTL Core en una página JSP debemos declarar la cabecera del documento como a continuación:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

JSTL incluye una gran variedad de tags que engloban distintas áreas funcionales. Para segmentar estas áreas, se utilizan namespaces.

JSTL expone múltiples tags en estas url:

- **Core:** <http://java.sun.com/jsp/jstl/core>
- **XML:** <http://java.sun.com/jsp/jstl/xml>
- **Internationalization:** <http://java.sun.com/jsp/jstl/fmt>
- **SQL:** <http://java.sun.com/jsp/jstl/sql>
- **Functions:** <http://java.sun.com/jsp/jstl/functions>

APLICACIONES WEB DINÁMICAS CON JAVA

Tabla con los múltiples tags disponibles

Area	Subfunction	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
I18N	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

PATRONES DE DISEÑO

[

]

APLICACIONES WEB DINÁMICAS CON JAVA

Java Server Page (JSP)

Competencias:

- Conocer la tecnología JSP
- Beneficios para los desarrolladores
- Entender diferencias entre JSP v/s Servlets
- Hola mundo JSP
- Elementos JSP Scriptlets, Expresiones y Declaraciones.

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

- En el mundo del desarrollo JEE, la capa de cliente es un pilar fundamental que sustenta el uso prolongado de los clientes hacia nuestras aplicaciones.
- En esta capa trabaja la tecnología JSP (Java Server Page) la cual provee al programador una mecánica de trabajo ágil, rápida simple para presentar datos al usuario.
- En esta unidad se estudiarán los fundamentos de esta tecnología, sus características y por sobre todo su uso en el desarrollo de sistemas empresariales.

APLICACIONES WEB DINÁMICAS CON JAVA

Conociendo la tecnología JSP

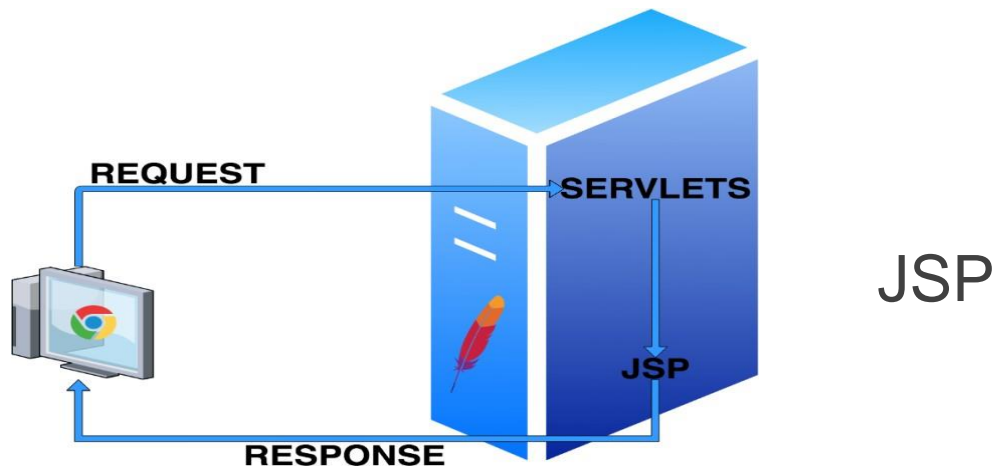
- Java Server Page (JSP) es la tecnología que provee una vía rápida y simple para crear contenido web dinámico.
- Provee herramientas para que los desarrolladores puedan construir páginas web generadas dinámicamente basadas en el lenguaje de marcado HTML, XML u otro tipo de documentos.
- Fue lanzado en el año 1999 por Sun Microsystems.
- JSP tiene muchas similitudes con otras tecnologías como lo son PHP o ASP, pero implementando el lenguaje de programación Java.

APLICACIONES WEB DINÁMICAS CON JAVA

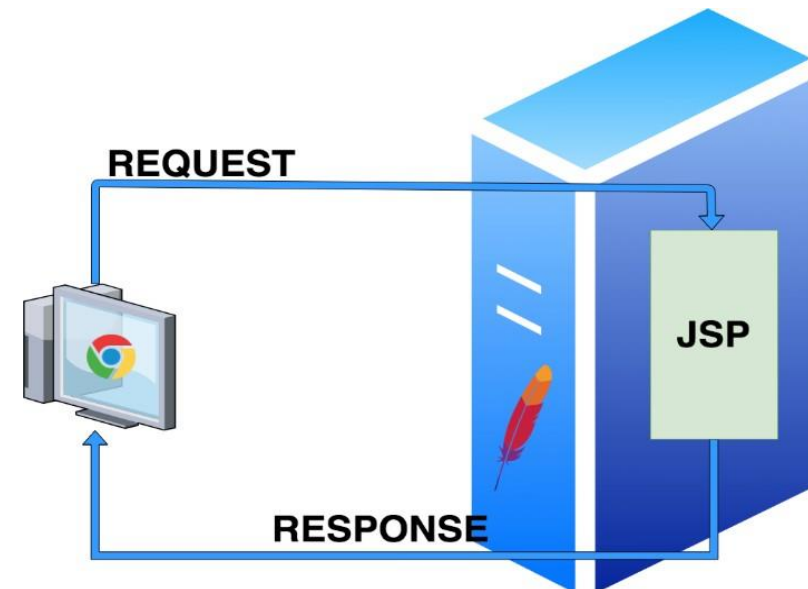
Una página JSP se puede construir utilizando dos mecánicas:

- Como componente de vista de un diseño del lado del servidor, trabajando codo a codo con clases Java Bean como el modelo y con servlets como controlador. En capítulos posteriores profundizaremos en la arquitectura cliente servidor.
- Como un componente independiente.

APLICACIONES WEB DINÁMICAS CON JAVA



Flujo de trabajo JSP:



APLICACIONES WEB DINÁMICAS CON JAVA

Beneficios para los desarrolladores

- Las ventajas de utilizar JSP en una aplicación empresarial para los desarrolladores se pueden enumerar en:
- JSP no es solo una página HTML, sino que es una completa implementación de todas las características de un servlets. En una página JSP se pueden utilizar todas las funcionalidades de la biblioteca de clases java, además de usar tags de funcionalidades y desplegar información al usuario.
- Con JSP es mucho más fácil hacer la separación entre código de programación java y código de presentación HTML (En servlets, generar código html es difícil).
- Usando JSP ya no es necesario recompilar y construir el proyecto cada vez que se haga un cambio. En términos prácticos esto significa que, si modificas un título a tu JSP, basta con guardar el cambio y refrescar el navegador. Esto aumenta la rapidez de desarrollo en comparación con los servlets.
- Con JSP se utiliza menos código para hacer cosas similares que en un servlet.

APLICACIONES WEB DINÁMICAS CON JAVA

JSP vs Servlets

- Estos dos componentes de la tecnología java, generan contenido dinámicamente desde un servidor con contenedor de servlets a una página web, pero de una manera distinta.
- Ambas tecnologías se complementan entre sí, así que no es posible elegir un camino u otro, ya que juntas alcanzan su mayor potencial. La mayor diferencia que encontramos es que un servlet es un programa java simple con una herencia de clases característica (HttpServlet) que se incrusta en una página web simple.
- El programa java se ejecuta en el lado del servidor y muestra el resultado en el navegador incrustado html que se envía al navegador. El servlet incrusta HTML en el código java a través de las declaraciones **out.println**

APLICACIONES WEB DINÁMICAS CON JAVA

- **Relación entre JSP y Servlets**
- Por regla general, las páginas JSP y los archivos Servlets están muy relacionados entre sí. Cada JSP se compila primero en un servlet antes de poder hacer cualquier cosa. Cuando se utiliza una primera llamada a JSP, se traduce al código fuente de java servlet y luego, utilizando el compilador, se compila en un archivo de clase Servlet, convirtiéndose en la salida para el cliente.
- Las grandes ventajas de una página JSP vs un Servlet son las siguientes:
- Los servlets utilizan sentencias de impresión para imprimir un documento HTML, que resulta muy difícil de leer y mantener. JSP no tiene esta característica y mejora sustancialmente la legibilidad del código.

APLICACIONES WEB DINÁMICAS CON JAVA

Relación entre JSP y Servlets

- JSP no necesita de compilación, ni configuración de variables de entorno ni tampoco de empaquetado.
- En una página JSP, el contenido visual y la lógica están separados, lo cual con un servlet no se puede lograr.
- Con jsp contamos con la implementación automática. Cuando se genera un cambio en el código no es necesario volver a compilar o construir el elemento, a diferencia de los servlets que al más mínimo cambio es necesario recompilar el código y por ende el servidor debe levantar nuevamente.

.

APLICACIONES WEB DINÁMICAS CON JAVA

El “Hola Mundo” con JSP

Una página JSP en esencia es una página con código HTML, pero tiene una pequeña diferencia en su declaración. Si observamos la imagen veremos que:

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <meta charset="UTF-8">
6   <title>Hola Mundo JSP</title>
7 </head>
8 <body>
9
10 </body>
11 </html>
```

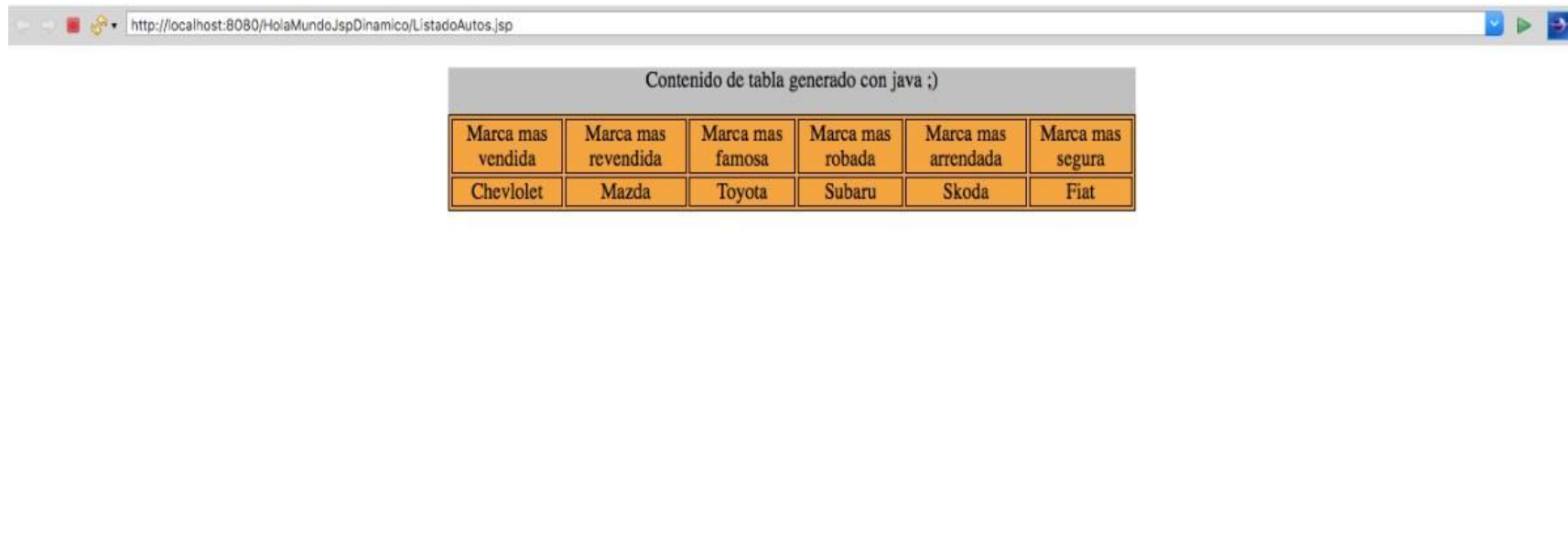
APLICACIONES WEB DINÁMICAS CON JAVA

Otro ejemplo JSP: Listado de autos (listadoAutos.jsp): usando scriptlets (`<% %>`)

```
26 <body>
27 <%
28     //declaracion de una lista de marcas de vehiculos
29 List<String> autos = new ArrayList<String>();
30 autos.add("Chevrolet");
31 autos.add("Mazda");
32 autos.add("Toyota");
33 autos.add("Subaru");
34 autos.add("Skoda");
35 autos.add("Fiat");
36 %>
37 <!-- comienza el HTML -->
38
39 <div id="contenido">
40 <p>Contenido de tabla generado con java ;)</p>
41 <table>
42 <thead>
43 <tr>
44 <td>Marca mas vendida</td>
45 <td>Marca mas revendida</td>
46 <td>Marca mas famosa</td>
47 <td>Marca mas robada</td>
48 <td>Marca mas arrendada</td>
49 <td>Marca mas segura</td>
50 </tr>
51 </thead>
52 <tbody>
53 <tr>
54 <%
55     for(int i = 0; i < autos.size(); i++){
56         out.println("<td>" + autos.get(i)+ "</td>");
57     }
58 %>
59 </tr>
60 </tbody>
61 </table>
62 </div>
63 </body>
```

APLICACIONES WEB DINÁMICAS CON JAVA

El resultado de la clase:



A screenshot of a web browser window. The address bar shows the URL: `http://localhost:8080/HolaMundoJspDinamico/ListadoAutos.jsp`. The page content displays a table with the title "Contenido de tabla generado con java :)" in a grey header. The table has two rows and six columns. The first row contains the following text: "Marca mas vendida", "Marca mas revendida", "Marca mas famosa", "Marca mas robada", "Marca mas arrendada", and "Marca mas segura". The second row contains the following text: "Chevrolet", "Mazda", "Toyota", "Subaru", "Skoda", and "Fiat".

Contenido de tabla generado con java :)					
Marca mas vendida	Marca mas revendida	Marca mas famosa	Marca mas robada	Marca mas arrendada	Marca mas segura
Chevrolet	Mazda	Toyota	Subaru	Skoda	Fiat

APLICACIONES WEB DINÁMICAS CON JAVA

Uso de formularios para captura de información

Competencias:

- Conocer el concepto de formulario
- Uso de controles html de formularios
- Estructura de un formulario
- Comunicación de JSP a JSP Conocer los objetos implícitos
-

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

- Como se ha estudiado en capítulos anteriores, una página JSP es capaz de manejar lógica de negocio ya sea mediante código nativo java entre scriptlet y también utilizando la librería JSTL.
- El elemento principal que permite que un usuario alimente de datos a un sistema web son los formularios, los cuales están diseñados para capturar la información del usuario y enviarlos al servidor para su posterior procesamiento.

Formulario JSP a formulario JSP

- En una arquitectura empresarial, es común la comunicación entre entidades del mismo tipo, ya sea para generar un flujo de trabajo en donde la información solo deba pasar de página a página o definitivamente procesar la información hasta almacenarlos en una base de datos.
- En el caso de la tecnología JSP es posible transferir información desde dos páginas java server page. Veremos a base de ejemplos prácticos los mecanismos para empezar la comunicación.

.

APLICACIONES WEB DINÁMICAS CON JAVA

Un poco de teoría

Un formulario está compuesto por componentes HTML que tienen como misión capturar información ingresada por el usuario. Algunos de estos controles son:

·
Input Text

`type=text`

Nombre

Contraseña

Input Text

`type=password`

APLICACIONES WEB DINÁMICAS CON JAVA

CheckBox

type=checkbox

Puestos de trabajo buscados

☐ Dirección

☐ Técnico

☐ Empleado

Radio Button

type=radio

☒ RadioButton1

☐ RadioButton2

☐ RadioButton3

Botón de envío de formulario

type=submit

La mayoría de formularios dispone de un botón para enviar al servidor los datos introducidos por el usuario:

Buscar

APLICACIONES WEB DINÁMICAS CON JAVA

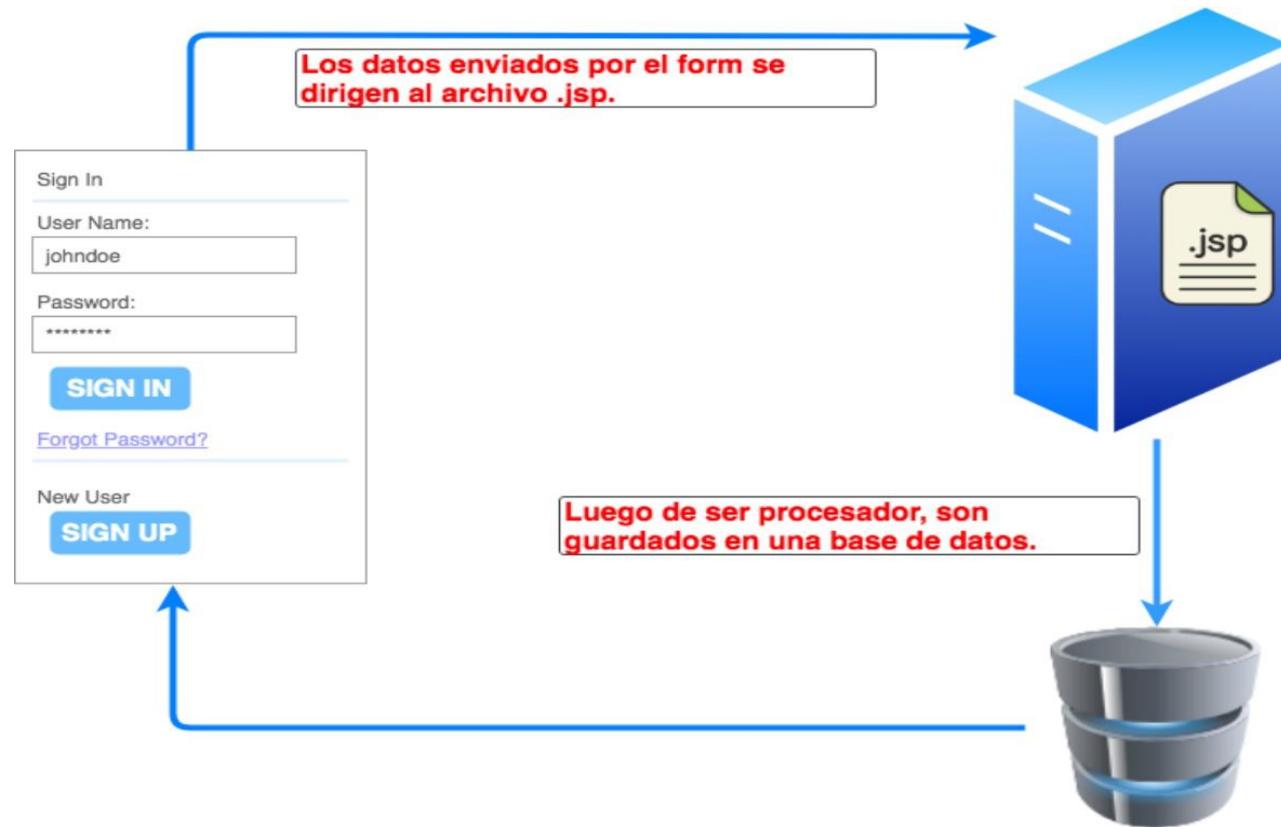
Estructura de un formulario

- La etiqueta de inicio de un formulario se presenta a continuación:

```
<form action="proceso.jsp" method="POST">  
  
</form>
```


APLICACIONES WEB DINÁMICAS CON JAVA

Flujo entre formulario y servidor.



APLICACIONES WEB DINÁMICAS CON JAVA

Comunicación de JSP a JSP

- Se crearán dos archivos Java Server Page, uno encargado de recopilar los datos que ingrese el usuario y otro que los capture y despliegue en la página.

▪



Formulario de entrada de datos:

Usuario

Comentario

APLICACIONES WEB DINÁMICAS CON JAVA

Comunicación de JSP a JSP

- El código de la anterior página:

```
92- <form action="validacion.jsp" method="POST">
93-   <div>
94-       <label>Usuario</label>
95-       <input type="text" id="user" name="user">
96-   </div>
97-
98-   <div>
99-       <label>Comentario</label>
100-       <textarea id="comentario" name="comentario"></textarea>
101-   </div>
102-   <div id="boton">
103-       <button type="submit" class="btn btn-primary">Enviar</button>
104-   </div>
105- </form>
```

APLICACIONES WEB DINÁMICAS CON JAVA

Comunicación de JSP a JSP

- El código de validación:

```
86     <div class="container" style="border:1px solid black;height: 600px;">
87         <h2>Recepcion de datos desde Contacto.jsp</h2>
88         <%
89             String nombre = request.getParameter("user");
90             String comentario = request.getParameter("comentario");
91         %>
92         <form action="Contacto.jsp" method="POST">
93             <div>
94                 <label>Usuario</label>
95                 <input type="text" id="user" value="<%out.println(nombre);%>">
96             </div>
97             <div>
98                 <label>Comentario</label>
99                 <textarea id="comentario"><%out.println(comentario);%></textarea>
100            </div>
101            <div id="boton">
102                <button type="submit" class="btn btn-secondary">Volver</button>
103            </div>
104        </form>
105    </div>
---
```

APLICACIONES WEB DINÁMICAS CON JAVA

El objeto **request** es el utilizado para obtener los valores enviados desde el formulario por lo tanto hay que entender que es y cómo se utiliza. A

Estas variables son llamadas implícitas porque pueden ser utilizadas en cualquier lugar de las páginas JSP con tan solo llamarlas, y además de la variable **request** existen un conjunto de ellas las cuales son:

- **Request**: método de petición para indicar la acción que se desea realizar para un recurso determinado
- **Response**: Respuesta generada por el servidor.
- **out**: Representa el flujo de salida del cuerpo de la respuesta HTTP
- **session** : Permite mantener una sesión para cada uno de los usuarios conectados a la aplicación Web
- **config**: Contiene información relativa a la configuración del servlet generado
- **exception**: Excepción que se ha producido en una página JSP

APLICACIONES WEB DINÁMICAS CON JAVA

El objeto request

- Las páginas JSP son componentes web que responden a los request mediante el protocolo HTTP. El objeto implícito **request** es quien representa este mensaje que se envía desde el navegador y puede contener la información de los valores enviados desde el cliente además de información relevante de las cabeceras del protocolo. Cuando un navegador envía información al servidor lo hace a través del objeto request el cual viaja por la red mediante la url del sitio.
- Existen dos formas de transferencia de request mediante la url:
- **Url encoded parameters:**
- **Form encoded parameters**

APLICACIONES WEB DINÁMICAS CON JAVA

Modelo Vista Controlador (MVC) – Arquitectura en Capas Competencias

- Conocer el patrón de diseño MVC
- Diferencias entre el modelo de capas vs mvc
- Componentes del modelo MVC
- Arquitectura en capas Backend y Frontend

APLICACIONES WEB DINÁMICAS CON JAVA

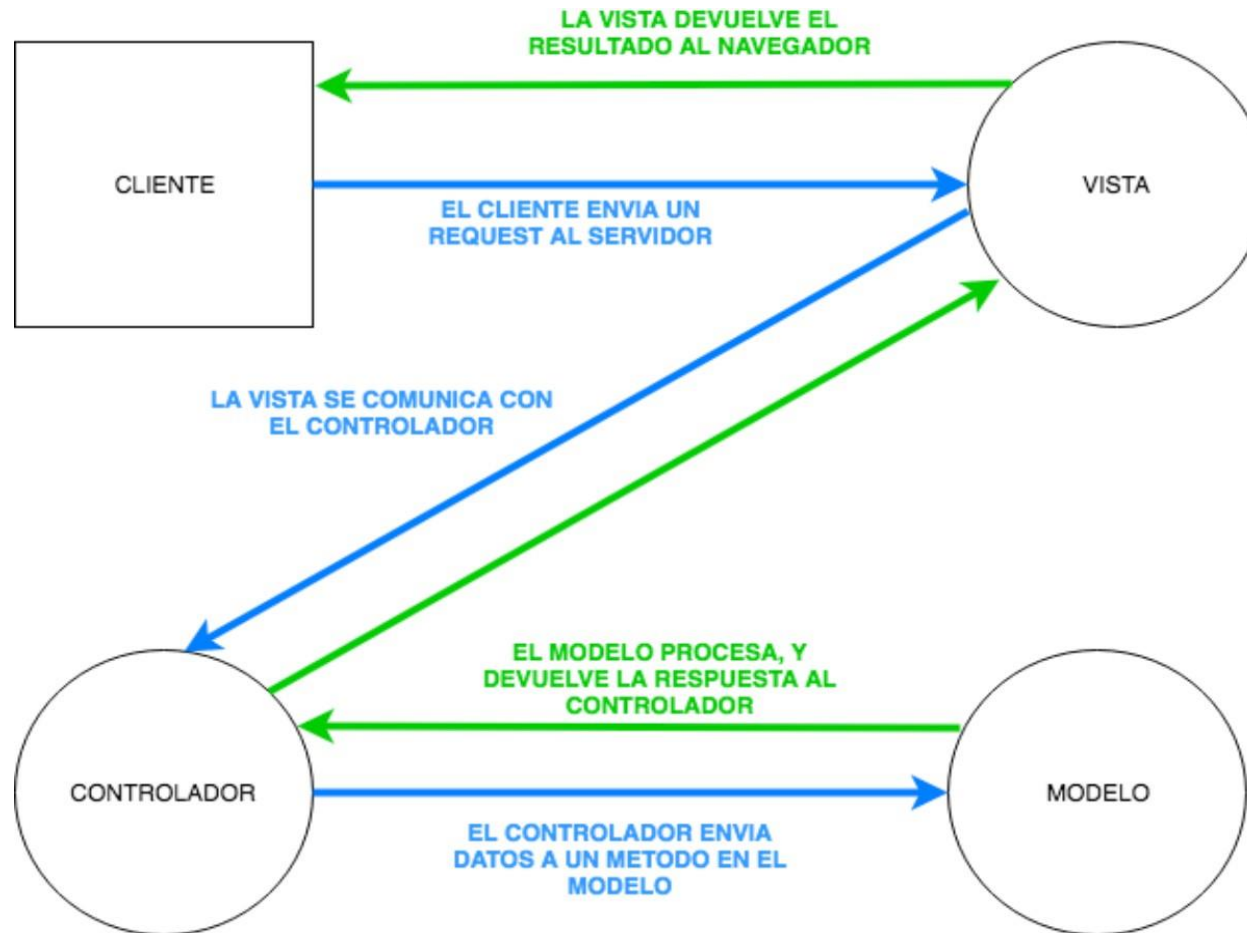
Componentes del modelo MVC

En una aplicación MVC, los componentes de una aplicación son divididos en tres categorías:

- **Modelo:** para explicar esta capa, primero tenemos que saber que, al programar con java, se utiliza el paradigma orientado a objetos, donde existen entidades expresadas en código que asemejan a las entidades del mundo real.
- **Vista:** En esta capa es donde se ubica todo lo que se mostrará al usuario
- **Controlador:** Intermediario que se comunica con la capa de vista y también con la capa de modelo.

APLICACIONES WEB DINÁMICAS CON JAVA

Diagrama de alto nivel: MVC



APLICACIONES WEB DINÁMICAS CON JAVA

Arquitectura en capas

- En estas arquitecturas cada elemento del sistema se describe y se contiene de acuerdo a su responsabilidad en capas muy bien definidas. En los proyectos empresariales con java que se pueden encontrar en distintas entidades como por ejemplo grandes bancos, tiendas de retail y minería por ejemplo se utilizan estas capas para estructurar los sistemas. A continuación, describiremos estas capas.
- **Capa de datos:** Encargada de trabajar con los datos del sistema, comunicando directamente con algún sistema de base de datos.

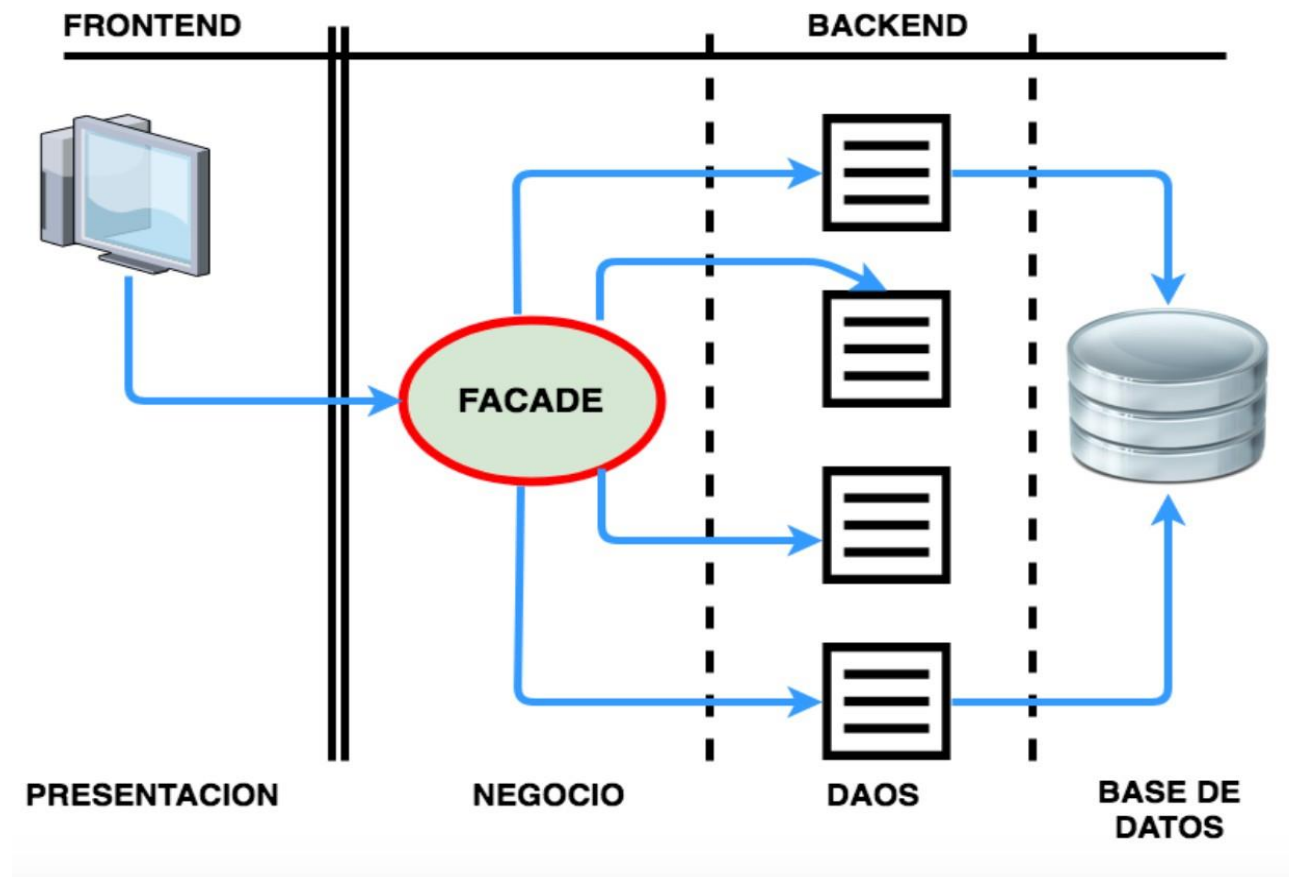
APLICACIONES WEB DINÁMICAS CON JAVA

Arquitectura en capas

- **Capa de acceso a datos:** Esta capa es en donde se representan las entidades de negocios en objetos perfectamente reproducibles en lenguaje java. En capítulos posteriores se profundizará en esta capa, la cual es conocida como DAO (Data Access Object).
- **Capa de aplicación:** Corresponde la modelo, pero en términos prácticos en esta capa se programa la lógica de negocio de la aplicación.
- **Capa web:** Esta es la capa de presentación, la cual contiene todos los componentes que presentan los datos al usuario.

APLICACIONES WEB DINÁMICAS CON JAVA

Arquitectura en capas



APLICACIONES WEB DINÁMICAS CON JAVA

Cada una de las capas descritas anteriormente se definen de acuerdo a su responsabilidad.

- **Capa de datos:** Comenzaremos describiendo la última capa, la encargada de los datos. En esta capa se proveen los mecanismos de persistencia que permiten que se almacenen, se consulten modifiquen y eliminen los datos. Esta capa se implementa junto a algún motor de base de datos como oracle, mysql, sqlserver, etc.
- **La capa de acceso a datos DAO:** Este sector de la arquitectura interactúa directamente con la capa de datos y se encarga de acceder a ella para poder crear, eliminar, modificar y buscar información. Esta capa si bien se comunica con la base de datos, también cuenta con otro tipo de entidad de nombre DTO (Data Transfer Object).
- **DTO:** Si bien esta clase no aparece en el diagrama ya que trabaja codo a codo con el dao, es necesario dar su descripción. Esta clase está encargada de proveer los objetos que son un fiel reflejo de las entidades de base de datos (tablas) que residen en la base de datos.

APLICACIONES WEB DINÁMICAS CON JAVA

- En términos simples imaginemos que tenemos la tabla usuarios que tienen los atributos nombre, edad y apellidos. Para que el sistema pueda interactuar con la base de datos y con esta tabla, el DTO debe proveer una clase que tenga exactamente la misma estructura que la tabla usuarios. Esto se traduce en una clase java de nombre usuarios con los atributos de nombre edad y apellidos.
- **La capa de negocios:** Esta capa como se describió anteriormente es la encargada de procesar los datos y de otorgar la lógica de negocios al sistema. En el diagrama se ve el nombre facade, el cual es la denominación de un patrón de diseño que al igual que el conserje de un edificio, conoce a todos los residentes y sabe dónde viven de acuerdo al número de departamento. La capa de negocios es parte importante de una arquitectura en capas.

APLICACIONES WEB DINÁMICAS CON JAVA

- Pensemos que nuestra aplicación provee la información de las ventas del mes y en la capa de daos existen multitud de clases. El facade es quien sabe dónde encontrar por ejemplo las ventas con boletas y se comunica con ella sin problemas. Este patrón facade provee un acceso ordenado a las clases DAOS o también a mas clases de negocios. Esta capa es el controlador si lo extrapolamos a la arquitectura MVC.
- **La capa web:** Aquí se generan las páginas jsp que se mostraran al usuario final después de que el requiest pase al controlador y su clase facade, a la capa de datos dao, consulte en la base de datos para luego retornar la salida.

APLICACIONES WEB DINÁMICAS CON JAVA

Aplicaciones en capas

Competencias

- Análisis de la solución
- Implementación del sistema Plain Old Java Object POJOS
- Api JDBC

Introducción

- Como programadores java EE, programadores java, o como le quieran llamar al puesto necesitamos conocer todas estas capas y su implementación, ya sea para la creación de nuevos proyectos o dar mantenimientos a los ya existentes, por lo cual a continuación se empezará a desarrollar un pequeño sistema de gestión de cursos.

APLICACIONES WEB DINÁMICAS CON JAVA

Análisis de la solución

En ingeniería de software, al momento de plasmar los requerimientos de un sistema se utiliza un artefacto llamado casos de uso.

Los siguientes pasos corresponden al caso de uso inscripción a un curso, que servirá de ejemplo para la futura construcción.

- El usuario solicita una página de inscripción
- El sistema la arma y se la entrega al usuario
- El usuario selecciona un curso y sus datos para que, al momento de confirmar, se genere una transacción a la base de datos.

APLICACIONES WEB DINÁMICAS CON JAVA

Pensando un poco:

- Queremos inscribirnos a un curso: **curso** es una entidad.
- Si queremos inscribirnos al curso, debemos dar nuestros datos personales y además cómo pagar la inscripción: tenemos la entidad **inscripción**.
- Actualmente existen varios métodos de pago, por lo cual tenemos la última entidad que sería la **forma de pago**.

Estas tres entidades se convertirán en el modelo de datos, con tres tablas:

APLICACIONES WEB DINÁMICAS CON JAVA

Tablas de la BD:

Object Type		TABLE		Object		CURSO				
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment	
CURSO	ID_CURSO	VARCHAR2	50	-	-	1	-	-	-	
	DESCRIPCION	VARCHAR2	100	-	-	-	✓	-	-	
	PRECIO	NUMBER	22	-	-	-	✓	-	-	
									1 - 3	

Results	Explain	Describe	Saved SQL	History						
Object Type		TABLE		Object		FORMA_PAGO				
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment	
FORMA_PAGO	ID_FORMA_PAGO	VARCHAR2	100	-	-	1	-	-	-	
	DESCRIPCION	VARCHAR2	100	-	-	-	✓	-	-	
	RECARGA	VARCHAR2	100	-	-	-	✓	-	-	
									1 - 3	

Results	Explain	Describe	Saved SQL	History						
Object Type		TABLE		Object		INSCRIPCION				
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment	
INSCRIPCION	ID_INSCRIPCION	NUMBER	22	-	-	1	-	-	-	
	NOMBRE	VARCHAR2	100	-	-	-	✓	-	-	
	TELEFONO	NUMBER	22	-	-	-	✓	-	-	
	ID_CURSO	VARCHAR2	50	-	-	-	✓	-	-	
	ID_FORMA_PAGO	VARCHAR2	50	-	-	-	✓	-	-	
									1 - 5	

APLICACIONES WEB DINÁMICAS CON JAVA

Implementación del sistema

- Cuando se empieza a programar una solución empresarial, recomendamos empezar con las entidades de negocio, ya que son el corazón del sistema.
- En este caso ya analizamos el caso y obtuvimos 3 entidades, el curso, la forma de pago y la inscripción.
- Si bien utilizaremos lo aprendido hasta acá (páginas jsp, etiquetas html y servlets) debemos programar clases java que mapean las entidades del mundo real de una inscripción.
- A medida que avancemos en el proceso iremos conociendo nuevos integrantes de una aplicación java JEE, por lo que nos detendremos a explicar qué son y cuál es su función.

APLICACIONES WEB DINÁMICAS CON JAVA

Clase que representa
la forma de pago

```
1 package com.desafiolatam.entidades;
2
3 public class FormaDePagoDTO {
4     private int idFormaDePago;
5     private String descripcion;
6     private double recargo;
7
8     public int getIdFormaDePago() {
9         return idFormaDePago;
10    }
11
12    public void setIdFormaDePago(int idFormaDePago) {
13        this.idFormaDePago = idFormaDePago;
14    }
15
16    public String getDescripcion() {
17        return descripcion;
18    }
19
20    public void setDescripcion(String descripcion) {
21        this.descripcion = descripcion;
22    }
23
24    public double getRecargo() {
25        return recargo;
26    }
27
28    public void setRecargo(double recargo) {
29        this.recargo = recargo;
30    }
31 }
32
33
34
35
```

APLICACIONES WEB DINÁMICAS CON JAVA

Clase que representa
un curso

```
1 package com.desafiolatam.entidades;
2
3 public class CursoDTO {
4
5     private int idCurso;
6
7     private String descripcion;
8
9     private double precio;
10
11     public int getIdCurso() {
12         return idCurso;
13     }
14     public void setIdCurso(int idCurso) {
15         this.idCurso = idCurso;
16     }
17     public String getDescripcion() {
18         return descripcion;
19     }
20     public void setDescripcion(String descripcion) {
21         this.descripcion = descripcion;
22     }
23     public double getPrecio() {
24         return precio;
25     }
26     public void setPrecio(double precio) {
27         this.precio = precio;
28     }
29 }
30
```

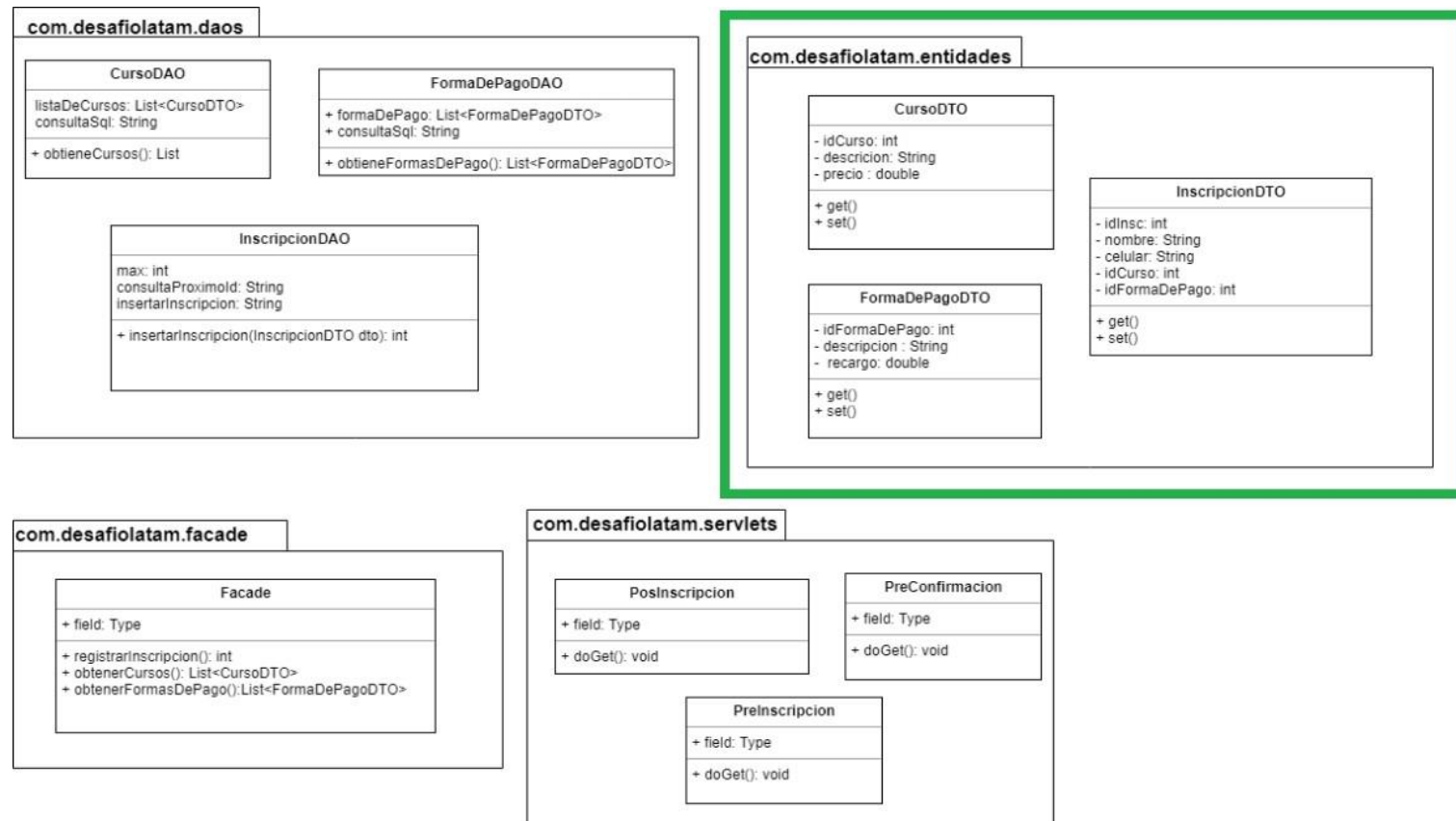
APLICACIONES WEB DINÁMICAS CON JAVA

Clase que representa
una inscripción

```
5     private int idInsc;  
6  
7     private String nombre;  
8  
9     private String celular;  
10  
11    private int idCurso;  
12  
13    private int idFormaDePago;  
14  
15    public int getIdInsc() {  
16        return idInsc;  
17    }  
18    public void setIdInsc(int idInsc) {  
19        this.idInsc = idInsc;  
20    }  
21    public String getNombre() {  
22        return nombre;  
23    }  
24    public void setNombre(String nombre) {  
25        this.nombre = nombre;  
26    }  
27    public String getCelular() {  
28        return celular;  
29    }  
30    public void setCelular(String celular) {  
31        this.celular = celular;  
32    }  
33    public int getIdCurso() {  
34        return idCurso;  
35    }  
36    public void setIdCurso(int idCurso) {  
37        this.idCurso = idCurso;  
38    }  
39    public int getIdFormaDePago() {  
40        return idFormaDePago;  
41    }  
42    public void setIdFormaDePago(int idFormaDePago) {  
43        this.idFormaDePago = idFormaDePago;  
44    }  
45 }
```


APLICACIONES WEB DINÁMICAS CON JAVA

En el diagrama de clases se está cubriendo el package de entidades
Diagrama de clases: Entidades



APLICACIONES WEB DINÁMICAS CON JAVA

POJOS

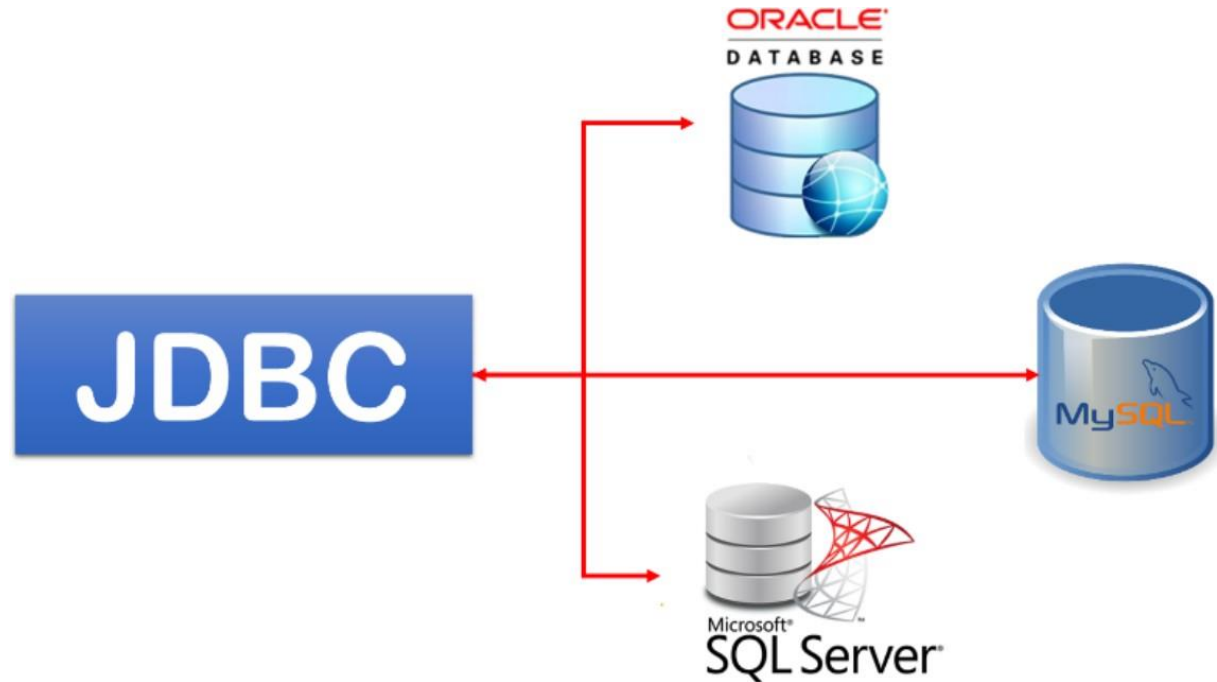
- Básicamente los POJOS (Plain Old Java Object) es un concepto creado al momento de la aparición de frameworks como Spring que describen a las clases java puras con una estructura estándar en donde se declaran los atributos, los setters y getters correspondientes. Si bien tiene esta estructura no tienen lógica de negocio en ellos y solo se usan para la comunicación y el modelamiento del sistema.

Api JDBC

- Sigla de **Java Database Connectivity**, es la interface de programación estándar que permite que los sistemas java puedan comunicarse y gestionar las bases de datos. La api jdbc consta de un conjunto de interfaces y clases escritas en java que permiten la manipulación y administración de los datos de un sistema de base de datos.

APLICACIONES WEB DINÁMICAS CON JAVA

Con jdbc es posible utilizar cualquier motor de BD



APLICACIONES WEB DINÁMICAS CON JAVA

Capa de acceso a datos y objetos DAO

Competencias

- Construcción Curso DAO
- Construcción Forma Pago DAO
- Construcción Inscripción DAO
- Construcción y uso de Facade

APLICACIONES WEB DINÁMICAS CON JAVA

Clase CursoDao: Se comunica con la BD

```
1 package com.desafiolatam.daos;
2
3 import java.sql.Connection;
4
12 public class CursoDao {
13
14     public List obtenerCursos() throws SQLException, ClassNotFoundException {
15
16         //creamos la lista de objetos que devolverán los resultados
17         List<CursoDTO> listaDeCursos = new ArrayList<CursoDTO>();
18
19         //creamos la consulta a la base de datos
20         String consultaSql = " SELECT id_curso, descripcion, precio "
21                               + " FROM DESAFIO.curso ";
22
23         //conexion a la base de datos y ejecucion de la sentencia
24         Class.forName("oracle.jdbc.driver.OracleDriver");
25         Connection conexion = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","SYSTEM","chotokan");
26
27         try(PreparedStatement stmt = conexion.prepareStatement(consultaSql)){
28
29             ResultSet resultado = stmt.executeQuery();
30             while(resultado.next()) {
31                 CursoDTO cursoDto = new CursoDTO();
32                 cursoDto.setIdCurso(resultado.getInt("id_curso"));
33                 cursoDto.setDescripcion(resultado.getString("descripcion"));
34                 cursoDto.setPrecio(resultado.getDouble("precio"));
35                 listaDeCursos.add(cursoDto);
36             }
37         }
38         catch(Exception ex) {
39             ex.printStackTrace();
40         }
41         return listaDeCursos;
42     }
43 }
44 }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Clase FormaDePagoDao: Se comunica con la BD

```
1 package com.desafiolatam.daos;
2
3 import java.sql.Connection;
4
13 public class FormaDePagoDAO {
14
15     public List obtieneFormasDePago() throws SQLException, ClassNotFoundException {
16
17         //creamos la lista de objetos que devolveran los resultados
18         List<FormaDePagoDTO> listaDeCursos = new ArrayList<FormaDePagoDTO>();
19
20         //creamos la consulta a la base de datos
21         String consultaSql = " SELECT id_forma_pago, descripcion, recarga "
22             + " FROM DESAFIO.forma_pago ";
23
24         //conexion a la base de datos y ejecucion de la sentencia
25         Class.forName("oracle.jdbc.driver.OracleDriver");
26         Connection conexion = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "SYSTEM", "chotokan");
27
28         try(PreparedStatement stmt = conexion.prepareStatement(consultaSql)){
29
30             ResultSet resultado = stmt.executeQuery();
31             while(resultado.next()) {
32                 FormaDePagoDTO formaPago = new FormaDePagoDTO();
33                 formaPago.setIdFormaDePago(resultado.getInt("id_forma_pago"));
34                 formaPago.setDescripcion(resultado.getString("descripcion"));
35                 formaPago.setRecargo(resultado.getDouble("recarga"));
36                 listaDeCursos.add(formaPago);
37             }
38
39         } catch (Exception ex) {
40             ex.printStackTrace();
41         }
42         return listaDeCursos;
43     }
44 }
45 }
```

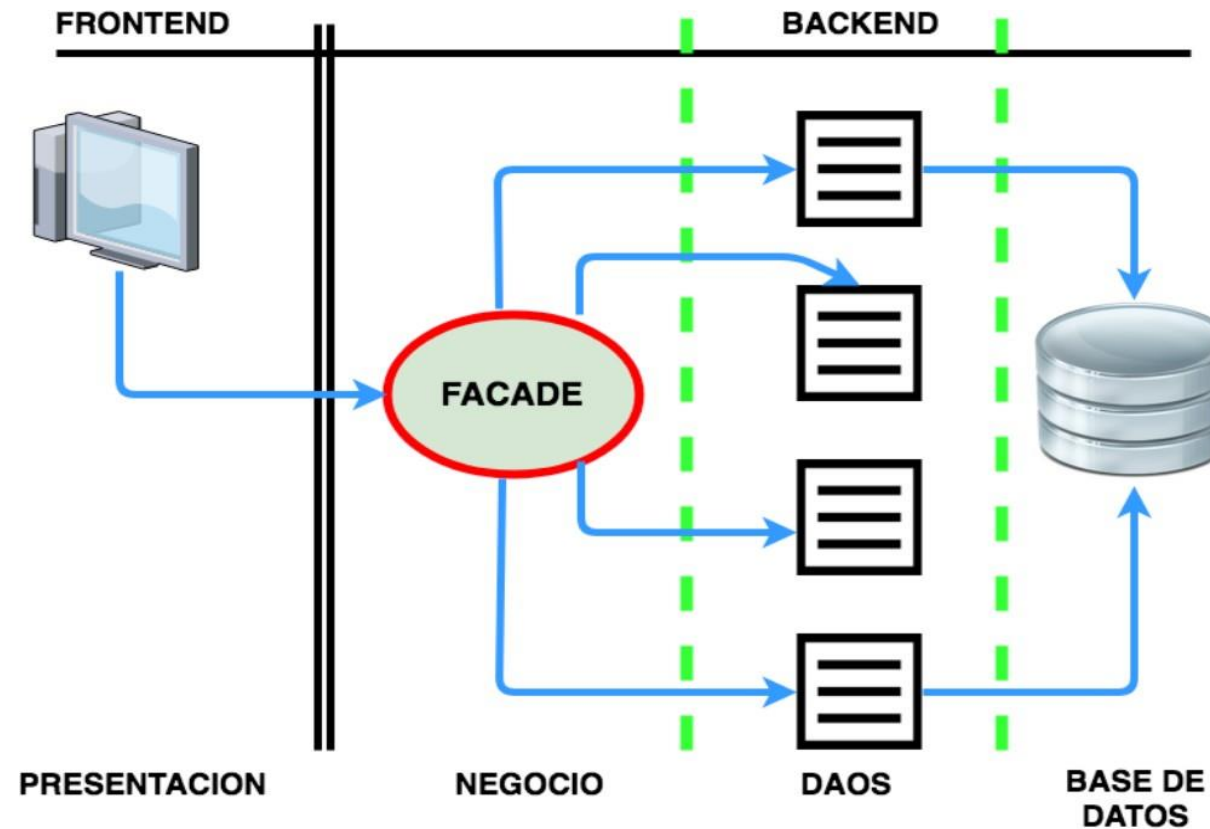
APLICACIONES WEB DINÁMICAS CON JAVA

Clase IncripcionDao: Se comunica con la BD

```
1 package com.desafiolatam.daos;
2
3 import java.sql.Connection;
4
15 public class IncripcionDAO {
16     public int insertarInscripcion(IncripcionDTO dto) throws SQLException, ClassNotFoundException {
17         int max = 0;
18         //Query para obtener una secuencia y asignar un id. La funcion MAX solo obtiene el valor de id_inscripcion
19         //y le suma 1, con eso hacemos el incremento
20         String consultaProximoId = " SELECT MAX(id_inscripcion)+1 FROM DESAFIO.inscripcion ";
21         //Query que insertara el valor
22         String insertarInscripcion = " INSERT INTO DESAFIO.inscripcion("
23             + " id_inscripcion, nombre, telefono, id_curso, id_forma_pago )"
24             + " VALUES (?, ?, ?, ?, ?) ";
25         //conexion a la base de datos y ejecucion de la sentencia
26         Class.forName("oracle.jdbc.driver.OracleDriver");
27         Connection conexion = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","SYSTEM","chotokan");
28
29         try(
30             PreparedStatement stmt1 = conexion.prepareStatement(consultaProximoId);
31             PreparedStatement stmt2 = conexion.prepareStatement(insertarInscripcion);
32         ){
33             ResultSet resultado = stmt1.executeQuery();
34             if(resultado.next()) {
35                 max = resultado.getInt(1);
36                 stmt2.setInt(1, max);
37                 stmt2.setString(2, dto.getNombre());
38                 stmt2.setString(3, dto.getCelular());
39                 stmt2.setInt(4, dto.getIdCurso());
40                 stmt2.setInt(5, dto.getIdFormaDePago());
41
42                 if(stmt2.executeUpdate() != 1) {
43                     throw new RuntimeException("A ocurrido un error inesperado");
44                 }
45             }
46         } catch (Exception ex) {
47             ex.printStackTrace();
48             throw new RuntimeException("A ocurrido un error inesperado" + ex);
49         }
50         return max;
51     }
52 }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Capa cubierta con clases DAOs y DTO



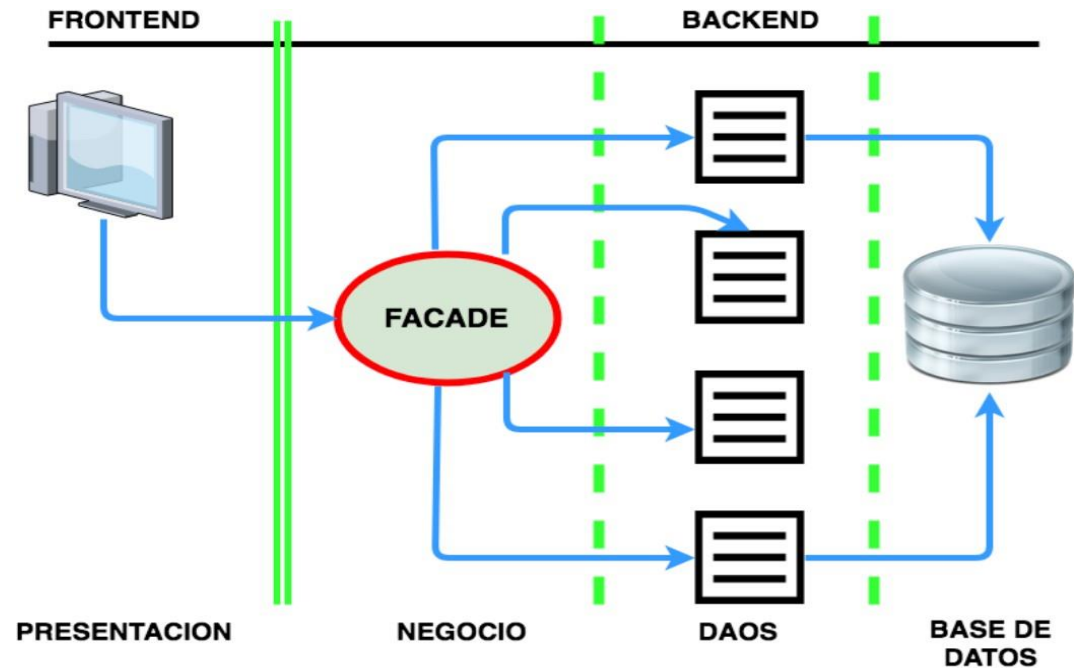
APLICACIONES WEB DINÁMICAS CON JAVA

Clase Facade

```
1 package com.desafiolatam.facade;
2
3 import java.sql.SQLException;
4 import java.util.List;
5
6 import com.desafiolatam.daosCursoDao;
7 import com.desafiolatam.daos.FormaDePagoDAO;
8 import com.desafiolatam.daos.InscripcionDAO;
9 import com.desafiolatam.entidadesCursoDTO;
10 import com.desafiolatam.entidades.FormaDePagoDTO;
11 import com.desafiolatam.entidades.InscripcionDTO;
12
13 public class Facade {
14
15     public int registrarInscripcion(InscripcionDTO dto) throws SQLException {
16         InscripcionDAO dao = new InscripcionDAO();
17         return dao.insertarInscripcion(dto);
18     }
19
20     public List<CursoDTO> obtenerCursos() throws SQLException{
21         CursoDao dao = new CursoDao();
22         return dao.obtieneCursos();
23     }
24
25     public List<FormaDePagoDTO> obtenerFormasDePago() throws SQLException{
26         FormaDePagoDAO dao = new FormaDePagoDAO();
27         return dao.obtieneFormasDePago();
28     }
29 }
```

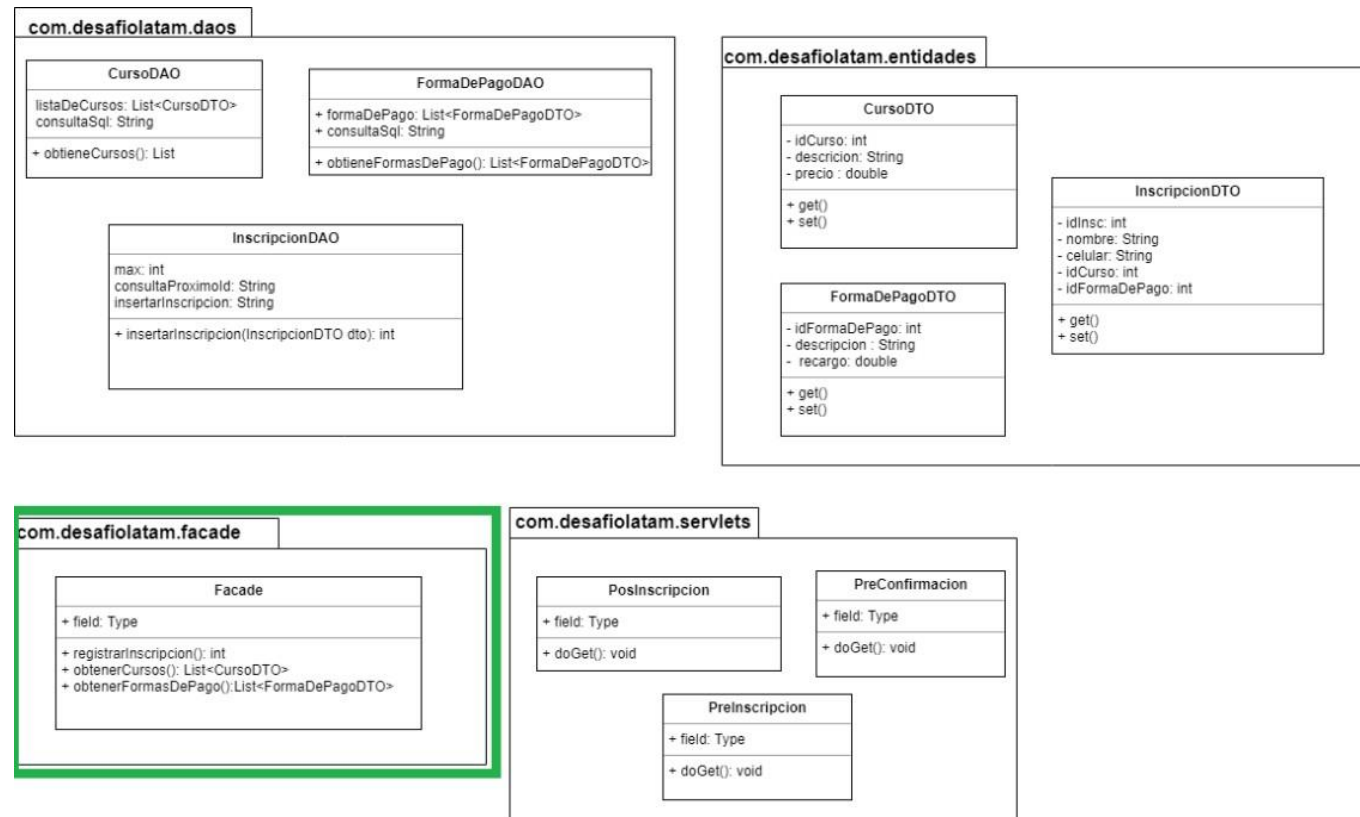

APLICACIONES WEB DINÁMICAS CON JAVA

Gracias al “facade” podemos comunicarnos con el backend



APLICACIONES WEB DINÁMICAS CON JAVA

Diagrama de clases Facade



JAVA EMPRESARIAL Y LAS BASES DE DATOS

[

]

APLICACIONES WEB DINÁMICAS CON JAVA

Competencias

- Conocer los conceptos básicos sobre base de datos Instalar el motor Oracle 18g
- Crear base de datos en Oracle Conocer
- IDE de BD Dbeaver
- Conectar aplicaciones Java con base de datos

APLICACIONES WEB DINÁMICAS CON JAVA

Introducción

- Toda aplicación debe interactuar con una base de datos, y es obligatorio para un desarrollador conocer las distintas técnicas utilizadas para llevar a cabo este cometido.
- Entrar en el mundo de la capa de persistencia en un sistema java involucra no solo conocer sobre sentencias SQL o conocer las capas de forma teórica, también involucra familiarizarse con los entornos de trabajo con bases de datos, con las herramientas que permiten administrar las tablas, con las formas de comunicación entre aplicación y motores de bases de datos, con el concepto de driver de conexión y en general con todo el ecosistema que si o si uno se encuentra en un equipo de desarrollo.

APLICACIONES WEB DINÁMICAS CON JAVA

Conceptos básicos sobre bases de datos

- Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
- Podemos decir que una base de datos es comparable con una gran biblioteca con documentos y archivos, ordenados mediante estantes con una serie de índices que permiten al bibliotecario encontrar un archivo de forma rápida, sin tener que revisar cada uno de los elementos almacenados. En el área informática, una base de datos almacena y gestiona la información organizada en tablas.

Instalación base de datos Oracle 18g express Edition

- Para seguir los ejercicios, en esta parte del curso utilizaremos una base de datos robusta y quizás un poco intimidante como es Oracle en su versión 18g bajo Windows.

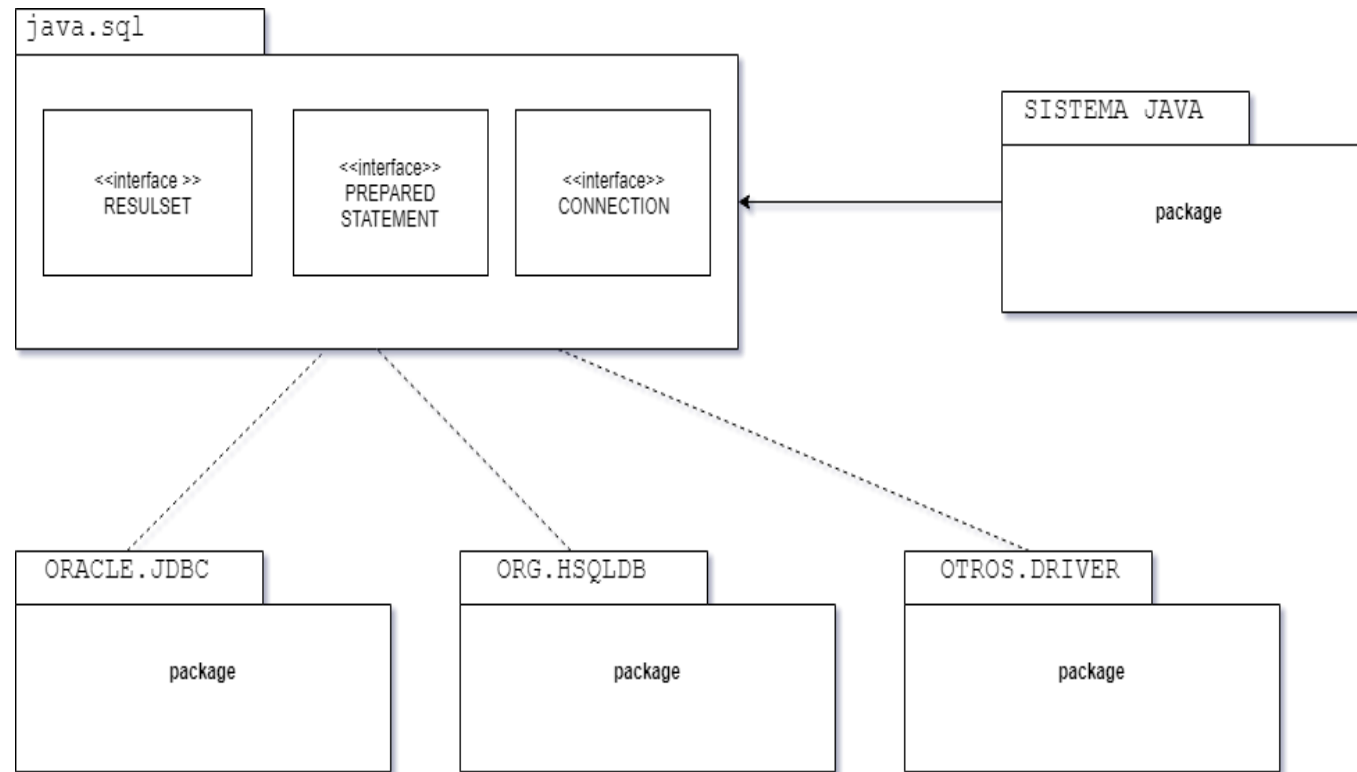
APLICACIONES WEB DINÁMICAS CON JAVA

Conectar programas java con base de datos con JDBC

- La API encargada de conectar una aplicación java con un motor de base de datos tiene por nombre JDBC, de **data base java connectivity** y se encuentra alojado en el paquete `java.sql`. Dentro de este paquete, encontraremos clases e interfaces a través de las cuales podremos ejecutar queries y updates, invocar procedimientos y funciones y en definitiva, manejar y administrar una base de datos desde el programa java que se esté realizando.
- Todo la api JDBC está diseñada sobre interfaces que definen cómo se ejecuta la conexión, la forma de ejecutar las sentencias sql, entre otras funcionalidades. Para poder trabajar junto a una base de datos utilizando esta api, es necesario tener una serie de clases que implementen todas estas interfaces. A este conjunto de clases se le llama drivers.

APLICACIONES WEB DINÁMICAS CON JAVA

Diagrama JDBC



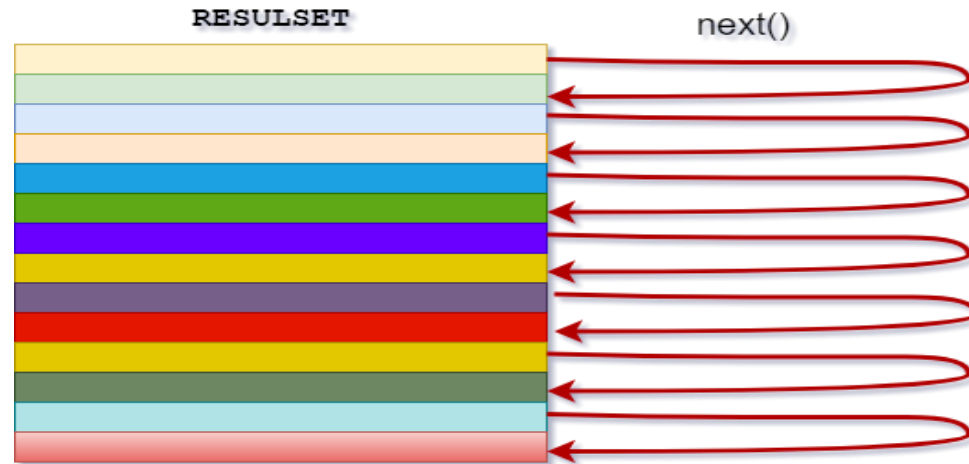
APLICACIONES WEB DINÁMICAS CON JAVA

Componentes del api JDBC

- **ResultSet:** La interface pública `resultset` proporciona los métodos necesarios para poder obtener una representación de los datos de una tabla, previa consulta `sql`. El `resultset` utiliza un cursor el cual se
- posiciona después de la primera fila. Utilizando el método `next` se mueve el cursor a la siguiente fila
- y cuando no encuentra más filas devuelve un parámetro booleano falso. Para trabajar con el `resultset` se utiliza un ciclo `while` en donde se pregunta si el `resultset` tiene más elementos y mientras la condición se cumpla podemos trabajar con la fila obtenida

APLICACIONES WEB DINÁMICAS CON JAVA

ResultSet



APLICACIONES WEB DINÁMICAS CON JAVA

- **PreparedStatement:** Interface que representa una sentencia sql precompilada. La sentencia sql es almacenada en el preparedStatement y gracias a ella es posible ejecutarla para trabajar con sentencias select, update, delete, etc.
- **Connection:** Una conexión (sesión) con una base de datos específica. Las instrucciones SQL se ejecutan y los resultados se devuelven dentro del contexto de una conexión. La base de datos de un objeto puede proporcionar información que describe sus tablas, su gramática SQL admitida, sus procedimientos almacenados, las capacidades de esta conexión, etc. Esta información se obtiene con el método `getMetaData`.

APLICACIONES WEB DINÁMICAS CON JAVA

Drivers de conexion con Oracle

- En las siguientes líneas se describen los tipos de drivers de conexion con Oracle
- **Oracle JDBC Thin Driver**
- **Oracle JDBC OCI Driver**
 - Levantar el driver y establecer la conexión.
 - Ejecutar la query, recorrerlo y mostrar los datos por pantalla
 - Cerrar la conexión.

APLICACIONES WEB DINÁMICAS CON JAVA

Manipular información de una base de datos

Competencias

- Crear el modelo de datos
- Vincular el driver ojdbc8.jar al proyecto
- Programar la clase utilizando el api JDBC
- Ejecutar un insert, delete, delete en cascada y update.

APLICACIONES WEB DINÁMICAS CON JAVA

Creación del modelo de datos

- El siguiente script es utilizado para crear las dos tablas del ejemplo. El negocio indica que quiere registrar los departamentos de la empresa las cuales están repartidas en el mundo y a la vez mantener a los empleados y su departamento. Para eso creamos dos tablas: DEPARTAMENTO y EMPLEADO.

```
*<Oracle - XE> Script
--creacion de tablas para el ejemplo 01 jdbc desafio latam

CREATE TABLE DEPARTAMENTO (NUMDEPTO NUMBER PRIMARY KEY NOT NULL, NOMDEPTO VARCHAR(100), UBICACIONDPTO VARCHAR(100));

CREATE TABLE EMPLEADO (NUMEMPLEADO NUMBER PRIMARY KEY, NOMBRE VARCHAR(100), NUMDEPTO INTEGER);

ALTER TABLE EMPLEADO
ADD CONSTRAINT DEP_EMPLEADO
FOREIGN KEY (NUMDEPTO)
REFERENCES DEPARTAMENTO (NUMDEPTO);

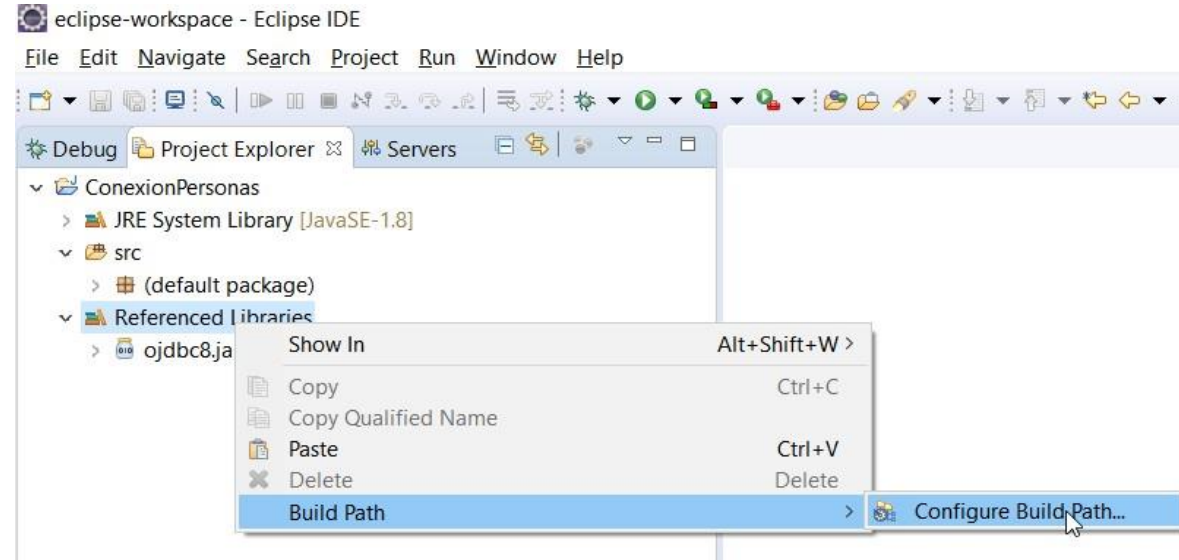
INSERT INTO DEPARTAMENTO VALUES(11,'INFORMATICA','CHILE');

INSERT INTO EMPLEADO VALUES (0101,'Matias Zarate', 11);
```

APLICACIONES WEB DINÁMICAS CON JAVA

Importar Driver JDBC a Eclipse: Configure Build Path

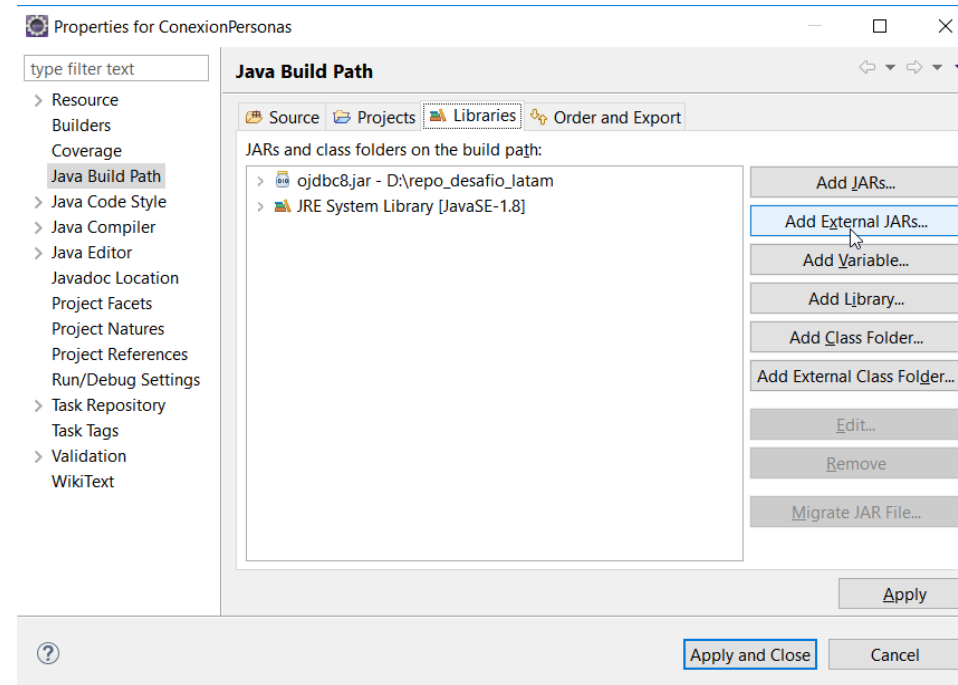
- Abrimos el Eclipse y generamos un nuevo proyecto java normal. Para añadir el driver simplemente seleccionamos con botón derecho sobre la opción Referenced Libraries y pinchamos la opción Build Path, Configure Build Path.



APLICACIONES WEB DINÁMICAS CON JAVA

Agregando .jar

- En la ventana Properties buscar la opción Java Build Path, y en la pestaña libraries pinchar el botón Add External Jar para seleccionar el ojdbc8.jar.



APLICACIONES WEB DINÁMICAS CON JAVA

Creación de clase de manipulación de datos

-
-

```
DemoPersonas.java
1 import java.sql.DriverManager;
2
3 public class DemoPersonas {
4     public static void main(String...args) {
5         String usr = "sys as sysdba";
6         String pwd = "admin";
7         String driver = "oracle.jdbc.driver.OracleDriver";
8         String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
9         Connection conn = null;
10        PreparedStatement pstmt = null;
11        ResultSet rs = null;
12        try {
13            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
14            // 1 PARTE LEVANTAR EL DRIVER
15            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
16            // levantamos el driver
17            Class.forName(driver);
18            //establecemos la conexion
19            conn = DriverManager.getConnection(url,usr,pwd);
20            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
21            // 2 PARTE GENERACION DE QUERY
22            //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23            //definimos la query
24            String sql = "SELECT * FROM departamento, empleado";
25            //preparamos sentencias que ejecutaremos
26            pstmt = conn.prepareStatement(sql);
27            //ejecutamos sentencia y obtenemos los resultados
28            rs = pstmt.executeQuery();
29            //recorremos los resultados
30            while(rs.next()) {
31                System.out.println("DEPARTAMENTOS::");
32                System.out.println(rs.getInt("NUMDEPTO"));
33                System.out.println(rs.getString("NOMDEPTO"));
34                System.out.println(rs.getString("UBICACIONDPTO"));
35                System.out.println("EMPLEADOS::");
36                System.out.println(rs.getInt("NUMEMPLEADO"));
37                System.out.println(rs.getString("NOMBRE"));
38                System.out.println(rs.getInt("NUMDEPTO"));
39            }
40        } catch (Exception ex) {
41            ex.printStackTrace();
42        }
43    }
44 }
45
46 }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Los import cuenta con:

```
1 import java.sql.DriverManager;  
2 import java.sql.Connection;  
3 import java.sql.PreparedStatement;  
4 import java.sql.ResultSet;
```

- . Son las clases propias de JDBC que nos permite conectar a las bases de datos. Los nombres son autoexplicativos
- **DriverManager**: Se encarga de administrar la carga del driver de conexión a base de datos. **Connection** : Disponibiliza las clases utilizadas para generar y administrar la conexión a base de datos.
- **PreparedStatement**: Ofrece clases para generar las sentencias con sus valores a utilizar.
- **ResultSet**: Elemento que nos otorga los datos rescatados desde Oracle. Mediante resulset podemos acceder a ellos y manipularlos.

APLICACIONES WEB DINÁMICAS CON JAVA

Donde:

- **usr**: Corresponde al nombre de usuario del esquema de base de datos. En este caso utilizamos el usuario sys.
- **pwd**: Configuración de la password de oracle.
- **driver**: La ruta del driver que se está utilizando. Pueden buscar en internet si se necesita otro motor de base de datos.
- **url**: String de conexión necesario para poder acceder a los recursos. Utiliza la tecnología thin.

La siguiente es la declaración de las instancias Connection, Prepared Statement y ResultSet

```
11 Connection conn = null;  
12 PreparedStatement pstmt = null;  
13 ResultSet rs = null;
```

APLICACIONES WEB DINÁMICAS CON JAVA

Class.forName y DriverManager

```
14      try {  
15          //:.....  
16          // 1 PARTE LEVANTAR EL DRIVER  
17          //:.....  
18          // levantamos el driver  
19          Class.forName(driver);  
20          // establecemos la conexion  
21          conn = DriverManager.getConnection(url,usr,pwd);
```

APLICACIONES WEB DINÁMICAS CON JAVA

Generación de Query

```
22 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23 // 2 PARTE GENERACION DE QUERY
24 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25 //definimos la query
26 String sql = "";
27 sql+="INSERT INTO DEPARTAMENTO (NUMDEPTO,NOMDEPTO,UBICACIONDPTO)";
28 sql+="VALUES(?,?,?)";
29 //preparamos sentencias que ejecutaremos
30 pstmt = conn.prepareStatement(sql);
31 //adjuntamos los valores a los parametros
32 pstmt.setInt(1,12);
33 pstmt.setString(2, "CONTABILIDAD");
34 pstmt.setString(3, "MEXICO");
35 int resultado = pstmt.executeUpdate();
36 //si la variable resultado es mayor a 0 el insert fue correcto
37 if(resultado > 0) {
38     System.out.println("Fila correctamente insertada !");
39 }else {
40     System.out.println("Ocurrio un error insertando el departamento");
41 }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Resultado de la consola

```
Console Problems Debug Shell
<terminated> DemoPersonas [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (Jul 17, 2019, 6:42:03 PM)
DEPARTAMENTOS:::
11
INFORMATICA
CHILE
EMPLEADOS:::
101
Matias Zarate
11
```

finally:

```
44     }finally {
45         //cerramos todos los recursos en orden inverso al que fueron declarados
46         try {
47             if(rs != null) rs.close();
48             if(pstm!=null) pstm.close();
49             if(conn!=null) conn.close();
50         } catch (Exception e) {
51             e.printStackTrace();
52         }
53     }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Sentencias DML

Ejecutar un insert

- Son conocidas como sentencias de modificación de datos. Entre ellas están los “insert”, “delete”, “update”
- A continuación, veremos el procedimiento para generar un insert a la tabla de departamentos. Utilizaremos la misma clase generada anteriormente, así que para trabajar más rápido puedes copiar dicha clase y eliminar las sentencias de select.

APLICACIONES WEB DINÁMICAS CON JAVA

Generando consultas

```
22 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
23 // 2 PARTE GENERACION DE QUERY
24 //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
25 //definimos la query
26 String sql = "";
27 sql+="INSERT INTO DEPARTAMENTO (NUMDEPTO,NOMDEPTO,UBICACIONDPTO)";
28 sql+="VALUES(?,?,?)";
29 //preparamos sentencias que ejecutaremos
30 pstmt = conn.prepareStatement(sql);
31 //adjuntamos los valores a los parametros
32 pstmt.setInt(1,12);
33 pstmt.setString(2, "CONTABILIDAD");
34 pstmt.setString(3, "MEXICO");
35 int resultado = pstmt.executeUpdate();
36 //si la variable resultado es mayor a 0 el insert fue correcto
37 if(resultado > 0) {
38     System.out.println("Fila correctamente insertada !");
39 }else {
40     System.out.println("Ocurrio un error insertando el departamento");
41 }
```


APLICACIONES WEB DINÁMICAS CON JAVA

Ejecutar un delete

- Eliminaremos el mismo registro que se insertó en la clase anterior. Para ello se presenta el siguiente código:

```
25      //definimos la query
26      String sql = "";
27      sql+="DELETE FROM DEPARTAMENTO WHERE NUMDEPTO = ?";
28      //preparamos sentencias que ejecutaremos
29      pstmt = conn.prepareStatement(sql);
30      //adjuntamos los valores a los parametros
31      pstmt.setInt(1,12);
32      int resultado = pstmt.executeUpdate();
33      //si la variable resultado es mayor a 0 el insert fue correcto
34      if(resultado == 1) {
35          System.out.println("Fila correctamente eliminada !");
36      }else {
37          System.out.println("Ocurrio un error eliminando el departamento");
38      }
```

APLICACIONES WEB DINÁMICAS CON JAVA

Ejecutar un delete masivo en cascada

- Hay ocasiones en que se necesitan eliminar datos de tablas que tienen relación entre sí

```
ALTER TABLE EMPLEADO  
ADD CONSTRAINT DEP_EMPLEADOFOR  
KEY (NUMDEPTO)  
REFERENCES DEPARTAMENTO (NUMDEPTO) ON DELETE CASCADE;
```

```
DELETE FROM EMPLEADO A WHERE  
A.NUMDEPTO = 11;
```

APLICACIONES WEB DINÁMICAS CON JAVA

Realizando un update

```
27 //definimos la query
28 String sql = "";
29 sql+="UPDATE DEPARTAMENTO SET UBICACIONDPTO = ? ";
30 //preparamos sentencias que ejecutaremos
31 pstmt = conn.prepareStatement(sql);
32 //adjuntamos los valores a los parametros
33 pstmt.setString(1,"JAPON");
34 int resultado = pstmt.executeUpdate();
35 System.out.println(resultado + "filas actualizadas");
```

APLICACIONES WEB DINÁMICAS CON JAVA

Patrón Data Access Object (DAO)

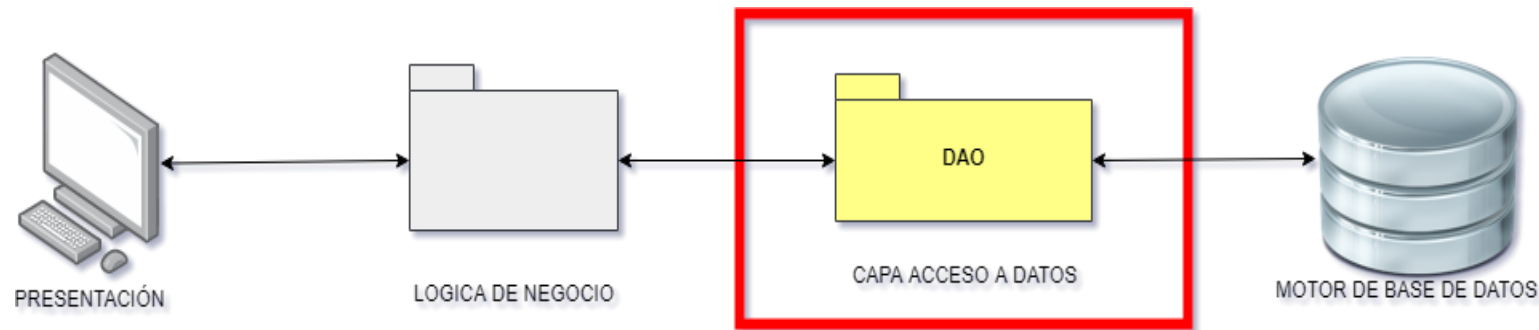
Competencias

- Conocer el patrón DAO
- Entender la estructura del patrón DAO
- Entender y aplicar el patrón Singleton para la conexión
- Generar aplicación básica DAO

APLICACIONES WEB DINÁMICAS CON JAVA

Conocer el patrón DAO

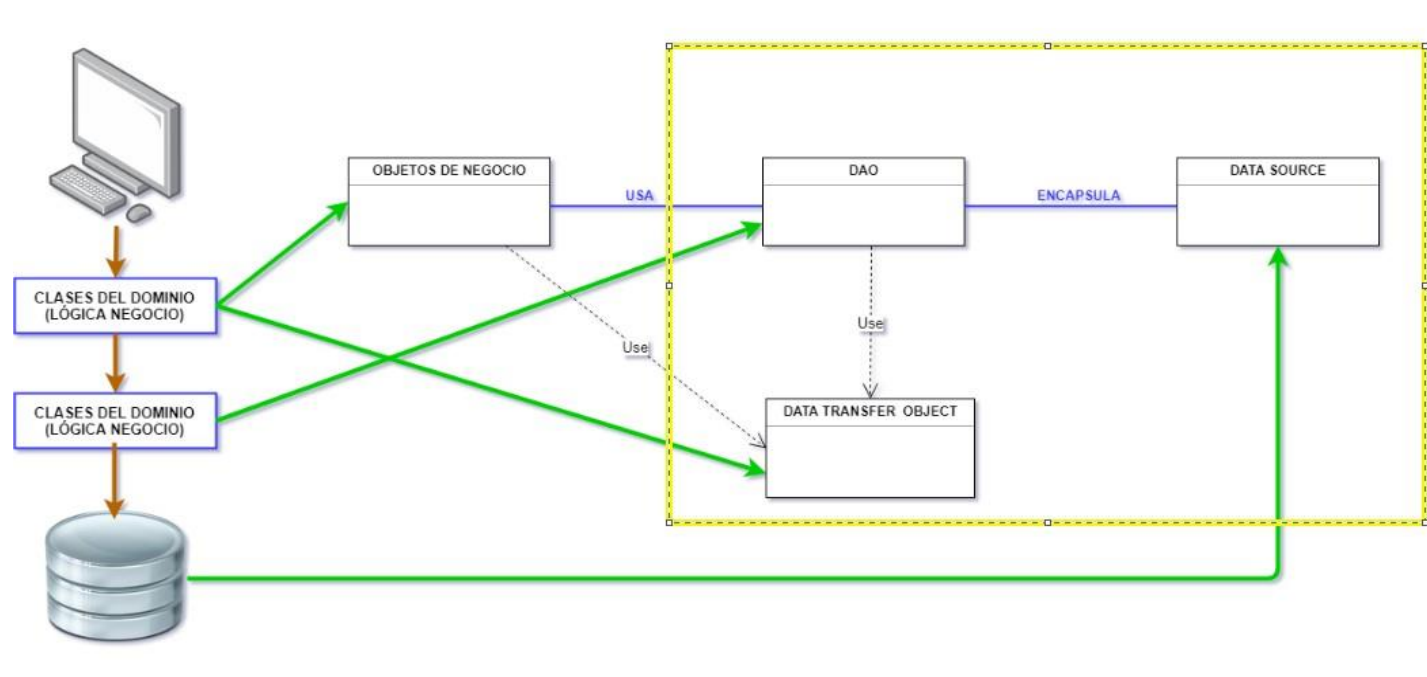
- El patrón DAO (Data Access Object) es un patrón de diseño catalogado como patrón estructural que se encarga de abstraer y encapsular el acceso a las bases de datos
- Capa DAO



APLICACIONES WEB DINÁMICAS CON JAVA

Estructura del patrón DAO

- Para entender la estructura del patrón analizaremos su diagrama de clases simplificado.



APLICACIONES WEB DINÁMICAS CON JAVA

Implementación del patrón DAO en una aplicación web

Ahora se explica cómo implementar el patrón DAO en una aplicación web utilizando:

- JSP para la vista
- Un servlet para procesar las peticiones
- Patrón DAO para el acceso a datos JDBC para establecer la conexión
- Base de datos de ejemplo

APLICACIONES WEB DINÁMICAS CON JAVA

Base de datos

- Utilizaremos la base de datos de ejemplo que se implementó en el previamente de JDBC de nombre desafio_ejemplo01. Tener a mano el DBEAVER o el SQL DEVELOPER ya que consultaremos constantemente los datos. Recordemos que esta base de datos cuenta solamente con dos tablas: Departamento y Empleados.
- Relación Departamento - Empleado



APLICACIONES WEB DINÁMICAS CON JAVA

Aplicación Web

Crear un nuevo proyecto en eclipse de tipo Dynamic Web Project. Para crear la estructura del proyecto, crear dentro de la carpeta src los siguientes packages, en donde dispondremos nuestras clases:

- `com.desafiolatam.dao`
- `com.desafiolatam.modelo`
- `com.desafiolatam.procesa`
- Conexion `com.desafiolatam.servlet`

El patrón DAO se implementará obviamente en el package dao. El resto es una arquitectura clásica compuesta por el modelo el cual contendrá las entidades de negocio, el procedimiento de conexión con JDBC y un servlet que se encargará de procesar las peticiones.

APLICACIONES WEB DINÁMICAS CON JAVA

Modelo: Empleado

```
1 package com.desafiolatam.modelo;
2
3 public class Empleado {
4     private int numEmpleado;
5     private String nombreEmpleado;
6     private int numDepto;
7
8     public Empleado(int numEmpleado, String nombreEmpleado, int numDepto) {
9         super();
10        this.numEmpleado = numEmpleado;
11        this.nombreEmpleado = nombreEmpleado;
12        this.numDepto = numDepto;
13    }
14
15    //getters y setters
```

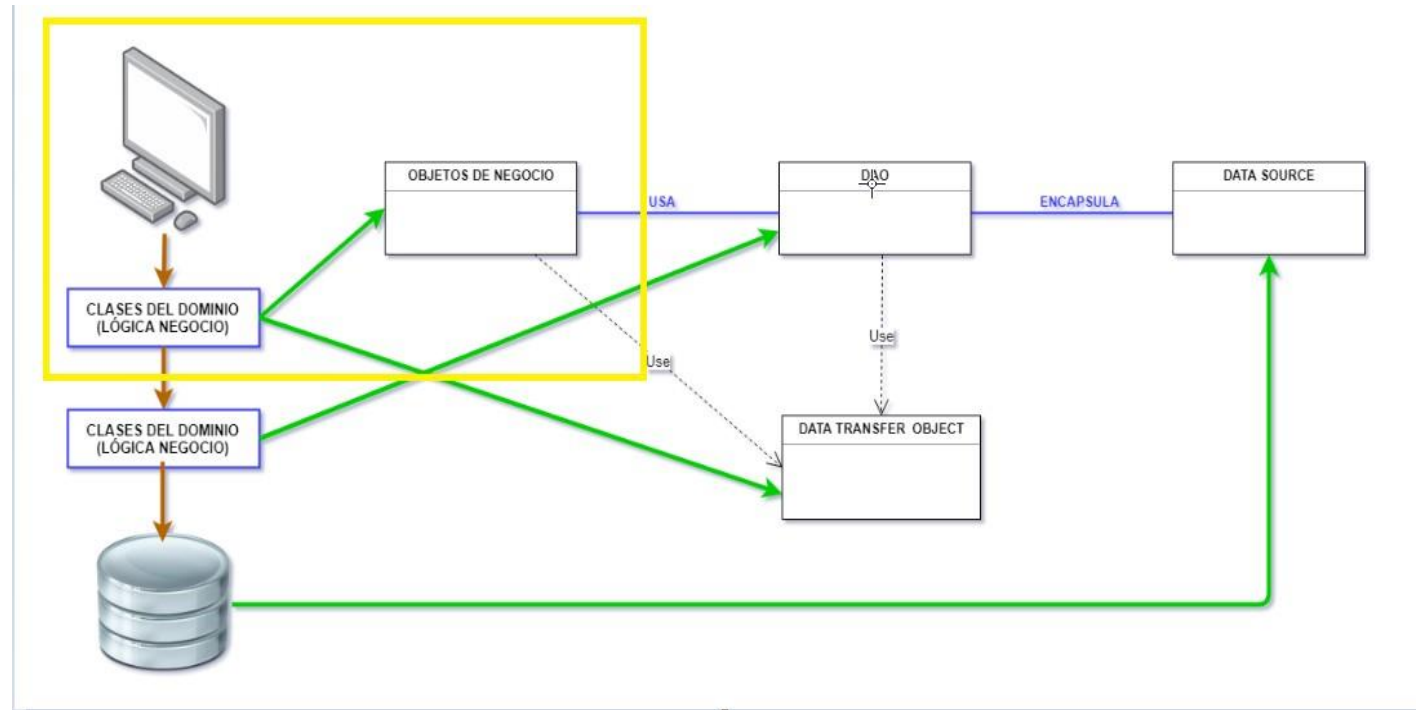
APLICACIONES WEB DINÁMICAS CON JAVA

Modelo: Departamento

```
1 package com.desafiolatam.modelo;
2
3 public class Departamento {
4     private int numDepto;
5     private String nombreDepto;
6     private String ubicacionDepto;
7
8     public Departamento(int numDepto, String nombreDepto, String ubicacionDepto) {
9         super();
10        this.numDepto = numDepto;
11        this.nombreDepto = nombreDepto;
12        this.ubicacionDepto = ubicacionDepto;
13    }
14
15    //getters y setters
```

APLICACIONES WEB DINÁMICAS CON JAVA

Diagrama DAO



APLICACIONES WEB DINÁMICAS CON JAVA

Creando Interfaz Departamento DAO

```
1 package com.desafiolatam.dao;
2
3 import java.util.List;
4
5
6
7 public interface DepartamentoDao {
8     public List<Departamento> obtieneDepartamento();
9 }
10
```

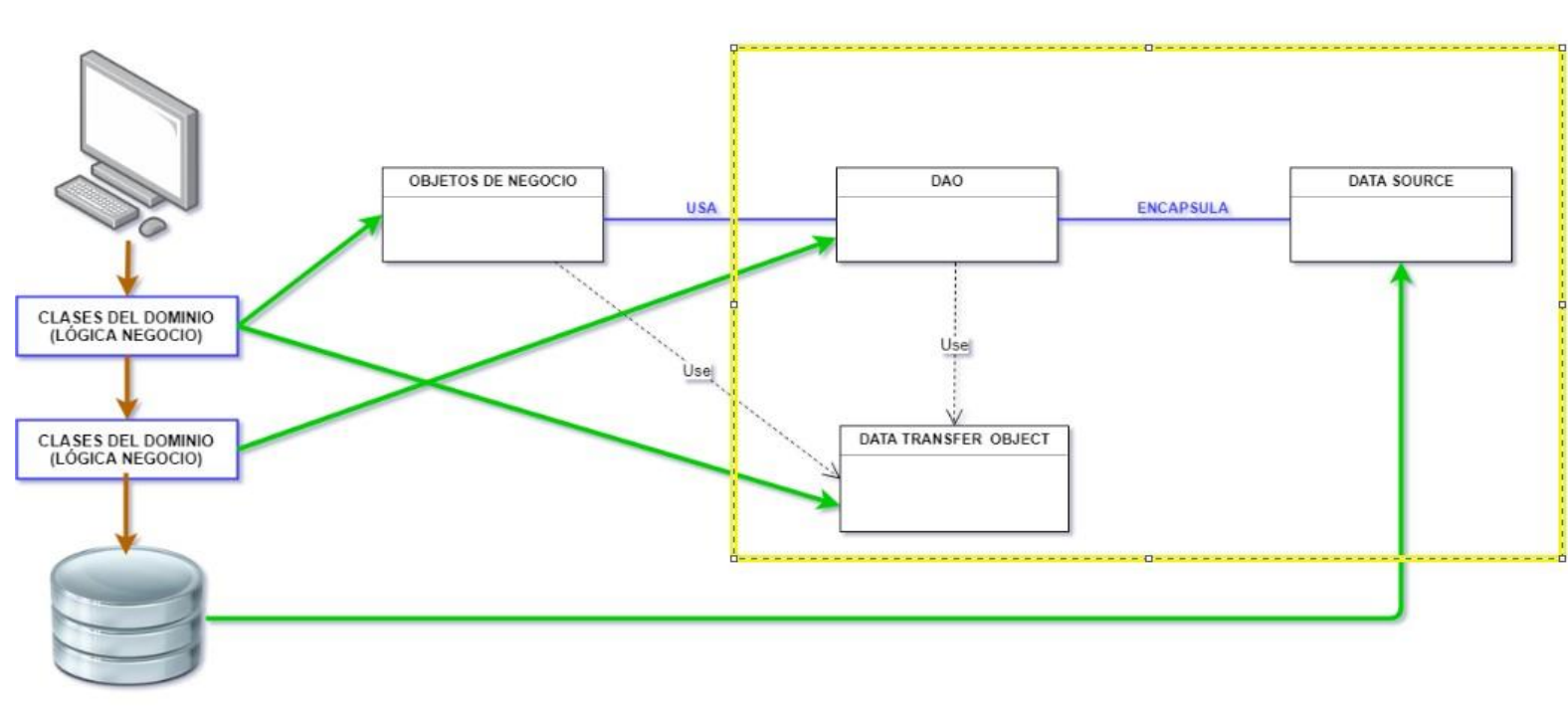
APLICACIONES WEB DINÁMICAS CON JAVA

Implementación Departamento DAO

```
1 package com.desafiolatam.dao;
2
3 import java.sql.Connection;
4
11 public class DepartamentoDaoImp extends AdministradorConexion implements DepartamentoDao{
12
13     public DepartamentoDaoImp() {
14         Connection conn = generaConexion();
15     }
16
17     @Override
18     public List<Departamento> obtieneDepartamento() {
19         String sql = "SELECT * FROM departamento";
20         List<Departamento> deptos = new ArrayList<Departamento>();
21         try {
22             pstm = conn.prepareStatement(sql);
23             rs = pstm.executeQuery();
24             while(rs.next()) {
25                 Departamento depto = new Departamento(rs.getInt("NUMDEPTO"),rs.getString("NOMDEPTO"),rs.getString("UBICACIONDPTO"));
26                 deptos.add(depto);
27             }
28         } catch (SQLException e) {
29             // TODO Auto-generated catch block
30             e.printStackTrace();
31         }
32         return deptos;
33     }
34 }
35
36
37
38
```

APLICACIONES WEB DINÁMICAS CON JAVA

Diagrama incorporando DATASOURCE.



APLICACIONES WEB DINÁMICAS CON JAVA

Clase AdministradorConexion (o Conexion)

```
1 package com.desafiolatam.procesaConexion;
2
3 import java.sql.Connection;
4
5
6
7 public class AdministradorConexion {
8
9
10     protected Connection conn = null;
11     protected PreparedStatement pstmt = null;
12     protected ResultSet rs = null;
13
14     protected Connection generaConexion() {
15         String usr = "sys as sysdba";
16         String pwd = "admin";
17         String driver = "oracle.jdbc.driver.OracleDriver";
18         String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";
19
20         try {
21             Class.forName(driver);
22             conn = DriverManager.getConnection(url,usr,pwd);
23         } catch (Exception ex) {
24             ex.printStackTrace();
25         }
26         return conn;
27     }
28
29 }
30
```


APLICACIONES WEB DINÁMICAS CON JAVA

El patrón Singleton

- ¿Qué pasaría si genero muchos objetos de la clase AdministradorConexion? pues generamos muchas conexiones simultáneas a la base de datos y a raíz de eso colgaremos el programa. Lo ideal sería que la conexión se genere **una** sola vez y nada más, independiente de cuantas instancias de la clase AdministradorConexion genere.
- Este análisis que hicimos, alguien hace muchos años atrás también lo hizo, y para nuestra fortuna documento la solución en un patrón de diseño, el cual nosotros solo tenemos que usar. La solución a este problema se llama **Patrón singleton** el cual como su nombre trata de explicar, es un patrón que nos permite generar una y **solo** una instancia de alguna clase.

APLICACIONES WEB DINÁMICAS CON JAVA

Creando una conexión.

```
20 protected Connection generaConexion() {  
21     String usr = "sys as sysdba";  
22     String pwd = "admin";  
23     String driver = "oracle.jdbc.driver.OracleDriver";  
24     String url = "jdbc:oracle:thin:@//localhost:1521/desafio_ejemplo01";  
25  
26     try {  
27         Class.forName(driver);  
28         conn = DriverManager.getConnection(url,usr,pwd);  
29     } catch (Exception ex) {  
30         ex.printStackTrace();  
31     }  
32     return conn;  
33 }
```

[illegible]

APLICACIONES WEB DINÁMICAS CON JAVA

¿Qué es el patrón Singleton?

- Es uno de los tantos patrones de diseños elaborados por la banda de los 4. De todos los existentes es el más fácil de comprender e implementar, ya que lo único que hace es impedir que se generen nuevas instancias de clases si es que existe una. Está catalogado dentro de los patrones creacionales. Para este proyecto tenemos que:
- Declarar la instancia de Connection como estatic. Con esto esta variable de conexión es parte de la clase y será compartida entre métodos.
- Verificar si el objeto de tipo Connection es nulo o no. En caso de ser nulo generamos la conexión con todo lo que conlleva, y si resulta ser válida, retornamos la misma variable ya que no es necesario crearla nuevamente.

APLICACIONES WEB DINÁMICAS CON JAVA

Verificando que no sea nulo

```
35= protected Connection generaPoolConexion() {
36     Context initContext;
37     if(conn == null) {
38         try {
39             initContext = new InitialContext();
40             DataSource ds = (DataSource) initContext.lookup("java:/comp/env/jdbc/ConexionOracle");
41             try {
42
43                 conn = ds.getConnection();
44
45                 System.out.println("CREACION DE CONEXION CON GetConnection");
46             } catch (SQLException e) {
47                 e.printStackTrace();
48             }
49             } catch (NamingException e) {
50                 e.printStackTrace();
51             }
52             return conn;
53         } else {
54             return conn;
55         }
56     }
57 }
58 }
```


APLICACIONES WEB DINÁMICAS CON JAVA

Una conexión creada

```
Sep 28, 2019 9:50:18 PM org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactory getObjectInstance
WARNING: Name = ConexioOracle Property maxActive is not used in DBCP2, use maxTotal instead. maxTotal default
Sep 28, 2019 9:50:18 PM org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactory getObjectInstance
WARNING: Name = ConexioOracle Property maxWait is not used in DBCP2 , use maxWaitMillis instead. maxWaitMilli
Sep 28, 2019 9:50:18 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a
Sep 28, 2019 9:50:18 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/ControlaDeptosEmpresa] is completed
CREACION DE CONEXION CON GetConnection
```

APLICACIONES WEB DINÁMICAS CON JAVA

Patrón dao

- El patrón dao es muy simple, y se verá cuando se descubra quien llama a esta jerarquía de clases. Se implementa un servlet que recibe un request con la llamada a la petición de departamentos.

```
1 package com.desafiolatam.servlet;
2
3 import java.io.IOException;
4
15
16 /**
17  * Servlet implementation class ProcesaDepartamento
18  */
19 @WebServlet("/procesaDepartamento")
20 public class ProcesaDepartamento extends HttpServlet {
21     private static final long serialVersionUID = 1L;
22
23
24     /**
25      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
26      */
27     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
28         // TODO Auto-generated method stub
29         List<Departamento> listaDeptos = new ArrayList<Departamento>();
30         DepartamentoDaoImp deptoDao = new DepartamentoDaoImp();
31         listaDeptos = deptoDao.obtieneDepartamento();
32         request.setAttribute("deptoDao", listaDeptos);
33         request.getRequestDispatcher("resultado.jsp").forward(request, response);
34     }
35
36 }
37
38
```



MUCHAS GRACIAS

Certified



Corporation®

