

Programación

Hito Individual

Sergio Felipe Garcia

12 de Mayo del 2023

Algoritmo	2
Proceso de construcción	2
Arquitectura y selección de herramientas de desarrollo	2
Pasos	3
Requerimientos:	3
Diseño de la interfaz de usuario	3
Introducción datos abonado	4
Registro de los datos en BBDD	6
Cálculo de Precios	8
Desarrollo de pruebas	9
Paradigmas de programación	9
Métodos procedimentales:	9
Procedimientos	9
Organización	9
Ventajas	10
Desventajas	10
Orientación a objetos:	11
POO	11
Polimorfismo	12
Herencia	12
Encapsulamiento	12
Basado en eventos	15
Un Evento	15
Flujo del programa	15
Control de los eventos	15
Implementación de algoritmo con un IDE	16
¿En que me ayuda?	16
Depuración	17
Normas de codificación	19
La convención de nombres	19
Comentarios	19
Constantes	20

Algoritmo

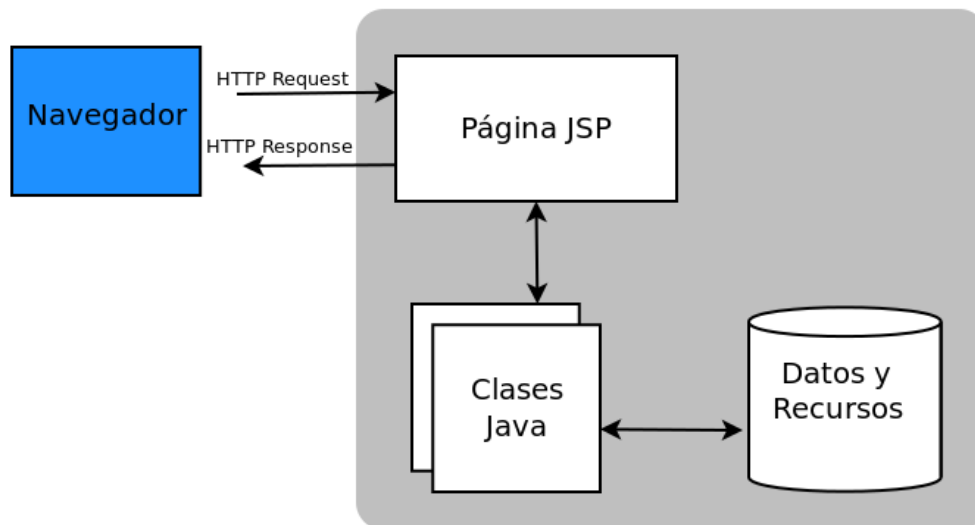
Un algoritmo es un conjunto ordenado y finito de instrucciones precisas y bien definidas que, al ser ejecutadas, permiten resolver un problema o realizar una tarea específica. En otras palabras, es un conjunto de pasos lógicos y matemáticos que se siguen para alcanzar un objetivo determinado.

Proceso de construcción

Para llevar a cabo la aplicación que se nos ha solicitado vamos utilizar una arquitectura simple de aplicación web en un entorno de desarrollo java.

Arquitectura y selección de herramientas de desarrollo

Las herramientas de desarrollo que he entendido son las adecuadas para el proyecto. Son: Eclipse (Java) , HTML, CSS, Servidor web Apache, BBDD MariaDB.





Pasos

Requerimientos:

Lo primero es entender los requisitos de la aplicación. En este caso, se ha solicitado un programa que permita a los usuarios ingresar información relacionada con sus entrenamientos y competiciones, y muestre una lista detallada de los gastos del mes, el costo total.

Diseño de la interfaz de usuario

Una vez entendidos los requerimientos, se procede al diseño de la interfaz de usuario. La aplicación debe ser sencilla, accesible y usable. En este caso, se trata de una aplicación web basada en Java.

Primeramente vamos a realizar la capa de presentación y utilizaré bootstrap para que la aplicación pueda ser consultada desde todos los dispositivos.

Añadiré un logo de la compañía para darle un aspecto más profesional a mi página

Incluiré una barra de navegación para facilitar al usuario final la realización de todas las acciones que se solicitan en la aplicación.

También añadido una página con información general de las tarifas.

Introducción datos abonado

Se presentará una página JSP [add-evento-cliente.jsp](#) donde el usuario podrá introducir sus datos con el plan de entrenamiento, su nombre y sus eventos del mes.

1. Crear una clase principal [Cliente.java](#) en el modelo.
2. Definir las variables necesarias para almacenar la información del usuario.

```
package com.empresa.modelo;

public class Cliente {
    public String nombre;
    public String plan;
    public int horas;
    public String peso;
    public int eventos;

    public Cliente( String nombre, String plan, int horas,String peso,int eventos) {
        super();
        this.nombre=nombre;
        this.plan= plan;
        this.horas = horas;
        this.peso= peso;
        this. eventos= eventos;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getPlan() {
        return plan;
    }

    public void setPlan(String plan) {
        this.plan = plan;
    }
}
```

En la interfaz desarrollamos un formulario para que el usuario pueda registrar sus datos

```

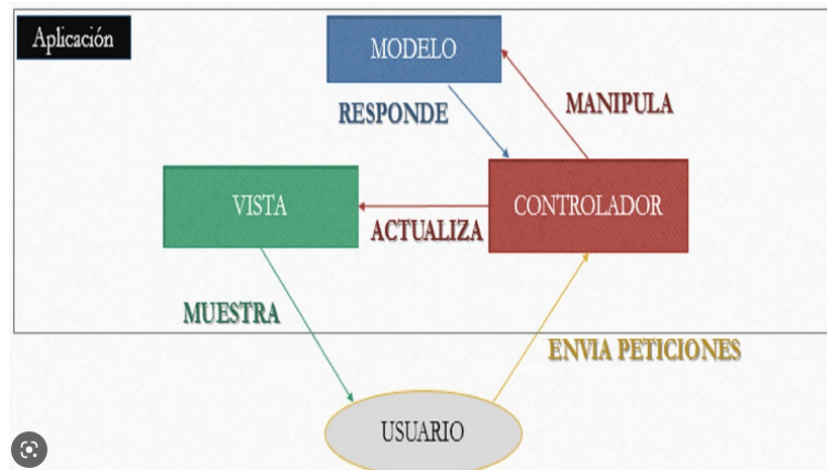
</nav>
<div class="container">
  <h2><u>Introduzca sus datos para registrarse en nuestra plataforma</u></h2>
  <form action="add" method="POST">
    <div class="form-group">
      <label for="nombre">Nombre y Apellidos:</label>
      <input type="text" class="form-control" id="nombre" placeholder="Nombre y Apellidos" name="nombre" required>
    </div>
    <br>
    <div class="form-group">
      <select class="form-select" aria-label="Default select example" id="plan" name="plan" required>
        <option selected>Plan de trabajo</option>
        <option value="Beginner">PLAN BEGINNER (2 sesiones a la semana) tarifa semanal 25 Libras</option>
        <option value="Intermediate">PLAN INTERMEDIATE (3 sesiones a la semana) - tarifa semanal 30 Libras</option>
        <option value="Elite">PLAN ELITE (3 sesiones a la semana) - tarifa semanal 35 Libras</option>
      </select>
    </div>
    <br>
    <label for="horas-extra">¿Quieres añadir un número de clases particulares - Precio hora (Máx 5 horas semana)</label>
    <input type="number" id="horas" name="horas" value="0"><br><br>
    <div class="form-group">
      <label for="peso">Indica cuanto pesas:</label>
      <select class="form-select" aria-label="Default select example" id="peso" name="peso" required>
        <option value="66">peso inferior a 66 Peso Mosca</option>
        <option value="66-73">66-73 Peso Ligero</option>
        <option value="73-81">73-81 Ligero de peso medio</option>
        <option value="81-90">81-90 Peso Medio</option>
        <option value="90-100">superior a 90 kg y menor o igual a 100 kg Peso Semi Pesado</option>
        <option value="100">más de 100 Peso Pesado</option>
      </select>
    </div>
  </form>
</div>

```

Registro de los datos en BBDD

Una vez que el abonado haya registrado sus eventos estos serán registrados en una base de datos para posteriormente poder realizar el cálculo de los costes que se le van a aplicar dependiendo de los eventos que el usuario haya registrado

Para el manejo de la BBDD, tanto para la conexión, como para la consulta e inserción de registros, he utilizado un MODELO-VISTA-CONTROLADOR.



Más o menos los pasos para realizar por ejemplo una consulta en la bbdd las acciones serían:

- El usuario al hacer clic en el botón de la página desde el navegador, envía una petición al controlador.
- El controlador hace una solicitud al modelo los datos
- El modelo devuelve los datos al controlador el cual solicita una vista concreta.
- Se devuelve la vistas que ha solicitado el controlador
- Y por último se devuelve una vista que carga los datos.

En la siguiente imagen se muestra donde se encuentra el código de los pasos anteriormente resumidos:



A parte de esto. También creo que es importante mencionar que estamos utilizando un **servidor web** (Apache) que soporta servlets y JSP. No todos lo soportan.

Para que funcione esta clase `ClienteController.java` es necesario importar las siguientes librerías de Java.

```
import java.io.IOException;
import javax.servlet.ServletConfig;
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 com.empresa.dao.ClienteDAO;
import com.empresa.model.Cliente;
```


Para encontrar información sobre servlets y JSP, son de utilidad las siguientes direcciones:

- <http://java.sun.com/j2ee/>
- <http://java.sun.com/products/jsp:>
- <http://java.sun.com/products/servlets:>

Cálculo de Precios

Cuando un abonado desee calcular los costes en los que ha incurrido, deberá acceder desde la barra de navegación a la página [select-cliente.jsp](#) en la cual hay un formulario para que introduzca su nombre.

Al dar al botón calcular extraemos los costes que el abonado ha registrado previamente en BBDD y realizaremos los cálculos y los mostraremos en la página [list-clientes.jsp](#).

```
// Obtén los datos del request
String nombre = (String) request.getAttribute("nombre");
String plan = (String) request.getAttribute("plan");

int horas = (int) request.getAttribute("horas");

String peso = (String) request.getAttribute("peso");

int eventos = (int) request.getAttribute("eventos");


// Calcula el precio de las competiciones
double precioEventos = eventos * 22.00;

String precioEventosConSimbolo = String.format("%.2f€", precioEventos);

double precioHoras = horas * 9.50;

String precioHorasConSimbolo = String.format("%.2f€", precioHoras);
```

Desarrollo de pruebas

Con el fin de asegurarse de que los algoritmos funcionan correctamente y que la aplicación cumple con los requisitos del cliente. He introducido en el código algunas comprobaciones por ejemplo: mostrar en pantalla los datos introducidos por un abonado en el formulario.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String email = request.getParameter( string:"email");
    String password = request.getParameter( string:"password");
    boolean policy = Boolean.parseBoolean( s:request.getParameter( string:"policy"));
    //esto es para comprobar que los datos se reciben correctamente los imprimo
    System.out.println("email:" + email);
    System.out.println("password:" + password);
    System.out.println("policy:" + policy);
}
```

```
public Connection conectar() {
    System.out.println("hola funciona!");
    Connection connection = null;
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        connection = DriverManager.getConnection(endpoint, user, pass);
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
    }
}
```

Paradigmas de programación

Métodos procedimentales:


Procedimientos

Son Programas/Procedimientos formados por una serie de instrucciones que describen paso a paso el flujo a seguir para resolver un problema. Las instrucciones se ejecutan unas a continuación de otras en el orden en que han sido escritas.

Un procedimiento (un método en lenguaje Java) contiene una serie de instrucciones que realizan una tarea muy concreta.

Organización

Los datos y las acciones a realizar sobre ellos son cosas distintas.



Se definen por un lado los datos y se definen por separado una serie de métodos o procedimientos que operan sobre ellas.

Ventajas

Un pequeño trozo de código que se puede llamar en diferentes partes del programa, en lugar de repetir el mismo código en varios lugares del programa.

Si el problema está en un procedimiento llamado, entonces podemos encontrar el error fácilmente y eliminarlo rápidamente.

Facilita la división de las tareas en un equipo de programadores.

Desventajas

Cuando definimos una variable dentro de una función, esa variable, solo es válida dentro de la función.

Un abuso de funciones, nos puede llevar a un programa difícil de entender.

El paradigma procedimental lo usado por ejemplo en la clase ClienteDAO para definir una serie de pasos a seguir para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos.

Los métodos definidos en esta clase, cómo `conectar()`, `insertCliente()` y `publicarCliente()`, podríamos llamarlos también funciones, siguen un flujo secuencial de pasos para lograr su objetivo, siguiendo una lógica procedimental de programación estructurada.

```

// Paradigma procedimental: se implementan los métodos CRUD de la clase ClienteDAO definiendo una serie de pasos secuenciales para crear,
// leer, actualizar y eliminar clientes en la base de datos
public void insertCliente(Cliente c) {

    // try-with-resource statement will auto close the connection.
    Connection connection = conectar();
    PreparedStatement ps;
    try {

        ps = connection.prepareStatement("INSERT INTO abonados VALUES (?, ?, ?, ?, ?);");
        ps.setString(1, c.getNombre());
        ps.setString(2, c.getPlan());
        ps.setInt(3, c.getHoras());
        ps.setString(4, c.getPeso());
        ps.setInt(5, c.getEventos());
        ps.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// Paradigma basado en eventos: se utilizan objetos como Connection, PreparedStatement y ResultSet para realizar operaciones con la base de datos
// y recibir eventos que indican si la operación se realizó con éxito o no. Además, se emplea el try-with-resources statement
// para cerrar automáticamente las conexiones y liberar los recursos utilizados.
public Cliente publicarCliente(String usuario) {

    Connection connection = conectar();

    PreparedStatement ps;

    ResultSet rs;

    Cliente cliente = null;

    try {

        ps = connection.prepareStatement("SELECT * FROM abonados WHERE nombre = ?;");

        ps.setString(1, usuario);

        rs = ps.executeQuery();

        if (rs.next()) {

            String nombre = rs.getString("nombre");
            String plan = rs.getString("plan");

            int horas = rs.getInt("horas");

            String peso = rs.getString("peso");

            int eventos = rs.getInt("eventos");


            // Crear el objeto Cliente con los datos obtenidos
            cliente = new Cliente(nombre, plan, horas, peso, eventos);
        }
    }
}

```

Orientación a objetos:

POO

Este paradigma organiza el código en clases (estructuras de datos), para posteriormente, a partir de esa clase crear diferentes instancias de un objeto



reutilizando dicho código y evitando duplicidades.

Polimorfismo

Ocurre cuando hay una jerarquía de clases relacionadas entre sí a través de la herencia permite diseñar objetos para compartir comportamientos entre objetos de diferentes clases. Creo que esto no lo he utilizado.

Herencia

Permite reutilizar el código programado en cada clase “heredando” o las características de un objeto a sus “descendientes”.

Encapsulamiento

Protege la información de los atributos de una clase para que no se pueda ver o modificar la información del objeto. Para ello, es necesario utilizar métodos para recuperar la información (getters) y (setters) para asignar un nuevo valor.

El paradigma orientado a objetos se utiliza en la clase *Cliente.java* para modelar los datos de un cliente con sus respectivas propiedades y métodos.

```
public class Cliente {
    public String nombre;
    public String plan;
    public int horas;
    public String peso;
    public int eventos;

    public Cliente( String nombre, String plan, int horas,String peso,int eventos) {
        super();
        this.nombre=nombre;
        this.plan= plan;
        this.horas = horas;
        this.peso= peso;
        this. eventos= eventos;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getPlan() {
        return plan;
    }

    public void setPlan(String plan) {
        this.plan = plan;
    }

    public int getHoras() {
        return horas;
    }
}
```

En la clase *ClienteDAO.java* se utiliza la instancia de un objeto de la clase *Cliente* para pasar datos de clientes a los métodos CRUD y para devolver datos de la base de datos como un objeto de la clase *Cliente*.

```
public void insertCliente(Cliente c) {  
    // try-with-resource statement will auto close the connection.  
    Connection connection = conectar();  
    PreparedStatement ps;  
    try {  
        ps = connection.prepareStatement("INSERT INTO abonados VALUES (?, ?, ?, ?, ?);");  
        ps.setString(1, c.getNombre());  
        ps.setString(2, c.getPlan());  
        ps.setInt(3, c.getHoras());  
        ps.setString(4, c.getPeso());  
        ps.setInt(5, c.getEventos());  
        ps.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Además, se utiliza la encapsulación para definir los métodos get y set de las propiedades del objeto cliente.

Basado en eventos

Un Evento

Es algo, una condición que ocurre mientras el programa está en ejecución. Normalmente el evento requiere realizar alguna acción por parte del sistema. Por ejemplo, un evento puede requerir que el programa muestre información , o realice algún cálculo.

Flujo del programa

El flujo viene dado por eventos que pueden ser acciones del usuario, mensajes de otros programas, etc.

Control de los eventos

El programa que se activa para reaccionar ante un evento. Es decir, es una función o método que ejecuta una acción específica cuando se activa un evento determinado. Por ejemplo, un botón que cuando el usuario haga clic en él muestre un mensaje. El evento podría *ser = Onclick*

Aplicaré este paradigma a la hora de la creación de la interfaz gráfica de usuario (GUI), en mi proyecto.

```
<form action="select" method="POST">
  <div class="form-group">
    <label for="nombre">Nombre y Apellidos:</label>
    <input type="text" class="form-control" id="nombre" placeholder="Nombre y Apellidos" name="nombre" required>
  </div>
  <br>
  <button type="submit" class="btn btn-primary">Calcular</button>
</form>
```

No se utiliza directamente en el código fuente proporcionado, ya que este paradigma está más relacionado con el diseño de interfaces gráficas de usuario y el manejo de eventos asociados a ellas.

Sin embargo, se podría decir que la llamada a los métodos CRUD desde la capa de interfaz de usuario podría ser considerada una forma simplificada de programación dirigida por eventos, donde los eventos se disparan por las acciones del usuario en la interfaz gráfica y se procesan en la capa de datos.



Implementación de algoritmo con un IDE

¿En que me ayuda?

Un IDE, en este caso Eclipse, recoge un conjunto de herramientas y funcionalidades que permite facilitar las cosas a los desarrolladores. Por ejemplo, yo he aprendido que:.

- Tiene un editor de texto con el resaltado de la sintaxis.
- Dispone un asistente para la creación de proyectos web, también tiene asistentes para crear clases.
- Al igual que VScode dispone de algunos plugins para aumentar su funcionalidad y escribir en otros lenguajes o tipo de ficheros.

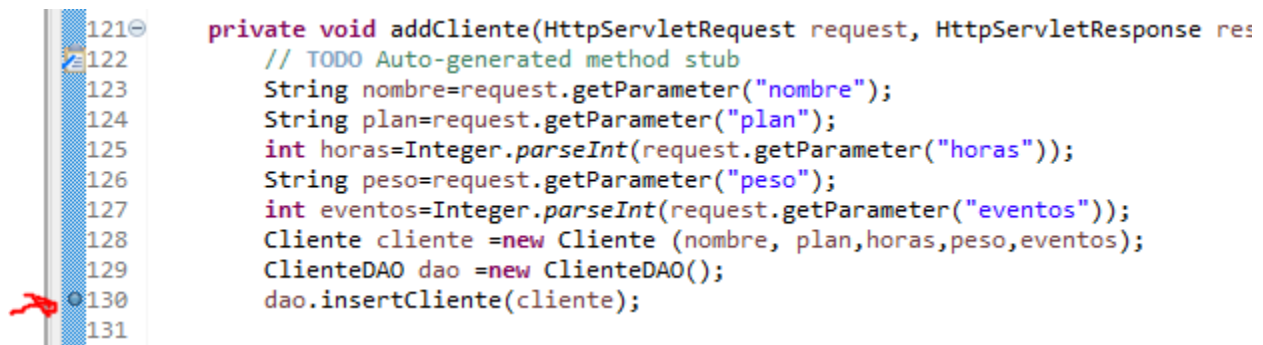
Sin embargo, aunque me he decidido en un principio utilizar el IDE de Eclipse, he realizado alguna prueba con Netbeans y me ha resultado un poco más sencillo de utilizar.

Depuración

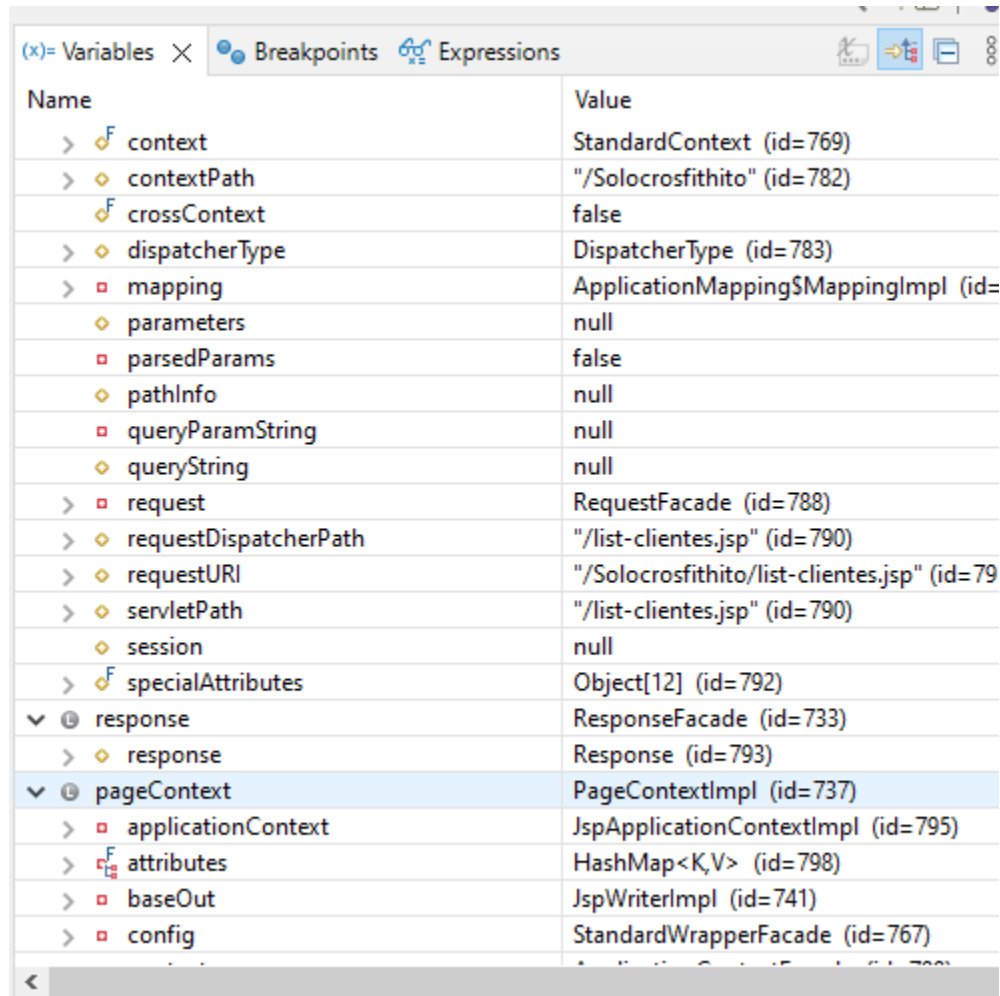
La depuración es el proceso de encontrar y corregir errores en el código fuente de un programa. Este proceso implica ejecutar el programa en un entorno de depuración especial que permite al desarrollador detener la ejecución del programa en cualquier momento, examinar el estado de las variables y las estructuras de datos, y seguir la ejecución línea por línea para identificar errores.

El IDE Eclipse proporciona varias facilidades de depuración para ayudar a los desarrolladores a encontrar y corregir errores en sus programas. Algunas de estas facilidades son:

1. **Breakpoints:** Un breakpoint es un punto de interrupción en el código fuente donde se detendrá la ejecución del programa cuando se alcanza. Los desarrolladores pueden establecer breakpoints en cualquier línea de código y, cuando se alcanza el breakpoint, el programa se detiene y el desarrollador puede examinar el estado de las variables y las estructuras de datos en ese punto.



2. **Variables View:** La vista de variables muestra el estado de las variables y las estructuras de datos en un punto específico en la ejecución del programa. Los desarrolladores pueden examinar las variables y las estructuras de datos para encontrar errores en el código.



The screenshot shows the Eclipse IDE's 'Variables View' window. The window has tabs for '(x)= Variables', 'Breakpoints', and 'Expressions'. The 'Variables' tab is active, displaying a table of variables and their values. The table has two columns: 'Name' and 'Value'. The variables are listed in a tree-like structure, with expandable icons (chevrons) next to them. The 'pageContext' variable is expanded, showing its sub-variables: 'applicationContext', 'attributes', 'baseOut', and 'config'. The 'attributes' variable is further expanded, showing its value as a 'HashMap<K,V>'.

Name	Value
> context	StandardContext (id=769)
> contextPath	"/Solocrosfithito" (id=782)
crossContext	false
> dispatcherType	DispatcherType (id=783)
> mapping	ApplicationMapping\$MappingImpl (id=784)
parameters	null
parsedParams	false
pathInfo	null
queryParamString	null
queryString	null
> request	RequestFacade (id=788)
> requestDispatcherPath	"/list-clientes.jsp" (id=790)
> requestURI	"/Solocrosfithito/list-clientes.jsp" (id=791)
> servletPath	"/list-clientes.jsp" (id=790)
session	null
> specialAttributes	Object[12] (id=792)
response	ResponseFacade (id=733)
> response	Response (id=793)
pageContext	PageContextImpl (id=737)
> applicationContext	JspApplicationContextImpl (id=795)
> attributes	HashMap<K,V> (id=798)
> baseOut	JspWriterImpl (id=741)
> config	StandardWrapperFacade (id=767)

3. **Debug Perspective:** Eclipse tiene una perspectiva de depuración que proporciona una vista detallada del estado del programa durante la depuración. La perspectiva de depuración incluye vistas de código fuente, variables y ejecución del programa.
4. **Step Debugging:** Los desarrolladores pueden ejecutar el programa paso a paso, siguiendo la ejecución línea por línea, para identificar errores en el código.

5. **Inspect:** El inspector permite a los desarrolladores examinar el estado de las variables y las estructuras de datos en tiempo de ejecución sin detener la ejecución del programa.

Normas de codificación

Las normas de codificación que he utilizado son :

La convención de nombres

He utilizado la notación CamelCase la he utilizado para nombrar variables y métodos, y consiste en escribir las palabras juntas, comenzando cada palabra con una letra mayúscula, excepto la primera o también en al revés. Por ejemplo, *"nombreCliente" o "obtenerDatosUsuario()""init", "doGet", "selectCliente", "listCliente", "updateCliente" y "addCliente".*

Comentarios

He utilizado el uso de comentarios para explicar el propósito de los métodos y bloques de código en el código fuente. Los comentarios son precedidos por doble barra "//" en línea y por `"/ **"` y `"*/"` para comentarios de varias líneas.

```
// Obtén los datos del request
String nombre = (String) request.getAttribute("nombre");
String plan = (String) request.getAttribute("plan");
int horas = (int) request.getAttribute("horas");
String peso = (String) request.getAttribute("peso");
int eventos = (int) request.getAttribute("eventos");

// Calcula el precio de las competiciones
double precioEventos = eventos * 22.00;
String precioEventosConSimbolo = String.format("%.2f", precioEventos);
double precioHoras = horas * 9.50;
String precioHorasConSimbolo = String.format("%.2f", precioHoras);

// Calcula el precio dependiendo del plan
double precioPlan = 0.0;
if (plan.equals("beginner")) {
    precioPlan = 5.0;
} else if (plan.equals("intermediate")) {
    precioPlan = 10.0;
} else if (plan.equals("elite")) {
    precioPlan = 15.0;
}
```



Constantes

He utilizado constantes definidas en mayúsculas, como por ejemplo `"/"`, que se utiliza en la anotación `@WebServlet("/")` para indicar la URL raíz del servlet.

Aunque no se definen explícitamente en una sección separada de constantes, también se pueden considerar como tales los nombres de las acciones que se manejan en la sentencia switch-case del método `doGet`, ya que son valores que no cambian durante la ejecución del programa y se utilizan para tomar decisiones en diferentes partes del código. Por ejemplo, `"add"`, `"update"`, `"select"`, etc.