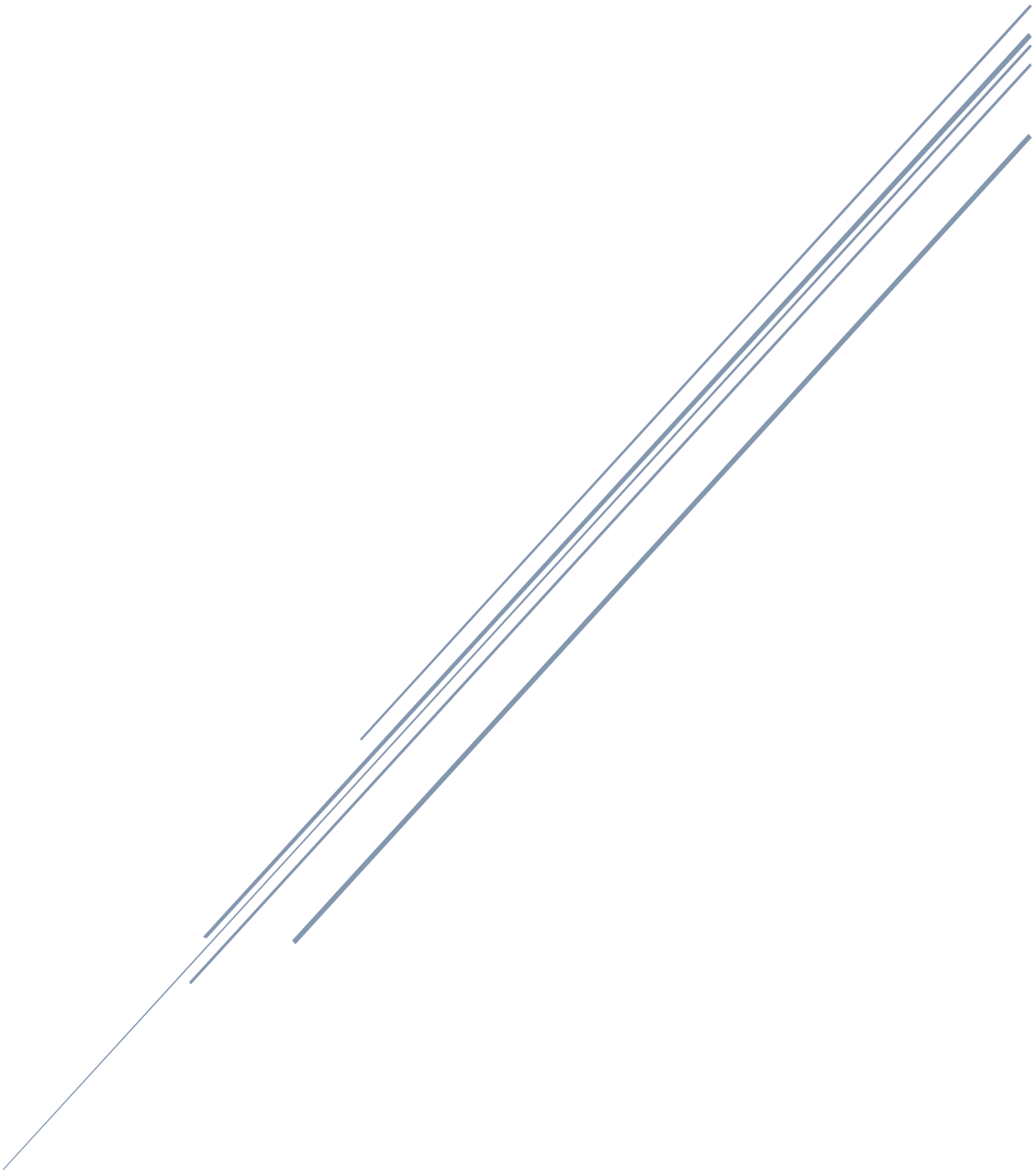
HITO INDIVIDUAL

Programación



David Millán Domínguez
1ºDAM

Índice

Algoritmo. P1

El proyecto consiste en un formulario que registra a los clientes en una base de datos y una tabla que muestra a todos los clientes. Para garantizar el éxito del proyecto lo primero que tendremos que hacer será definir un algoritmo:

1. Lo primero será crear la base de datos.
2. Tendremos que organizar la estructura del programa. Serán tres archivos que estarán en el paquete com.empresa: la clase **UsuarioDAO**, la clase **Usuario** y el servlet **UsuarioController**.
3. En **UsuarioDAO** estableceremos la conexión con la base de datos.
4. En **Usuario** se declararán: los atributos, el constructor, los **getters** y los **setters**.
5. En **UsuarioController** haremos la recogida de datos del formulario, la creación de usuarios en la base de datos, la recogida de los datos y el envío de estos para la creación de la tabla de los usuarios.
6. Crearemos dos jsp (Java Server Page): formulario.jsp tendrá el formulario que mandará los datos al servlet y lista-usuarios.jsp tendrá un bucle que crea la tabla donde estarán los usuarios.

Pasos desde la escritura hasta la ejecución del código. P2

Después de la escritura del código no solo se ejecuta el programa. Estos son los pasos que se dan desde la escritura de código hasta su ejecución:

- Compilación del código fuente: se compila el código en un archivo que puede ser interpretado por la máquina virtual de Java.
- Empaquetado de la aplicación: se empaquetan los archivos, los recursos y las vistas en un archivo WAR que se puede implementar en un servidor.
- Implementación en el servidor: se implementa la aplicación en un servidor, este puede set Tomcat, y es responsable de ejecutar la aplicación web.

En qué consisten los métodos procedimentales, orientados a objetos y los paradigmas basados en eventos y su relación. P3

Métodos procedimentales: se basan en la ejecución de una serie de instrucciones organizadas en funciones que realizan tareas específicas.

Características:

- La función es el principal elemento.
- Las variables y los datos son organizados en estructuras de datos.
- Es fácil de diseñar para programas pequeños.

Métodos orientados a objetos: se centran en la creación de objetos que contienen datos y funciones. Cada objeto es una clase instanciada que define los datos y métodos que contiene el objeto.

Características:

- Se enfoca principalmente en el objeto.
- Las variables y los datos son encapsulados.
- Es más fácil de diseñar para programas grandes.
- La clase es el elemento central.

Paradigmas basados en eventos: se centran en la creación de programas que detectan eventos ocurridos durante la ejecución del programa.

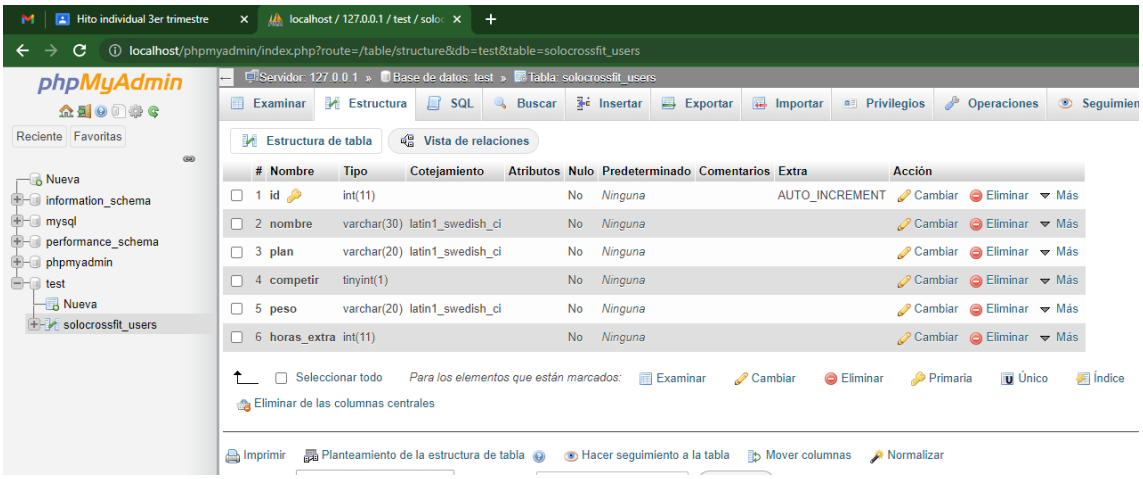
Características:

- Se enfoca en los eventos.
- Detecta eventos a tiempo real.
- No hay una secuencia fija.

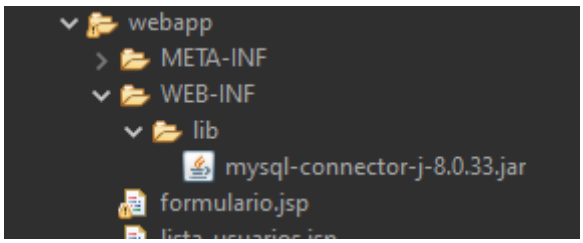
Los métodos procedimentales, los orientados a objetos y los paradigmas basados en eventos son cosas diferentes y se utilizan para resolver problemas diferentes. Esto permite muchas más posibilidades a la hora de desarrollar un programa, pues aunque cada uno tiene sus limitaciones, se pueden combinar para obtener mejores resultados en algunos proyectos.

Escribir un programa que implemente un algoritmo y analizar el proceso de escritura de código. P4 y M1

Lo primero es crear la base de datos y la tabla donde se van a guardar los datos. He creado la tabla “solocrossfit_users” en la base de datos por defecto “test”.



Lo siguiente es hacer la conexión a la base de datos, para ello se necesita implementar el driver necesario a la carpeta “lib” del proyecto.



Creamos la clase UsuarioDAO (Data Access Object). Aquí estarán varios métodos, los más importantes ahora serán “conectar()”, “insertUsuario()” y “selectUsuario()”.

Función “conectar()”:

```java
public class UsuarioDAO {
    private String endpoint="jdbc:mysql://localhost:3306/test";
    private String usuario="root";
    private String pass="";

////PARADIGMA PROCEDIMENTAL ////
    //Función conectar
    public Connection conectar(){
        Connection connection = null;
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection(endpoint, usuario, pass);
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        return connection;
    }//cierra conectar
```

Función “insertUsuario()”:

```java
////PARADIGMA PROCEDIMENTAL ////
   //Funcion insertar
   public void insertUsuario(Usuario c) {
       // try-with-resource statement will auto close the connection.
           Connection connection = conectar();
           PreparedStatement ps;
           try {
               ps = connection.prepareStatement("INSERT INTO solocrossfit_users (id, nombre, plan, competir, peso, horas_extra) VALUES (NULL,?,?,?,?,?);");
               ps.setString(1, c.getNombre());
               ps.setString(2, c.getPlan());
               ps.setInt(3, c.getCompetir());
               ps.setString(4, c.getPeso());
               ps.setInt(5, c.getHorasExtra());
               ps.executeUpdate();
           } catch (SQLException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
           }
   }//cierra insertar

   //resto de métodos crud
```

Función “selectUsuario()”:

Este método tiene varios **switches** para definir algunos datos que se van a mostrar en la tabla de la web.

```java
////PARADIGMA PROCEDIMENTAL ////
    //Funcion select usuario
    public List <Usuario> selectUsuario() {

        List <Usuario> users = new ArrayList < > ();
        Connection connection = conectar();
        PreparedStatement ps;
        try {
            ps = connection.prepareStatement("SELECT * FROM solocrossfit_users;");
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                String nombre = rs.getString("nombre");
                String plan = rs.getString("plan");
                int competir = rs.getInt("competir");
                String peso = rs.getString("peso");
                int horasExtra = rs.getInt("horas_extra");
                int precioPlan = 0;
                int precioHoras = 0;
                int precioCompetir = 0;
                int total;
                String categoria = "";

                //precio del plan
                switch (plan) {
                case "iniciado":
                    precioPlan = 25;
                    break;
                case "intermedio":
                    precioPlan = 30;
                    break;
                case "avanzado":
                    precioPlan = 35;
                    break;
                }//cierra switch
```

```
                //precio competir
                switch (competir) {
                case 0:
                    precioCompetir = 0;
                    break;
                case 1:
                    precioCompetir = 22;
                    break;
                }//cierra switch

                total = precioHoras+precioCompetir+precioPlan;

                //categoria y peso
                switch (peso) {
                case "pluma":
                    peso = "Menor de 66";
                    categoria = "pluma";
                    break;
                case "ligero":
                    peso = "67-73";
                    categoria = "ligero";
                    break;
                case "medioLigero":
                    peso = "74-81";
                    categoria = "medioLigero";
                    break;
                case "medio":
                    peso = "82-90";
                    categoria = "medio";
                    break;
                case "ligeroPesado":
                    peso = "91-100";
                    categoria = "ligeroPesado";
                    break;
                case "pesado":
                    peso = "Mayor de 100";
                    categoria = "pesado";
                    break;
                }

                users.add(new Usuario(nombre, plan, precioPlan, peso, categoria, precioCompetir, horasExtra, precioHoras, total));
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return users;
    }
}//cierra DAO
```

La clase Usuario tiene 4 partes: la definición de atributos, dos constructores y los **setters/getters**.

```
//// PARADIGMA ORIENTADO A OBJETOS ////
public class Usuario {
    public String nombre;
    public String plan;
    public int competir;
    public String peso;
    public int horasExtra;
    public String categoria;
    public int total;
    public int precioPlan;
    public int precioHoras;
    public int precioCompetir;
```

Cada uno de los constructores tiene una función específica. Si nos fijamos en el DAO, la instancia de la clase usuario del método “insertUsuario()” llama al primer constructor mientras que la instancia del método “selectUsuario()” llama al segundo constructor. Esto lo hago porque me conviene más guardar unos datos u otros según el proceso que se esté ejecutando.

```
    //constructor para la función add
    public Usuario(String nombre, String plan, int competir, String peso, int horasExtra) {
        super();
        this.nombre = nombre;
        this.plan = plan;
        this.competir = competir;
        this.peso = peso;
        this.horasExtra = horasExtra;
    }

    //constructor para la función select
    public Usuario(String nombre, String plan, int precioPlan, String peso, String categoria, int precioCompetir, int horasExtra, int precioHoras, int total) {
        super();
        this.nombre = nombre;
        this.plan = plan;
        this.peso = peso;
        this.horasExtra = horasExtra;
        this.categoria = categoria;
        this.total = total;
        this.precioPlan = precioPlan;
        this.precioHoras = precioHoras;
        this.precioCompetir = precioCompetir;
    }
```

La función de estos **getters** y **setters** es la de llamar al atributo de la clase instanciada o por otro lado cámbialo.

```java
// getters y setters
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;
}
```

El servlet “UsuarioController” es el corazón del programa, se encarga de recibir los datos de la web, dirigir el programa, enviar información, llamar a funciones, etc.

La función doGet recibe la información de la web y se encarga de redirigirla hacia la función deseada. Esto se le conoce como enrutador.

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

    /**
     * @see HttpServlet#HttpServlet()
     */
    public UsuarioController() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */

    //// PARADIGMA DIRIGIDO A EVENTOS ////
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        String action = request.getServletPath();
        switch (action) {
        case "/add":
            addUsuario(request, response);
            break;
        case "/update":
            updateUsuario(request, response);
            break;
        case "/delete":
            deleteUsuario(request, response);
            break;
        default:
            listUsuario(request, response);
            break;
        }//cierra switch
    }
```

La función “addUsuario()” guarda la información recibida en sus respectivas variables y hace los pasos necesarios para añadir un usuario.

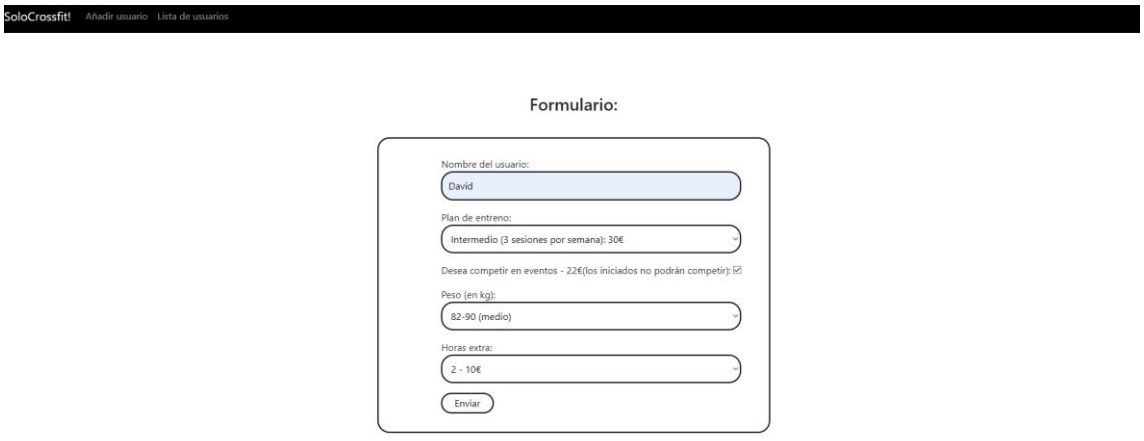
```
//// PARADIGMA PROCEDIMENTAL ////
private void addUsuario(HttpServletRequest request, HttpServletResponse response) {
    // TODO Auto-generated method stub
    String nombre = request.getParameter("nombre");
    String plan = request.getParameter("plan");
    String competirBool = request.getParameter("competir");
    String peso = request.getParameter("peso");
    int horasExtra = Integer.parseInt(request.getParameter("horasExtra"));
    int competir;

    if(plan.equals("iniciado") || competirBool == null){
        competir = 0;
    }else {
        competir = 1;
    }

    Usuario usuario = new Usuario(nombre,plan,competir,peso,horasExtra);
    UsuarioDAO dao = new UsuarioDAO();
    dao.insertUsuario(usuario);
    //dao.selectUsuario();
    try {
        response.sendRedirect("lista-usuarios.jsp");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

La función “listUser()” se encarga de guardar la información recibida del método “selectUsuario()” del DAO en un **array** y lo envía al archivo JSP donde se encuentra la tabla de usuarios.

```
////PARADIGMA PROCEDIMENTAL ////
private void listUser(HttpServletRequest request, HttpServletResponse response){
    UsuarioDAO dao = new UsuarioDAO();
    List < Usuario > listaUsuario = dao.selectUsuario();
    request.setAttribute("listaUsuario", listaUsuario);
    RequestDispatcher dispatcher = request.getRequestDispatcher("lista-usuarios.jsp");
    try {
        dispatcher.forward(request, response);
    } catch (ServletException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

El archivo JSP que contiene el formulario contiene un simple formulario que envía la información a “add” en el servlet.

```
<div class="container" id="contenedor_todo">
    <h3>Formulario:</h3>
    <div class="col-6" id="contenedor_formulario">
        <form action="add" method="post">
            <p>Nombre del usuario: <input type="text" name="nombre" id="nombre_input"></p>
            <p>Plan de entreno: <select name="plan" id="plan_input">
                <option value="iniciado">Iniciado (2 sesiones por semana): 25€</option>
                <option value="intermedio">Intermedio (3 sesiones por semana): 30€</option>
                <option value="avanzado">Avanzado (5 sesiones por semana): 35€</option>
            </select></p>
            <p>Desea competir en eventos - 22€(los iniciados no podrán competir): <input type="checkbox" name="competir" id="competir"></p>
            <p>Peso (en kg): <select name="peso" id="peso_input">
                <option value="pluma">Menor de 66 (pluma)</option>
                <option value="ligero">67-73 (ligero)</option>
                <option value="medioLigero">74-81 (medio ligero)</option>
                <option value="medio">82-90 (medio)</option>
                <option value="ligeroPesado">91-100 (ligero pesado)</option>
                <option value="pesado">Mayor de 100 (pesado)</option>
            </select></p>
            <p>Horas extra: <select name="horasExtra" id="horasExtra">
                <option value="0">0</option>
                <option value="1">1 - 5€</option>
                <option value="2">2 - 10€</option>
                <option value="3">3 - 13€</option>
                <option value="4">4 - 15€</option>
            </select></p>
            <input type="submit" value="Enviar" id="boton_enviar">
        </form>
    </div>
</div>
```



El archivo JSP que contiene la lista de usuarios consta de una tabla y varios scriptlets que con el uso de un **for** muestra todos los datos requeridos de todos los usuarios insertados anteriormente en el **array** de “listUser()”.

```
<h3 class="text-center">Lista de usuarios</h3>
<br>
<table class="table table-bordered">
    <thead>
        <tr>
            <th>Nombre</th>
            <th>Plan</th>
            <th>Peso y categoría</th>
            <th>Compite</th>
            <th>Horas extra</th>
            <th>Total</th>
        </tr>
    </thead>
    <tbody>
        <%
                //List<Usuario> listaUsuarios = (List<Usuario>) request.getAttribute("listaUsuarios");
                UsuarioDAO dao = new UsuarioDAO();
                List <Usuario> listaUsuario = dao.selectUsuario();
                for (Usuario usuario : listaUsuario) {
            %>
            <tr>
                <td><%=usuario.getNombre()%></td>
                <td><%=usuario.getPlan()%> - <%=usuario.getPrecioPlan()%>€</td>
                <td><%=usuario.getPeso()%>kg - <%=usuario.getCategoria()%></td>
                <td><%=usuario.getPrecioCompetir()%>€</td>
                <td><%=usuario.getHorasExtra()%> horas - <%=usuario.getPrecioHoras()%>€</td>
                <td><%=usuario.getTotal()%>€</td>
            </tr>
            <% } %>
        </tbody>
    </table>
</div>
```

Proceso de depuración de Eclipse. P5

La depuración consiste en encontrar y solucionar errores en el código de cualquier software. Cuando se encuentra un error que causa que el programa no funcione correctamente, los programadores investigan en el código para explicar por qué ocurren.

Pasos del proceso de depuración:

- Identificar errores: cuando se encuentra un error, se le notifica al desarrollador y este busca en el código hasta localizar exactamente donde está el error.
- Análisis de errores: se analiza el error para averiguar cómo solucionarlo. En algunos casos se puede priorizar un error u otro en función de su importancia.
- Corrección y validación: el desarrollador corrige el error y hace pruebas sobre el para confirmar que el software funciona correctamente.

Facilidades de depuración de Eclipse:

- Puntos de interrupción: te permite parar la ejecución de un programa en un punto específico.
- Valor de las variables: eclipse permite ver el valor de las variables mientras está ejecutado el programa.
- Vista de depuración: proporciona información sobre el estado actual del programa.
- Paso a paso: permite ejecutar el programa paso a paso.
- Modificación de variables: mientras el programa está ejecutado, Eclipse permite cambiar las variables.

Normas de codificación del código. P6

Las normas de codificación son un grupo de reglas y recomendaciones para el estilo y formato del código.

- Nomenclatura: para las funciones y variables se ha usado camelCase y para las clases se ha usado UpperCamelCase.
- Tabulación: se ha hecho uso de la tabulación para distinguir los bloques dentro del código.
- Comentarios: se han usado los comentarios en el código para describir brevemente que hace cada cosa.
- Uso de excepciones: se han usado varios try/catch

Comparar los paradigmas usados en el código. M2

La clase **UsuarioDAO** es un ejemplo de programación orientada a objetos que incluye los métodos “conectar()”, “insertUsuario()” y “selectUsuario”.

La clase **Usuario** también es programación orientada a objetos, esta clase tiene los dos constructores, los getters y los setters.

En el servlet **UsuarioController** se pueden encontrar tres tipos de paradigmas: las funciones “doGet” y “doPost” son un ejemplo de programación dirigida a eventos, las funciones “addUsuario()” y “listUser” son un ejemplo de programación procedimental y por último la programación orientada a objetos la encontramos en las instancias de las clases “UsuarioDAO” y “Usuario”.

Como ayuda el proceso de depuración a desarrollar aplicaciones más seguras. M3

A la hora de desarrollar una aplicación, el proceso de depuración es muy importante. Pues como hemos dicho antes, te ayuda a identificar errores de software y a arreglarlos.

El proceso de depuración se puede usar para el desarrollo de las siguientes maneras:

- Identificación temprana de errores: cuanto antes identifiquemos los errores más sencillo nos resultará el desarrollo.
- Protección de datos: la depuración puede ayudar a detectar problemas de seguridad como la autenticación y la autorización.
- Pruebas a fondo: al realizar pruebas, se pueden identificar y corregir errores que fueran problemas mayores en el futuro.

Bibliografía

http://contenidos.sucerman.com/nivel3/dispositivos/unidad1/leccion2.html

https://es.wikipedia.org/wiki/Programación_dirigida_por_eventos

https://universidadeuropea.com/blog/programacion-orientada-objetos/

https://ifgeekthen.nttdata.com/es/herencia-en-programacion-orientada-objetos

https://gamedevtraum.com/es/programacion-informatica/programacion-orientada-a-objetos/diferencia-entre-clase-y-objeto/#:~:text=1.,cuando%20el%20programa%20está%20corriendo.

https://ifgeekthen.nttdata.com/es/polimorfismo-en-java-programación-orientada-objetos

https://www.getapp.com.mx/software/2050477/eclipse-ide#:~:text=Eclipse%20IDE%20es%20un%20IDE,de%20aplicaciones%20basadas%20en%20Java.

https://aws.amazon.com/es/what-is/debugging/#:~:text=La%20depuración%20es%20el%20proceso,por%20qué%20ocurren%20algunos%20errores.