

# MEMORIA PROYECTO SUPERMERCADO- RESTAURANTE

Modelado de Software

RESTAURANTE



Supermercado

## INTEGRANTES DEL GRUPO:

LUIS ENRIQUE BARRERO PEÑA  
JOSE LUIS BARTHA DE LAS PEÑAS  
BRYAN EDUARDO CÓRDOVA ASCURRA  
CARLOS FORRIOL MOLINA  
PABLO GAMO GONZÁLEZ  
CARLOS GÓMEZ LÓPEZ  
MARIO GONZÁLEZ DE SANTOS  
JAVIER DE HOYOS PINO  
RUBÉN MARTÍN CASTRO  
ALBERTO ALEJANDRO RIVAS FERNÁNDEZ  
JUAN ROMO IRIBARREN  
GABRIEL TORRES MARTIN

## Estructura del proyecto

Para este proyecto, hemos optado por un modelo de proceso evolutivo, gracias a las múltiples iteraciones que este modelo propone, facilitándonos la creación de un software incremental, sólido y estable. Concretamente nos hemos decidido por el modelo del Proceso unificado de desarrollo puesto que gracias a las cinco tareas que dividen cada iteración, podemos desarrollar cada módulo dedicándole el tiempo necesario a cada una de dichas actividades.

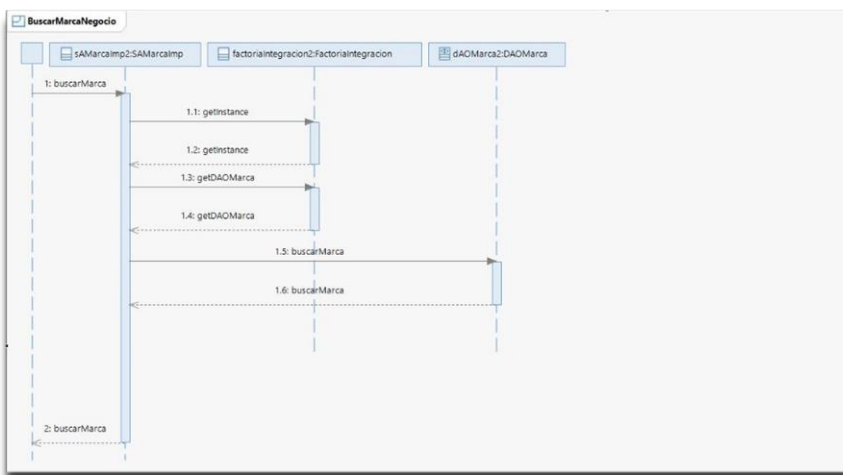
Estos flujos de trabajo son: requisitos, análisis, diseño, implementación, pruebas. A través de estas actividades hemos desarrollado tres documentos:

- **Modelo de requisitos.** En este apartado incluimos diagramas que representan el funcionamiento de nuestro proyecto de una forma abstracta, mostrando los requisitos funcionales y no funcionales del proyecto. En este paquete incluimos:
  - **Modelo del dominio.** Diagrama relacional que ilustra de forma sencilla el comportamiento y relaciones de los módulos que componen nuestro proyecto.
  - **Paquetes de cada Módulo.** Dentro de estos subpaquetes hemos realizado múltiples diagramas. En cada paquete podemos encontrar los diagramas de caso de uso que ayudan a capturar los requisitos funcionales y las acciones que realiza cada módulo. Además, dentro de cada paquete también contamos con los diagramas de actividades que describen el funcionamiento de cada caso de uso.
- **Modelo del diseño.** Paquete que se encarga de mostrar de forma gráfica el comportamiento real de cada caso de uso mediante diagramas de secuencia y diagramas de clases. Este paquete se divide en:
  - **Diagramas de Secuencia.** Conjunto de diagramas que describen las cadenas de llamadas y comunicación entre componentes para cada caso de uso general. Dentro de los diagramas de la parte del supermercado:
    - Alta Marca.
    - Buscar una Marca por su ID.
    - Buscar Todas las Marcas.
    - Modificar Marca.
    - Eliminar Marca.
    - Añadir Producto al carrito de la Compra.
    - Eliminar Producto del carrito de la Compra.
    - Mostrar ayuda en la Vista Principal.
    - Pagar Compra.
    - Buscar un Producto por Marca.
    - Vincular un Producto a un Proveedor.
    - Vincular un Producto de un Proveedor.
    - Identificar Trabajador en la Vista Principal.
    - Realizar Devolución de una Compra.
    - Commit, GetResource, Rollback, Start, eliminar, crear y obtener la transacción

Por parte del restaurante:

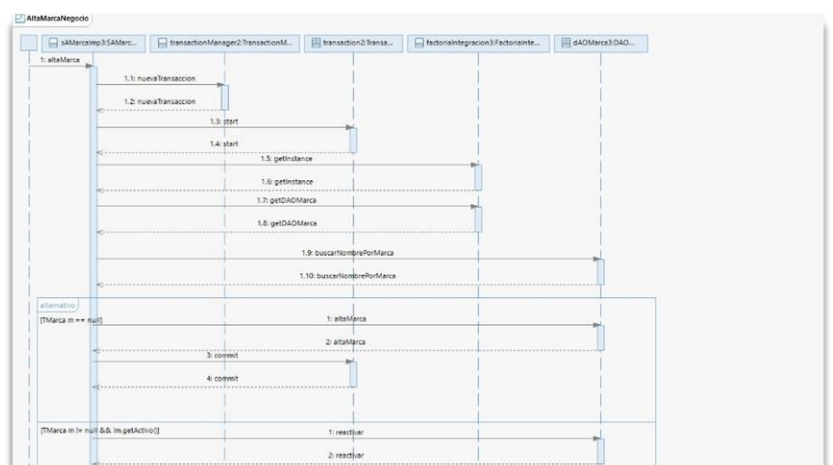
- Dar de Alta un Cliente
- Dar de Baja un Cliente
- Modificar un Cliente
- Buscar un Cliente por su ID
- Buscar Todos los Clientes
- Identificar Personal del restaurante
- Calcular la nómina de un Turno
- Dar de baja un Turno
- Añadir ingredientes a platos
- Mostrar los ingredientes de un plato
- Cerrar un Pedido
- Realizar devoluciones de platos de ese Pedido

Dentro de la parte de Supermercado implementados con DAOS:



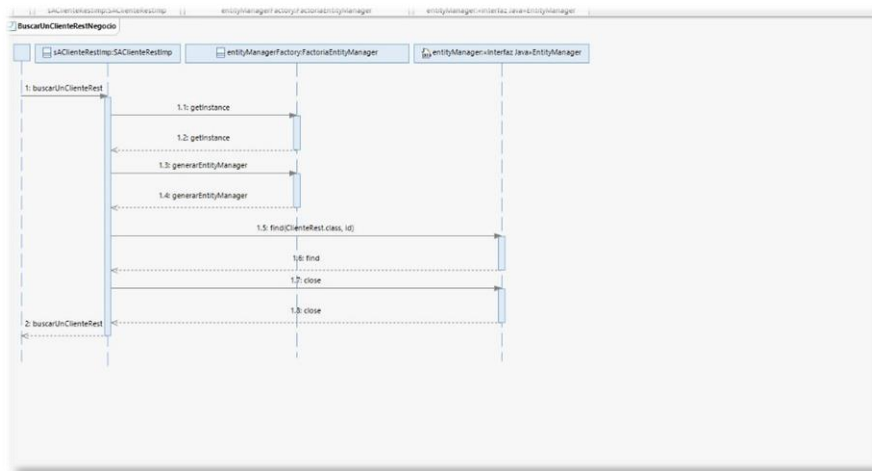
Ejemplo de diagrama de secuencia para el caso de uso de buscar una Marca por ID.

Fragmento del diagrama de secuencia del alta de una marca.



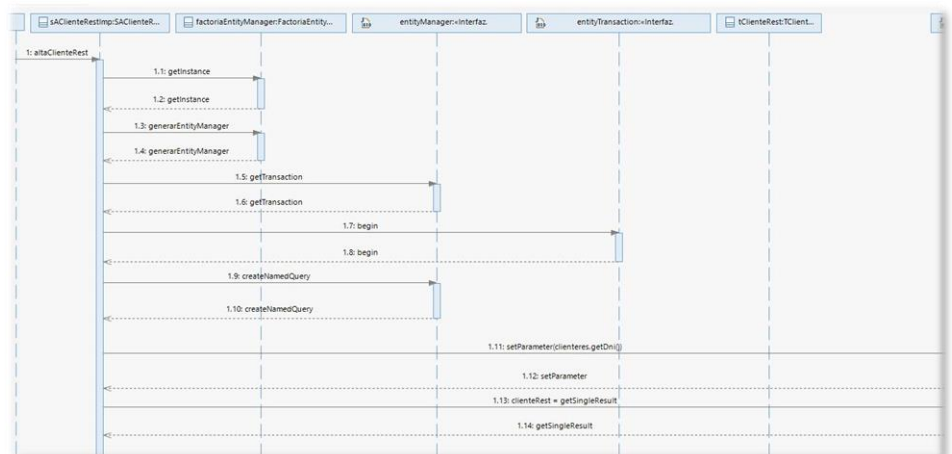
El funcionamiento del resto de casos de uso, son similares a los previamente descritos y por consiguiente pueden asemejarse a estos mismos.

En cambio, la parte del restaurante hemos tenido que implementarla con JPA, la API de persistencia creada por la plataforma de Java EE.



Ejemplo de diagrama de secuencia para el caso de uso de buscar un Cliente por su ID.

Fragmento del diagrama de secuencia del alta de un cliente



- **Diagramas de clases.** Estos diagramas representan aquellos elementos de los que hace uso clase perteneciente a cada módulo. Estos diagramas se estructuran a su vez en tres capas:
  - **Presentación.** Abarca todas las clases que componen el apartado visual e interfaces gráficas. Este paquete se encarga de la lógica que permite al usuario interactuar con el sistema.
  - **Negocio.** Lleva a cabo las operaciones lógicas para transmitir de forma consistente los datos entre la capa de presentación y la capa de integración.
  - **Integración.** Encargada de comunicarse y transmitir o recibir la información externa al sistema (base de datos) así como proporcionarla a la capa de negocio.



- **Codificación.** Una vez finalizada la etapa del diseño comenzamos con esta fase, la más compleja y larga, donde desarrollaremos nuestra aplicación. Fuimos implementando el proyecto siguiendo la estructura explicada anteriormente donde nos centramos en:
  - Definimos una organización del código.
  - Implementamos las clases de capa, vistas en la estructura.
  - Creamos una base de datos para nuestro proyecto y la conectamos con la aplicación.
  - Se crearon las tablas de la base de datos.
  - Se probó individualmente cada componente y se integró al sistema.
  - Por último, se realizaron pruebas donde se verifica si se realizó correctamente nuestra implementación. Las pruebas son un mecanismo para revisar los modelos, se pedirá una consistencia entre el modelo de análisis y modelo de diseño. Las pruebas son de:
    1. **Unidad:** Unidades de programa pequeñas en nuestro caso la clase con sus respectivas operaciones.
    2. **Integración:** Se prueban estas unidades en conjunto mediante pruebas basadas en uso o en hilos.
    3. **Validación del sistema:** Se prueba el sistema mediante pruebas alfa (llevadas a cabo por usuarios finales en lugar del desarrollador) o beta (sin el desarrollador).

Hemos llevado a cabo una automatización de pruebas mediante la herramienta Junit 4 que nos sirve para comparar objetos obtenidos con objetos esperados. Para ello definimos nuevas clases “test” replicadas a nuestra distribución de paquetes en el código con anotaciones que hacen referencia al marco especificado. Una diferencia del Junit 4 con el 5 es que los métodos deben de ser públicos.



Usamos los siguiente métodos para implementar nuestras pruebas:

1. **@FixMethodorder:** Que nos ayuda a elegir el orden de la ejecución de los métodos es nuestra clase de prueba. Esto lo hacemos puesto que en algunas versiones de java el orden de la ejecución no está garantizada. En nuestro caso lo hacemos en orden alfabético.

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class DAOSeccionTests {
    static DAOSeccionimp daoSeccion;
    static int id;
```

2. **@BeforeClass:** Que nos ayuda con la independencia de los test comunes cuando tenemos que ejecutar muchos a la vez, normalmente son operaciones costosas para la base de datos.

```
@BeforeClass
public static void beforeClass(){
    daoSeccion = new DAOSeccionimp();
}
```

3. **@Test:** Que denota un test.

```
@test
public void bCrearSeccionYaExistente(){
    TSeccion seccion = new TSeccion();
    seccion.setZona("Carniceria");
    seccion.setPasillo(1);
    int res = daoSeccion.crearSeccion(seccion);

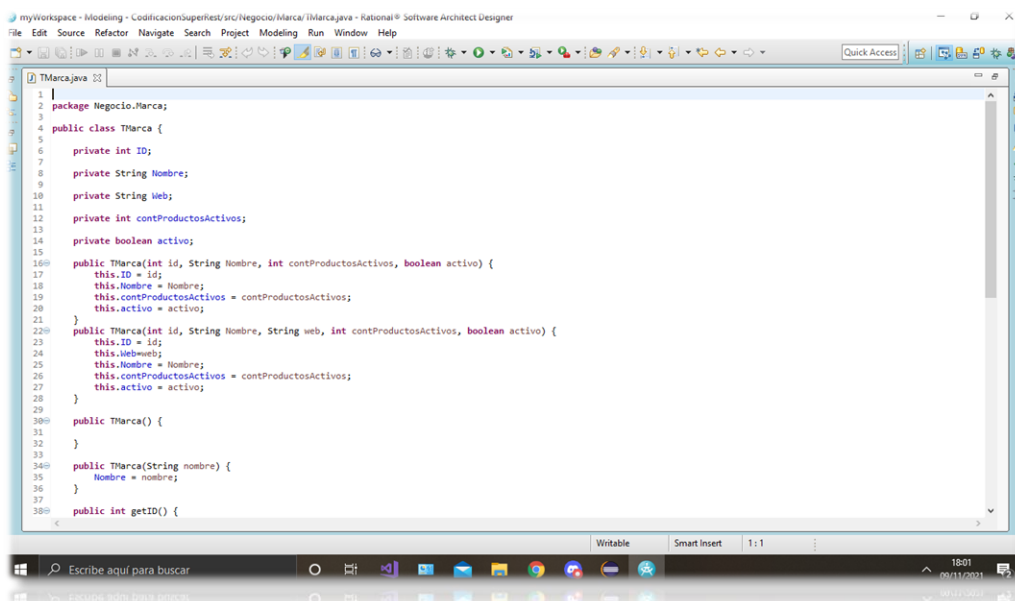
    assertEquals(-1,res);
}
```

## Patrones empleados en el proyecto

1. **Patrón Singleton.** El cual garantiza que sólo hay una instancia de una clase garantizada por ella misma, las utilizamos en factorías, distintas clases de presentación y en el controlador.

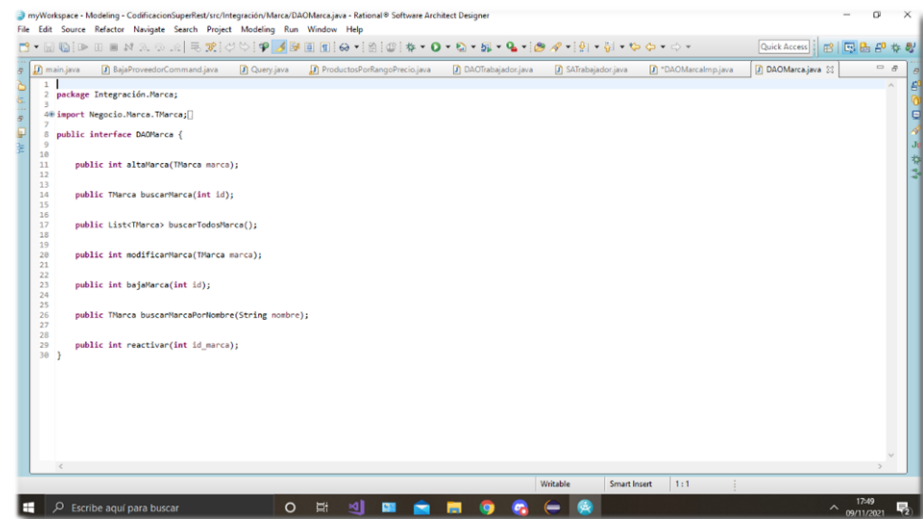
```
public abstract class FactoriaIntegracion {  
    private static FactoriaIntegracion instancia;  
  
    public static FactoriaIntegracion obtenerInstancia() {  
        if (instancia == null)  
            instancia = new FactoriaIntegracionImp();  
  
        return instancia;  
    }  
  
    public abstract DAOCliente generaDAOClientes();  
    public abstract DAOCompra generarDAOCompras();  
    public abstract DAOMarcas generarDAOMarcas();  
    public abstract DAOProductos generarDAOProductos();  
    public abstract DAOProveedores generarDAOProveedores();  
    public abstract DAOSecciones generarDAOSecciones();  
    public abstract DAOTrabajadores generarDAOTrabajadores();  
}
```

2. **Patrón transferencia.** El cual independiza el intercambio de datos entre capas y evita que una capa tenga que conocer la representación de las entidades, por ejemplo, un cliente no debe de conocer todos los datos de la base de datos como el id de un trabajador. En nuestro proyecto lo utilizamos en las capas de presentación, negocio e integración.

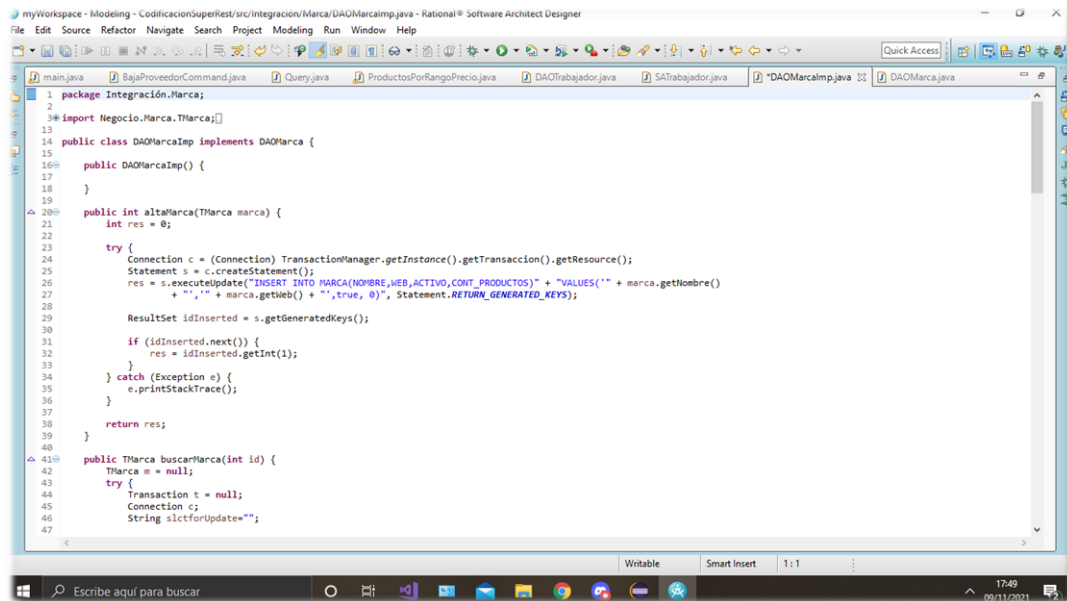


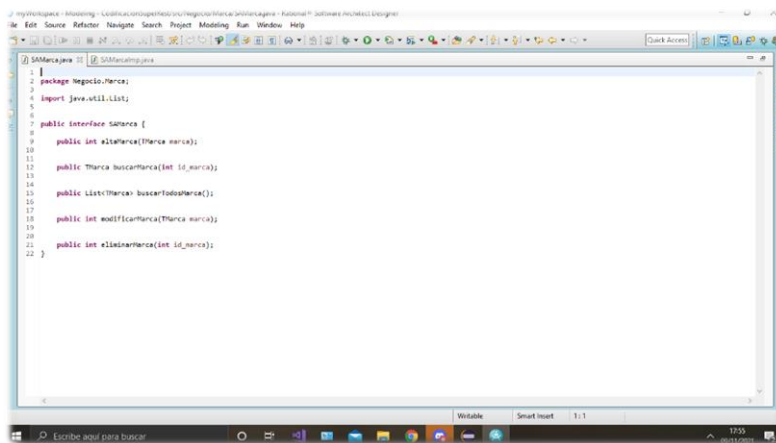


**3. Patrón DAO (Data Access Object).** El cual es utilizado para acceder a la de base de datos y proporciona acceso para el manejo de las peticiones de la capa de presentación, lo utilizamos en la capa de integración.



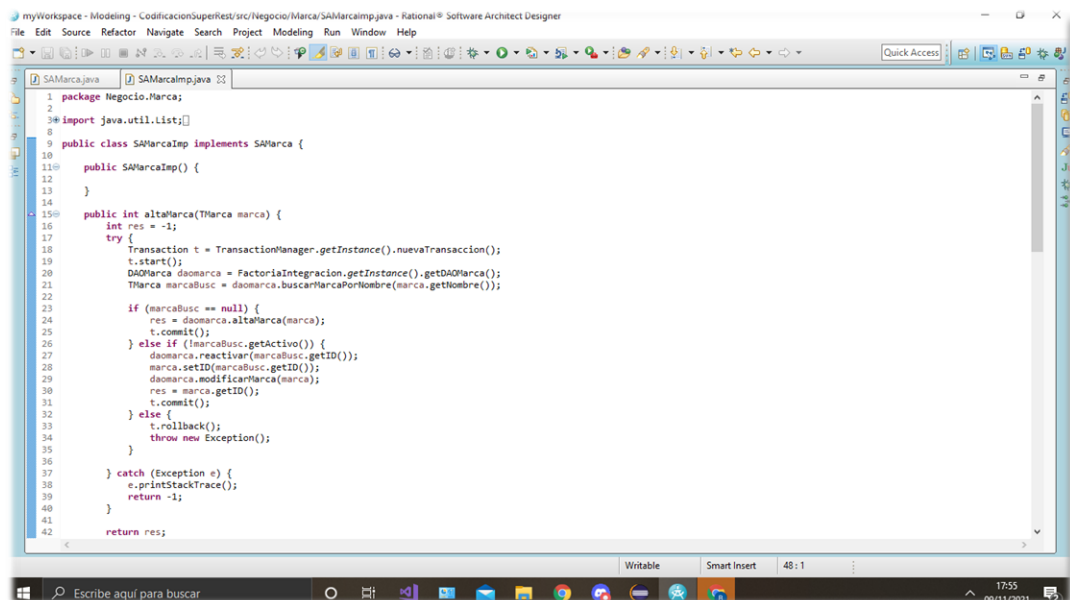
## Implementación del Patrón DAO



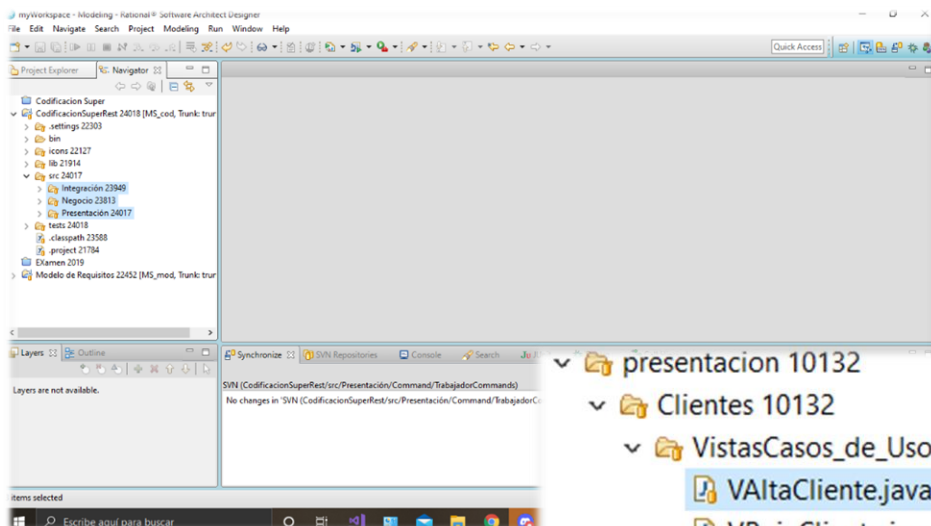


**4. Servicio de Aplicación.** El cual se centraliza en lógica de negocio que la aplicamos a objetos de negocio. La utilizamos en negocio.

## Implementación del patrón Servicio de Aplicación

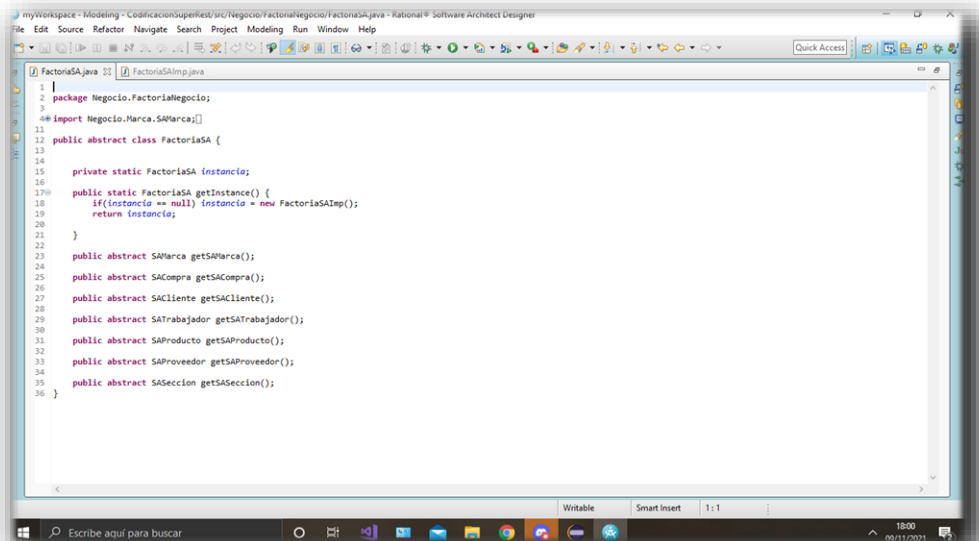


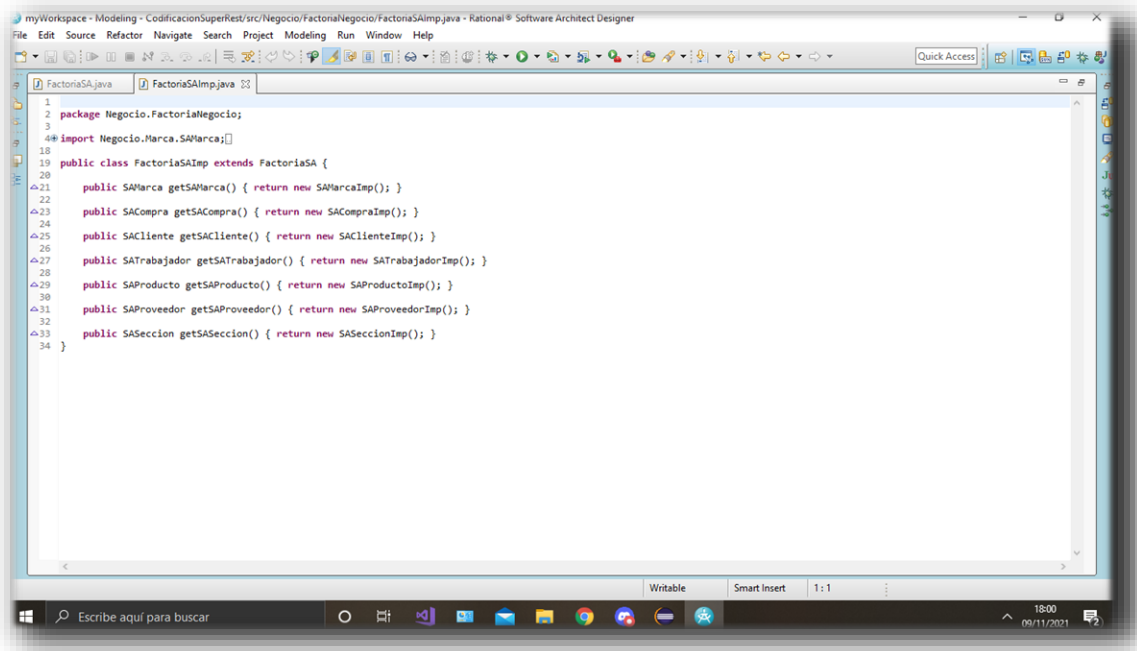
**5. MVC (Modelo Vista controlador).** El cual centraliza y modulariza la gestión de acciones y vistas, Vista->Controlador->Modelo. Te permite tener múltiples vistas para un mismo modelo



- presentacion 10132
  - Cientes 10132
    - VistasCasos\_de\_Uso 10132
      - ValtaCliente.java 10132
      - VBajaCliente.java 10132
      - VBuscarCliente.java 10132
      - VModificarCliente.java 10132
    - Evento.java 10132
    - IGUI.java 10132
    - PruebaVista.java 9995
    - VCliente.java 10132

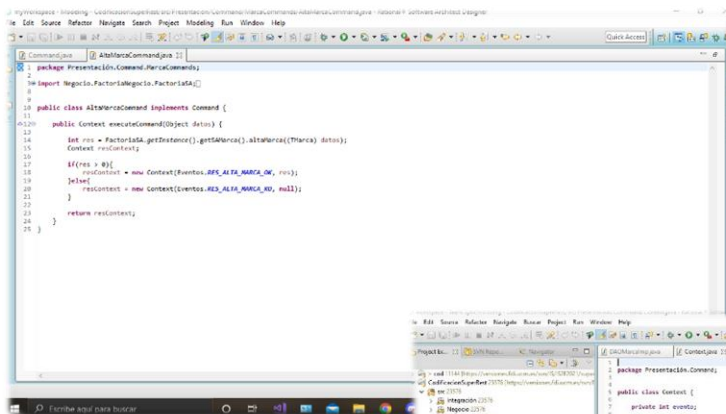
**6. Factoría abstracta.** El cual es utilizado por el DAO y SA y genera familias de objetos relacionados. Crea objetos, pero devuelve interfaces.



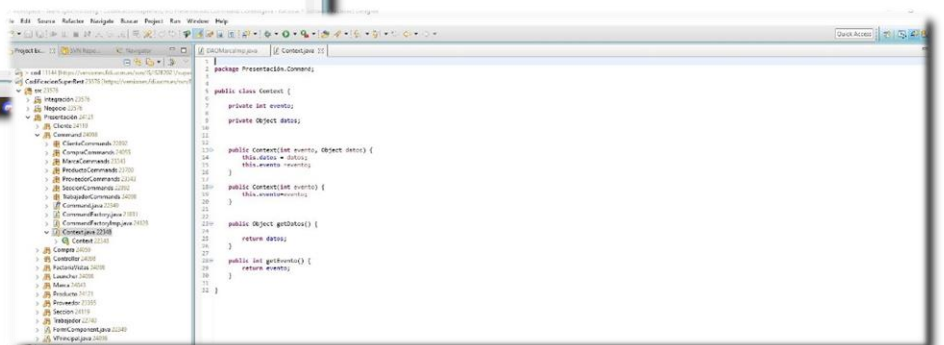


**7. Service To Worker.** Lo implementamos en nuestro proyecto para poder ejecutar lógica de negocio concreta y a partir de la respuesta obtenida generar/actualizar una vista concreta. Para ello nos apoyamos en dos elementos:

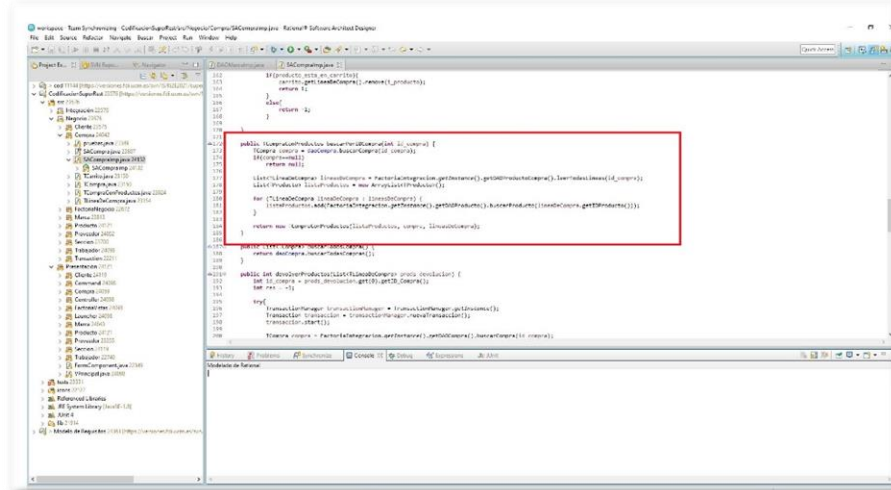
- Comando.** Se encarga de llamar a la lógica de negocio y tras recibir la respuesta, mandar ejecutar una u otra vista.
- Contexto.** Encapsula un evento y los datos que la vista proporciona al comando y que el comando devuelve a la vista que se generará como respuesta.



**Ejemplos del Comando y Contexto en la entidad**



- 8. Patrón TOA.** Patrón que proporciona un conjunto de objetos de transferencia (Transfers) encapsulados con el objetivo de mostrarlos o procesarlos. Por ejemplo, en nuestro proyecto utilizamos este patrón para mostrar la lista de productos (con todos sus atributos) dentro de una compra, a partir de la clase TCompraConProductos, o también TPlatoConIngredientes, el cual nos sirve para mostrar los ingredientes de un plato. Esto nos proporciona toda la información encapsulada para poder manipularla sin tener que llamar a los SA de ambas módulos entorpeciendo el proceso.



TOA de Compra con sus Productos

```
package Negocio.Plato;

import java.util.List;

public class TPlatoConIngredientes {

    private TPlato plato;
    private List<Ingrediente> ingredientes;

    public TPlatoConIngredientes(TPlato plato, List<Ingrediente> ingredientes) {
        this.plato = plato;
        this.ingredientes = ingredientes;
    }

    public List<Ingrediente> getIngredientes() {
        return ingredientes;
    }

    public TPlato getPlato() {
        return plato;
    }

    public void setIngrediente(List<Ingrediente> ingredientes) {
        this.ingredientes = ingredientes;
    }

    public void setPlato(TPlato plato) {
        this.plato = plato;
    }
}
```

TOA de Platos con sus Ingredientes

## 9. Almacén del dominio y objetos de negocio

JPA es la herramienta que nos otorga Java EE para implementar el almacén del dominio y los objetos de negocio.

Los objetos de negocio se interpretan a través de las entidades, las cuales tienen metadatos asociados, que describen su relación entre la representación y el almacén persistente. JPA automatizará estos metadatos y tendrá todo configurado.

```
package negocio.persistencia;

import jakarta.persistence.Column;

@Entity
@NamedQuery({
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where id = obj.id"),
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where pedido NUMBER(9) obj.pedido"),
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where nombre = obj.nombre"),
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where apellidos = obj.apellidos"),
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where dni = obj.dni"),
    @NamedQuery(name = "Negocio.ClienteRest.ClienteRest.findById", query = "select obj from ClienteRest obj where activo = obj.activo")
})
public class ClienteRest implements Serializable {
    private static final long serialVersionUID = 0;

    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    private Integer id;

    @OneToMany(mappedBy = "clienteRest")
    private List<Pedido> pedido;

    private String nombre;
    private String apellidos;

    @Column(unique = true)
    private String dni;

    private boolean activo;

    @Version
    private Long version;

    public ClienteRest() {
    }

    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;
    }
}
```

### Implementación de Objetos de Negocio (Entidades)

Para la gestión de las entidades JPA utiliza el entity manager, que es el elemento que mantendrá la persistencia de la base de datos.

Este debe tener una configuración que determinara las características de las entidades gestionadas por el entity manager.

Esta configuración se conoce como la unidad de persistencia, donde se describe el nombre de la configuración, que entidades se podrán gestionar, el sistema de cacheo, el tipo de gestión transaccional, la conexión a la base de datos donde guardaremos nuestros datos, la manera de crear las tablas de dicha base de datos.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <persistence xmlns="https://jakarta.ee/xml/ns/persistence"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
5     version="3.0">
6     <persistence-unit name="CodificacionSuperRest" transaction-type="RESOURCE_LOCAL">
7         <!-- TODAS LAS CLASES QUE UTILIZAMOS -->
8         <class>Negocio.ClienteRest.ClienteRest</class>
9         <class>Negocio.Plato.Plato</class>
10        <class>Negocio.Plato.Bebida</class>
11        <class>Negocio.Plato.Comida</class>
12        <class>Negocio.Personal.Personal</class>
13        <class>Negocio.Personal.TiempoCompletoP</class>
14        <class>Negocio.Personal.TiempoParcialP</class>
15        <class>Negocio.Pedido.Pedido</class>
16        <class>Negocio.Pedido.LineaPedido</class>
17        <class>Negocio.Ingrediente.Ingrediente</class>
18        <class>Negocio.Turno.Turno</class>
19        <shared-cache-mode>NONE</shared-cache-mode>
20        <validation-mode>NONE</validation-mode>
21
22        <!-- properties, INCLUIROS CONEXION, Y COMO QUEREMOS CREAR LAS TABLAS -->
23        <properties>
24            <property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
25            <property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://35.233.32.45:3306/BORest"/>
26            <property name="jakarta.persistence.jdbc.user" value="JBUser"/>
27            <property name="jakarta.persistence.jdbc.password" value="JavaJava99"/>
28
29            <!-- enable this EclipseLink property to see SQL and other logging -->
30            <!-- property name="eclipseLink.logging.level" value="FINE" -->
31
32            <property name="eclipseLink.ddl-generation" value="create-or-extend-tables"/>
33            <property name="eclipseLink.ddl-generation.output-mode" value="database"/>
34            <property name="eclipseLink.ddl-generation-mode" value="ddl_database_generation" />
35        </properties>
36    </persistence-unit>
37 </persistence>
38
39
40 </beans>
41 </beans>
42 </beans>
```

### Unidad de persistencia del proyecto

## **Ponderación de los miembros del equipo**

El factor de corrección lo pondremos sobre 1:

LUIS ENRIQUE BARRERO PEÑA = 1

JOSE LUIS BARTHA DE LAS PEÑAS = 1

BRYAN EDUARDO CÓRDOVA ASCURRA = 1

CARLOS FORRIOL MOLINA = 1

PABLO GAMO GONZÁLEZ = 1

CARLOS GÓMEZ LÓPEZ = 1

MARIO GONZÁLEZ DE SANTOS = 1

JAVIER DE HOYOS PINO = 1

RUBÉN MARTÍN CASTRO = 1

ALBERTO ALEJANDRO RIVAS FERNÁNDEZ = 1

JUAN ROMO IRIBARREN = 1

GABRIEL TORRES MARTIN = 1

## **Herramienta de control de versiones**

Para este proyecto, hemos empleado como sistema de control de versiones los repositorios de la facultad (SVN):

### **- Documentación:**

<https://versiones.fdi.ucm.es/svn/MS/2122E/supermercado/doc>

### **- Modelo:**

<https://versiones.fdi.ucm.es/svn/MS/2122E/supermercado/mod>

### **- Código:**

<https://versiones.fdi.ucm.es/svn/MS/2122E/supermercado/cod>