Instituto Politecnico Naional
Escuela Superior de Cómputo

Sistemas Distribuidos

Tarea Core J2EE, DTO, DAO, Business Delegate

Juan Carlos Jirón Juárez

2EE es una **plataforma para el cómputo empresarial** a partir de la cual es posible el desarrollo profesional de aplicaciones empresariales distribuidas sobre una arquitectura multicapa, que son escritas con el lenguaje de programación Java y son ejecutadas desde un servidor de aplicaciones.

J2EE es, por tanto, una plataforma de programación cuya especificación original fue desarrollada por la empresa Sun Microsystems, si bien en el año 2000 la compañía Oracle se hizo con su control.
La versión J2EE tiene su origen en el lenguaje de programación Java correspondiendo sus siglas “EE” a “**Enterprise Edition**”. Esta tecnología Java permite a los desarrolladores y programadores crear (escribir) aplicaciones una única vez vez y que sea compatibles sobre cualquier equipo ya que se comunican directamente con la máquina virtual, y no con el sistema operativo.

- Enterprise JavaBeans (EJB).
- Java ServletJavaServer Page (JSP)
- JavaServer Pages Standard Tag Library (JSTL).
- JavaServer Faces (JSF)
- Java Message Service (JMS).
- Java Transaction API (JTA).
- JavaMail API y JavaBeans Activation Framework (JAF).
- Tecnologías XML (JAXP, JAX-RPC, JAX-WS, JAXB, SAAJ, JAXR) JPA, JDBC API
- Java Naming and Directory Interface (JNDI)
- Java Authentication and Authorization Service (JAAS)

EL PATRÓN DATA TRANSFER OBJECT (DTO/VO)

Va de la mano con el patrón de diseño DAO.

Se utiliza para transferir varios atributos entre el cliente y el servidor o viceversa, básicamente consta de 2 clases:

La primera es una clase java conocida como **Value Object** que únicamente contiene sus atributos, constructor, getters y setters, esta clase no tiene comportamiento.
La segunda es una clase del lado del servidor conocida como clase de negocio (**en la implementación también se conoce como Business Object**) es la que se encarga de obtener datos desde la base de datos y llenar la clase **Value Object** y enviarla al cliente, o a su vez recibir la clase desde el cliente y enviar los datos al servidor, por lo general tiene todos los métodos CRUD (create, read, update y delete).

Clase ClienteVO patrón DTO (Data transfer object)

```java

public class ClienteVO {
    private int id;
    private String nombre;
    private String apellido;


    public ClienteVO(int id, String nombre, String apellido) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

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

    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    @Override
    public String toString() {
        return this.getNombre()+" "+this.getApellido();
    }
}
```

Se crea la clase ***ClienteBO.java*** conocida también como la clase de negocio, que es la que contiene todos los métodos CRUD:

```
6  import com.ecodeup.vo.ClienteVO;
7
8  public class ClienteBO {
9
10     //lista de tipo cliente
11     List<ClienteVO> clientes;
12
13
14     //constructor, se guarda en la lista 2 clientes
15     public ClienteBO() {
16         clientes = new ArrayList<>();
17         ClienteVO cliente1= new ClienteVO(,"Elivar","Largo");
18         ClienteVO cliente2= new ClienteVO(1,"Priscila","Morocho");
19         clientes.add(cliente1);
20         clientes.add(cliente2);
21     }
22
23     //elimina el cliente que se le pasa como paraámetro
24     public void eliminarCliente(ClienteVO cliente) {
25         clientes.remove(cliente.getId());
26         System.out.println("Cliente "+cliente.getId()+" eliminado satisfactoriamente");
27     }
28
29     //obtiene toda la lista de clientes
30     public List<ClienteVO> obtenerClientes(){
31         return clientes;
32     }
33
34     //obtiene un cliente de acuerdo al id pasado como parámetro
35     public ClienteVO obtenerCliente(int id) {
36         return clientes.get(id);
37     }
38
39     // actualiza el cliente que se le pasa como parámetro
40     public void actualizarCliente(ClienteVO cliente) {
41         clientes.get(cliente.getId()).setNombre(cliente.getNombre());
42        clientes.get(cliente.getId()).setApellido(cliente.getApellido());
43         System.out.println("Cliente id: "+ cliente.getId()+" actualizado satisfactoriamente");
44     }
```

Finalmente probamos el patrón Data Transfer Object:

```java
package com.ecodeup.dto;

import com.ecodeup.bo.ClienteBO;
import com.ecodeup.vo.ClienteVO;

public class DTODemo {
    public static void main(String[] args) {
        //objeto business object
        ClienteBO clienteBusinessObject = new ClienteBO();

        //obtiene todos los clientes
        clienteBusinessObject.obtenerClientes().forEach(System.out::println);

        // actualiza un cliente
        System.out.println("****");
        ClienteVO cliente = clienteBusinessObject.obtenerCliente();
        cliente.setNombre("Luis");
        clienteBusinessObject.actualizarCliente(cliente);

        // obtiene un cliente
        System.out.println("****");
        cliente=clienteBusinessObject.obtenerCliente();
        System.out.println(cliente);

        //elimina un cliente
        System.out.println("****");
        cliente=clienteBusinessObject.obtenerCliente();
        clienteBusinessObject.eliminarCliente(cliente);
    }
}
```

EL PATRÓN DATA ACCES OBJECT (DAO)

El problema que viene a resolver este patrón es netamente el acceso a los datos, que básicamente tiene que ver con la gestión de diversas fuentes de datos y además abstrae la forma de acceder a ellos.

Imagínate que tienes un sistema montado en producción con una base de datos MySQL y de pronto lo debes cambiar a PostgreSQL o a cualquier otro motor de base de datos.

Eso puede ser un verdadero problema.

Y precisamente esto lo que soluciona este patrón, tener una aplicación que no esté ligada al acceso a datos, que si por ejemplo la parte de la vista pide encontrar los clientes con compras mensuales mayores $ 200, el **DAO** se encargue de traer esos datos independientemente si está en un archivo o en una base de datos.

La capa DAO contiene todos los métodos CRUD (create, read, update, delete), por lo general se tiene un DAO para cada tabla en la base de datos, y bueno la implementación se la realiza de la siguiente manera.

Se crea una clase ***Cliente.java*** únicamente con sus constructores, getters y setters.

Clase ClienteVO patrón DTO (Data transfer object)

```java

public class ClienteVO {
    private int id;
    private String nombre;
    private String apellido;


    public ClienteVO(int id, String nombre, String apellido) {
        this.id = id;
        this.nombre = nombre;
        this.apellido = apellido;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

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

    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    @Override
    public String toString() {
        return this.getNombre()+" "+this.getApellido();
    }
}
```

Se crea el acceso a los datos a través de una interface **IClienteDao.java**, aquí se declara todos los métodos para acceder a los datos.

Interface IClienteDao patrón DAO

```java

import java.util.List;

import com.ecodeup.model.Cliente;

public interface IClienteDao {
    //declaración de métodos para acceder a la base de datos
    public List<Cliente> obtenerClientes();
    public Cliente obtenerCliente(int id);
    public void actualizarCliente(Cliente cliente);
    public void eliminarCliente(Cliente cliente);
}
```

Se implementa en la clase **ClienteDaoImpl.java** haciendo un *implements* de la interface ***IClienteDao.java,*** lo que se hace aquí, no es más que implementar cada método de la interface.

```java
public class ClienteDaoImpl implements IClienteDao {

    //lista de tipo cliente
    List<Cliente> clientes;

    //inicializar los objetos cliente y añadirlos a la lista
    public ClienteDaoImpl() {
        clientes = new ArrayList<>();
        Cliente cliente1 = new Cliente(,"Javier", "Molina");
        Cliente cliente2 = new Cliente(1,"Lillian","Álvarez");
        clientes.add(cliente1);
        clientes.add(cliente2);
    }

    //obtener todos los clientes
    @Override
    public List<Cliente> obtenerClientes() {
        return clientes;
    }

    //obtener un cliente por el id
    @Override
    public Cliente obtenerCliente(int id) {
        return clientes.get(id);
    }

    //actualizar un cliente
    @Override
    public void actualizarCliente(Cliente cliente) {
        clientes.get(cliente.getId()).setNombre(cliente.getNombre());
        clientes.get(cliente.getId()).setApellido(cliente.getApellido());
        System.out.println("Cliente con id: "+cliente.getId()+" actualizado satisfactoriamente")
    }

    //eliminar un cliente por el id
    @Override
    public void eliminarCliente(Cliente cliente) {
        clientes.remove(cliente.getId());
        System.out.println("Cliente con id: "+cliente.getId()+" elimnado satisfactoriamente");
    }
```

gPor último se prueba el patrón DAO a través de la clase DaoDemo.java

```java
package com.ecodeup.daodemo;

import com.ecodeup.dao.ClienteDaoImpl;
import com.ecodeup.idao.IClienteDao;
import com.ecodeup.model.Cliente;

public class DaoDemo {

    public static void main(String[] args) {
        // objeto para manipular el dao
        IClienteDao clienteDao = new ClienteDaoImpl();

        // imprimir los clientes
        clienteDao.obtenerClientes().forEach(System.out::println);

        // obtner un cliente
        Cliente cliente = clienteDao.obtenerCliente();
        cliente.setApellido("Pardo");
        //actualizar cliente
        clienteDao.actualizarCliente(cliente);

        // imprimir los clientes
        System.out.println("*****");
        clienteDao.obtenerClientes().forEach(System.out::println);
    }
}
```