

MEMORIA- PARTE DAO

(SEGUNDA ENTREGA)

INTEGRANTES DEL GRUPO:

ÁLVARO DEL CAMPO
JUAN JOSÉ ENCINA FERNÁNDEZ
JUAN EMBID
CARLOS FORRIOL
JUAN MONTERO GÓMEZ
VÍCTOR ORTEGO
SAMUEL PARRA
ALBERTO RAMOS SUÁREZ
JUAN ROMO IRIBARREN
IGNACIO REDONDO
ANTONIO TRENADO
FAN YE

Índice

1.	Componentes del proyecto.....	2
2.	Patrones aplicados en el proyecto.....	3
3.	Link al prototipo.....	6

1. Componentes del proyecto

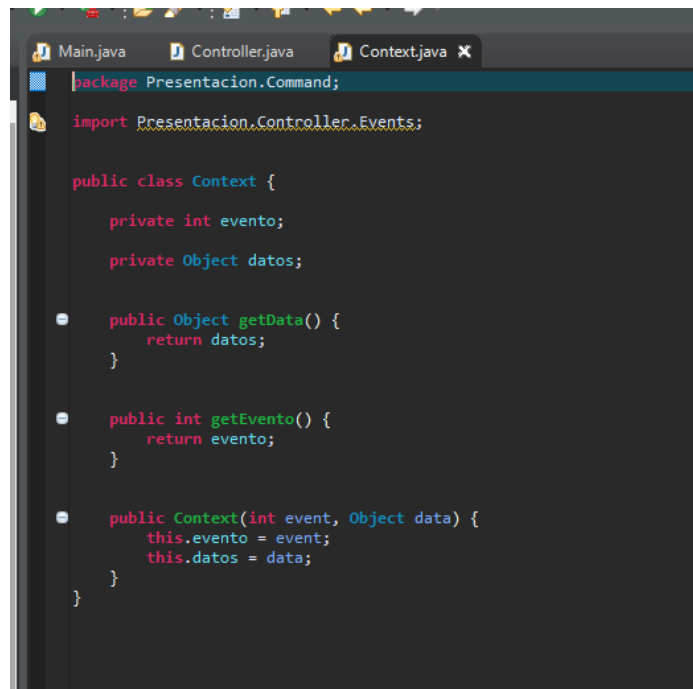
Nuestro proyecto consiste en la gestión de un restaurante, siendo una aplicación Java de escritorio, consistente en una primera versión tipo DAO, con arquitectura multicapa y con el uso de patrones, que gestiona transacciones y concurrencia. Se ha utilizado el control de versiones SVN de la Facultad de Informática.

El proyecto se compone de las siguientes partes:

- **Modelo del diseño:** En el cuál se recogen todos los diagramas de secuencia y de clases que proporcionan un diseño de la arquitectura. Hemos dividido en las 3 capas (integración, negocio y diseño) y a su vez por entidad donde se encuentra cada diagrama de clase y los de secuencia. Se ha realizado mediante UML x2 en las herramientas CASE que nos proporciona IBM RSAD.
- **Código:** Aquí se recogen todas las clases creadas a través de una transformación de diseño a código que IBM RSAD nos permite realizar. Este está dividido en:
 - **Carpeta src:** Donde se encuentran todas las clases Java alojadas en una estructura multicapa de integración, negocio y presentación y por entidades.
 - **Carpeta test:** Donde se encuentran los JUnits, que son las pruebas implementadas para verificar la corrección del código
- **Base de datos:** Para la persistencia de datos relacional se ha utilizado MySQL 5x, mediante los servidores de Microsoft Azure MYSQL, ya que la Universidad nos proporciona un acceso gratuito y como entorno hemos utilizado el MYSQL WorkBench para crear las tablas.

2. Patrones obligatorios aplicados en el proyecto

- **Service to worker:** Se utiliza para poder gestionar la lógica de negocio y según el contexto que nos proporcione generar una vista o llamar a un comando. El comando invoca al negocio, que una vez realizada una operación, se ejecuta la vista que se necesite. El contexto encapsula un comando a través del evento que reciba (o una vista si datos es nulo).



```
package Presentacion.Command;

import Presentacion.Controller.Events;

public class Context {

    private int evento;

    private Object datos;

    public Object getData() {
        return datos;
    }

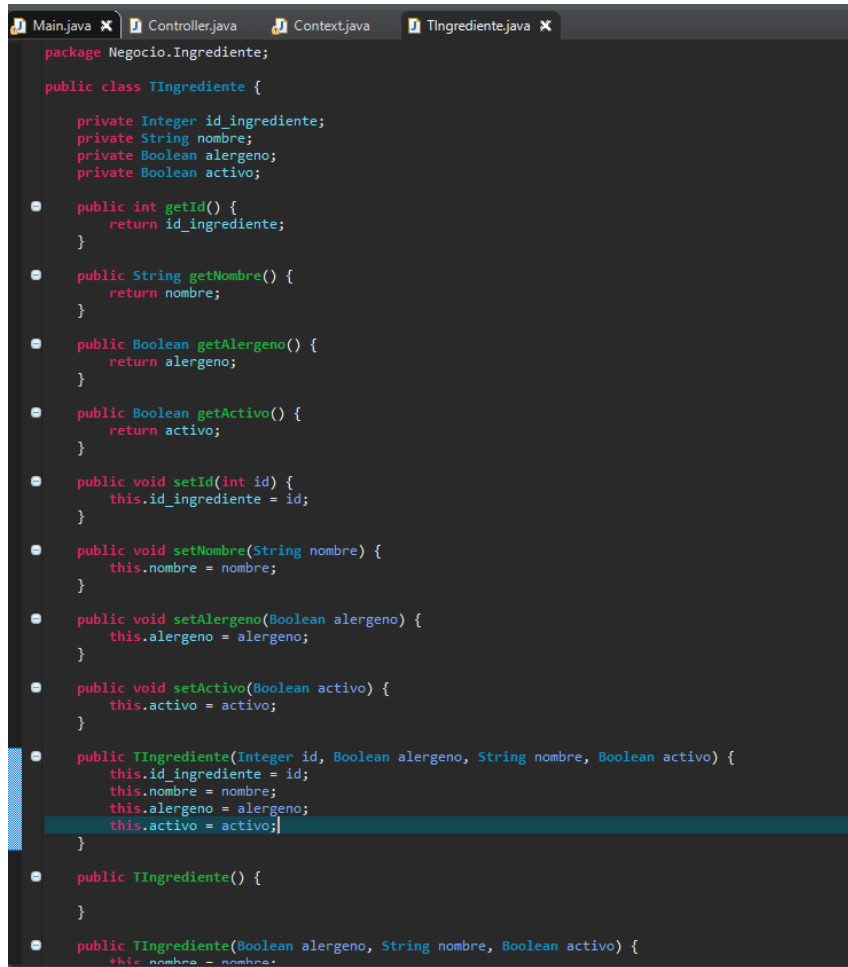
    public int getEvento() {
        return evento;
    }

    public Context(int event, Object data) {
        this.evento = event;
        this.datos = data;
    }

}
```

Clase Context

- **Transfer object:** Usado para intercambiar datos entre las capas de la arquitectura, siendo independiente del modelo de datos.



```

package Negocio.Ingrediente;

public class TIngrediente {

    private Integer id_ingrediente;
    private String nombre;
    private Boolean alergeno;
    private Boolean activo;

    public int getId() {
        return id_ingrediente;
    }

    public String getNombre() {
        return nombre;
    }

    public Boolean getAlergeno() {
        return alergeno;
    }

    public Boolean getActivo() {
        return activo;
    }

    public void setId(int id) {
        this.id_ingrediente = id;
    }

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

    public void setAlergeno(Boolean alergeno) {
        this.alergeno = alergeno;
    }

    public void setActivo(Boolean activo) {
        this.activo = activo;
    }

    public TIngrediente(Integer id, Boolean alergeno, String nombre, Boolean activo) {
        this.id_ingrediente = id;
        this.nombre = nombre;
        this.alergeno = alergeno;
        this.activo = activo;
    }

    public TIngrediente() {
    }

    public TIngrediente(Boolean alergeno, String nombre, Boolean activo) {
        this.nombre = nombre;
    }
}

```

Clase TIngrediente

- **Transfer Object Assembler:** Se aplican un conjunto de objetos transfer encapsulados para procesarlos. Lo hemos utilizado para mostrar una factura, ya que esta operación encapsula un TMesa, un TEmpleado, una lista de TLineaDePedido y una lista de TMenu.

```

public TFacturaCompleta mostrarUno(Integer id) {
    TransactionManager tManager = TransactionManager.getInstance();
    Transaction transaction = tManager.newTransaction();
    transaction.start();
    DAOFactura daoFactura = FactoriaIntegracion.getInstance().generarDAOFactura();

    TFactura tFactura = daoFactura.leerUno(id);

    if (tFactura == null) {
        transaction.rollback();
        return null;
    }
    TMesa mesa = FactoriaIntegracion.getInstance().generarDAO Mesa().leerPorId(tFactura.getIdMesa());
    TEmpleado emp = FactoriaIntegracion.getInstance().generarDAOEmpleado().leerUno(tFactura.getIdEmpleado());
    ArrayList<TLineaDePedido> lineas = (ArrayList<TLineaDePedido>) FactoriaIntegracion.getInstance().generarDAOFacturaMenu().leerMenusEnFactura(tFactura.getId());
    ArrayList<TMenu> menus = new ArrayList<TMenu>();
    DAOMenu daoMenu = FactoriaIntegracion.getInstance().generarDAOMenu();
    TMenu menu;
    for(TLineaDePedido l : lineas){
        menu = daoMenu.leerUno(l.getIdMenu());
        if(menu != null){
            menus.add(menu);
        }
    }
    TFacturaCompleta res = new TFacturaCompleta(tFactura, mesa, emp, menus, lineas);
    transaction.commit();

    return res;
}

```

Ejemplo de TOA

- **Application Service:** Alberga toda la lógica de negocio a través de distintos componentes de la capa de negocio y servicios.

```

SAIngredienteImp.java  SAIngrediente.java
package Negocio.Ingrediente;

import java.util.Collection;

public interface SAIngrediente {

    public int crear(TIngrediente tIng);

    public int editar(TIngrediente tIngr);

    public int eliminar(Integer id);

    public TIngrediente mostrarUno(Integer idIngr);

    public Collection<TIngrediente> mostrarTodos();

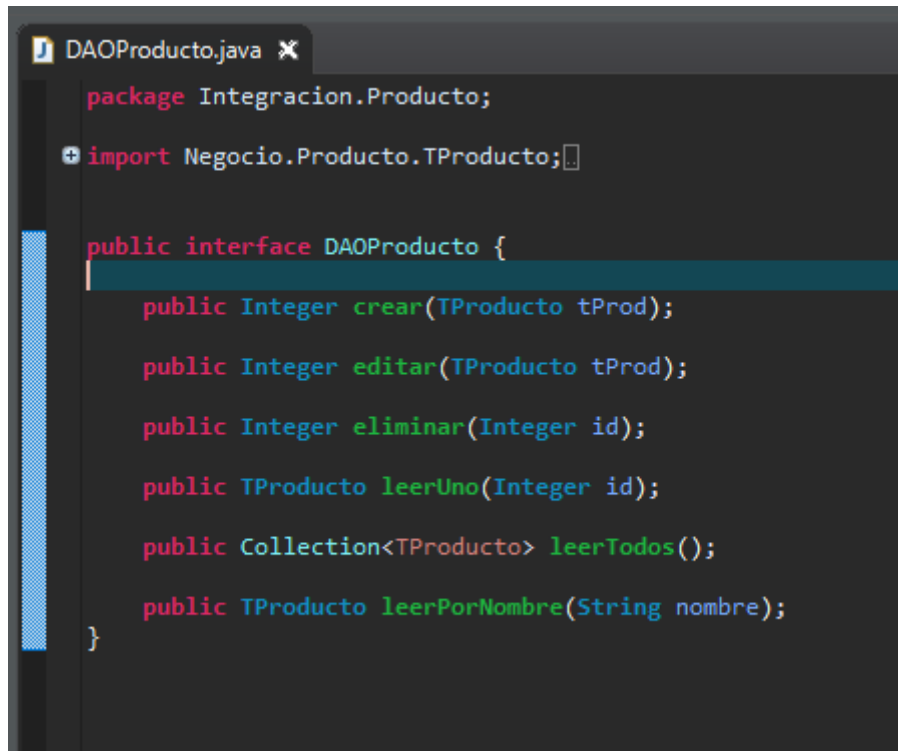
    public TIngrediente mostrarUnoPorNombre(String nombre);

}

```

Servicio de aplicación de ingrediente

- **Data Access Object:** Se usa para el acceso a la capa de datos (SGBDR), manejando las peticiones de la capa de presentación. Están alojados en la capa de integración



```
DAOProducto.java X
package Integracion.Producto;

+ import Negocio.Producto.TProducto;

public interface DAOProducto {

    public Integer crear(TProducto tProd);

    public Integer editar(TProducto tProd);

    public Integer eliminar(Integer id);

    public TProducto leerUno(Integer id);

    public Collection<TProducto> leerTodos();

    public TProducto leerPorNombre(String nombre);

}
```

DAOProducto

Destacar que se han aplicado otros patrones no obligatorios como el Singleton en las factorías o el Abstract Factory. Además se han hecho uso de queries tal y como hemos visto en clase, como por ejemplo en menú mostrar Menús por rango de precio.

3. Link al prototipo

- Documentación: <https://versiones.fdi.ucm.es/svn/MS/2223E/kernelpanic/doc>
- Modelo: <https://versiones.fdi.ucm.es/svn/MS/2223E/kernelpanic/mod>
- Código: <https://versiones.fdi.ucm.es/svn/MS/2223E/kernelpanic/cod>