# MEMORIA PROYECTO SUPERMERCADO-RESTAURANTE

## Modelado de Software


**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

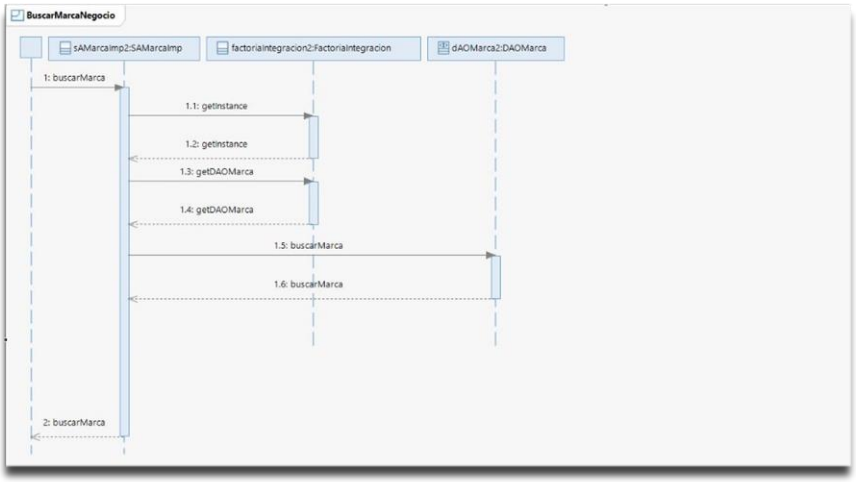
Para este proyecto, hemos optado por un modelo de proceso evolutivo, gracias a las múltiples iteraciones que este modelo propone, facilitándonos la creación de un software incremental, sólido y estable. Concretamente nos hemos decido por el modelo del Proceso unificado de desarrollo puesto que gracias 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 acaso de uso.

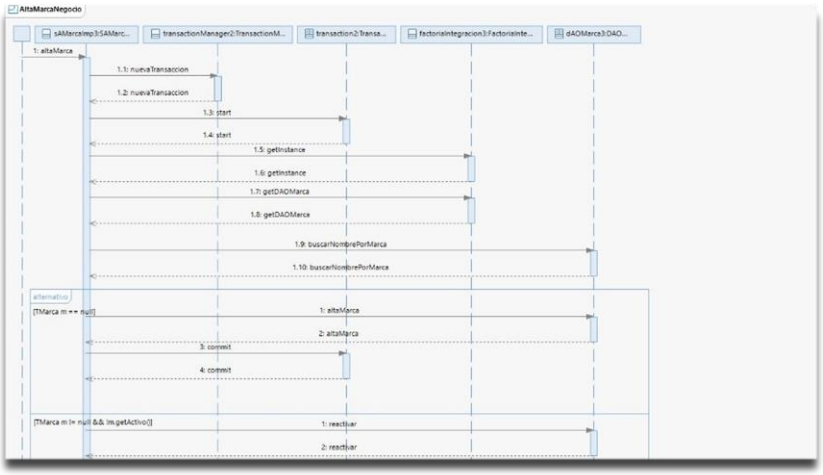
- **Modelo del diseño**. Paquete que se encarga de mostrar de forma gráfica el comportamiento real de cada caso de uso mediante diagramas de secuencia y diagramas de clases. Este paquete se divide en:

    - **Diagramas de Secuencia**. Conjunto de diagramas que describen las cadenas de llamadas y comunicación entre componentes para cada caso de uso general. Estos diagramas son:
        - 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




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.

- **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 trasmitir o recibir la información externa al sistema (base de datos) así como proporcionarla a la capa de negocio.

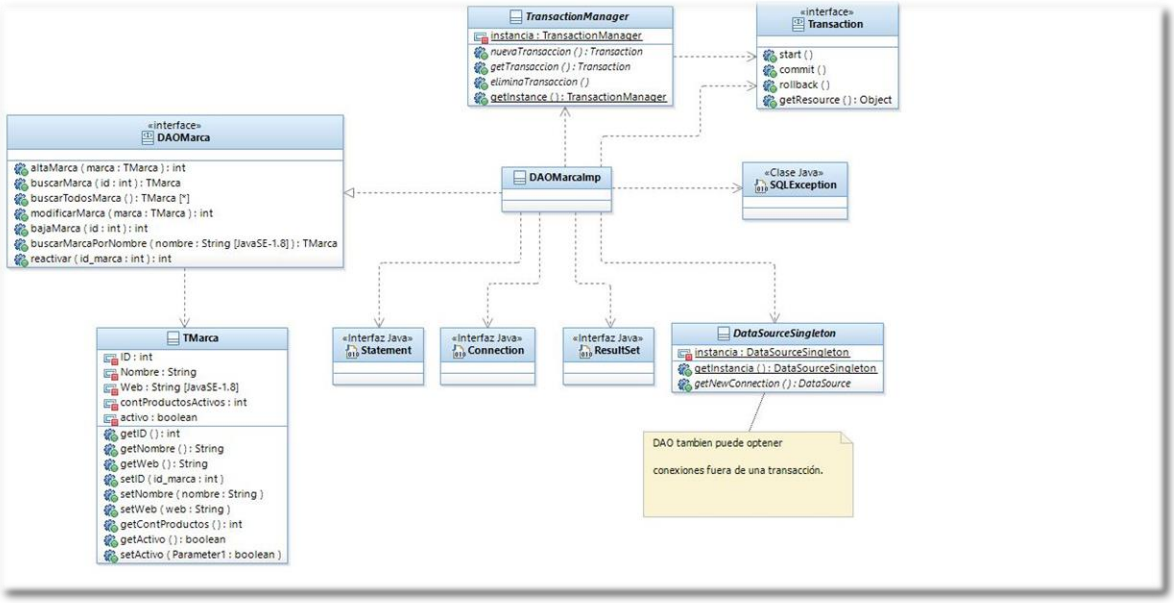



Diagrama de clases en el paquete de Integración-Cliente

- **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 probo 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 oden 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.

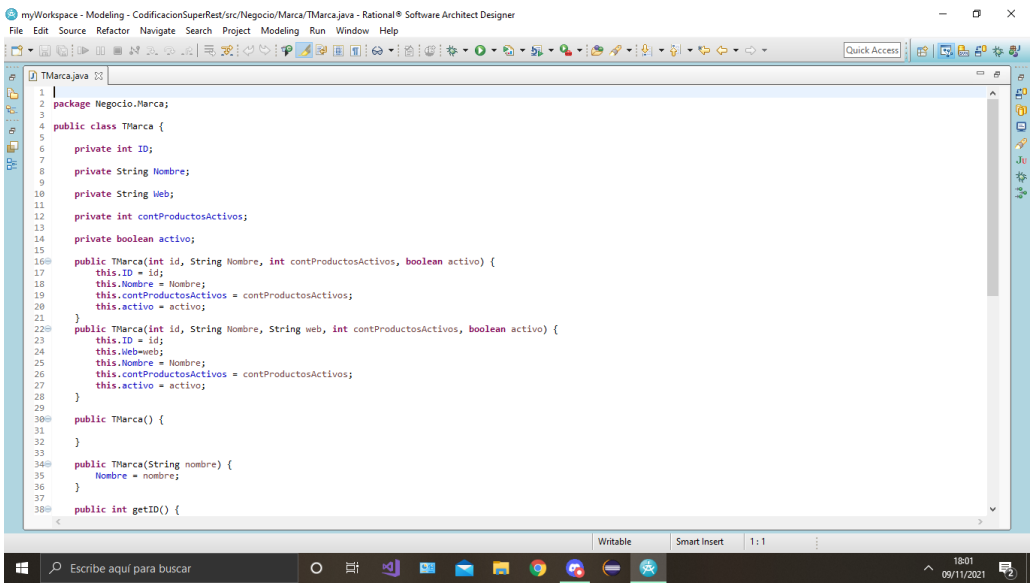
```
@Test
public void bCrearSeccionYaExistente(){
    TSeccion seccion = new TSeccion();
    seccion.setZona("Carniceria");
    seccion.setPasillo(1);
    int res = daoSeccion.crearSeccion(seccion);

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

# Patrones empleados en el proyecto

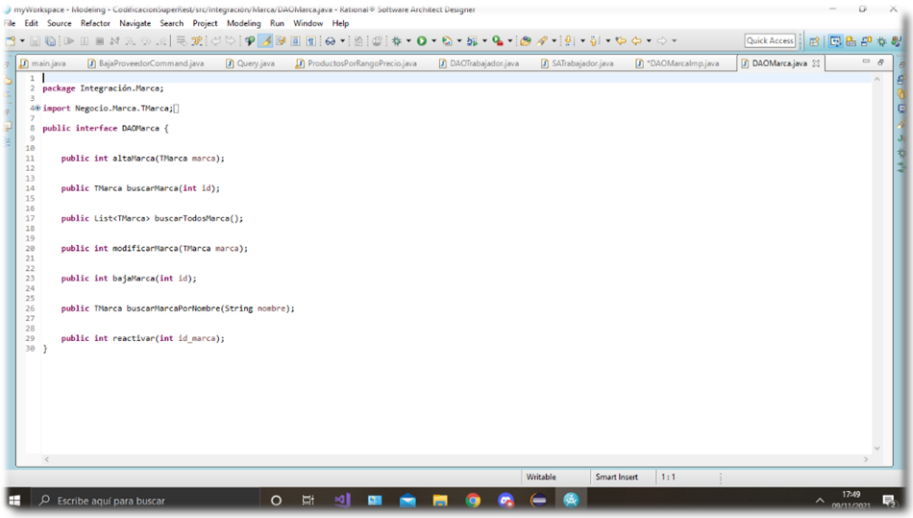
1. **Patrón Singleton.** El cual garantiza **que sólo hay una instancia** de una clase garantizada por ella misma, las utilizamos en **factorías**, distintas clases de **presentación** y en el **controlador.**

```java
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.

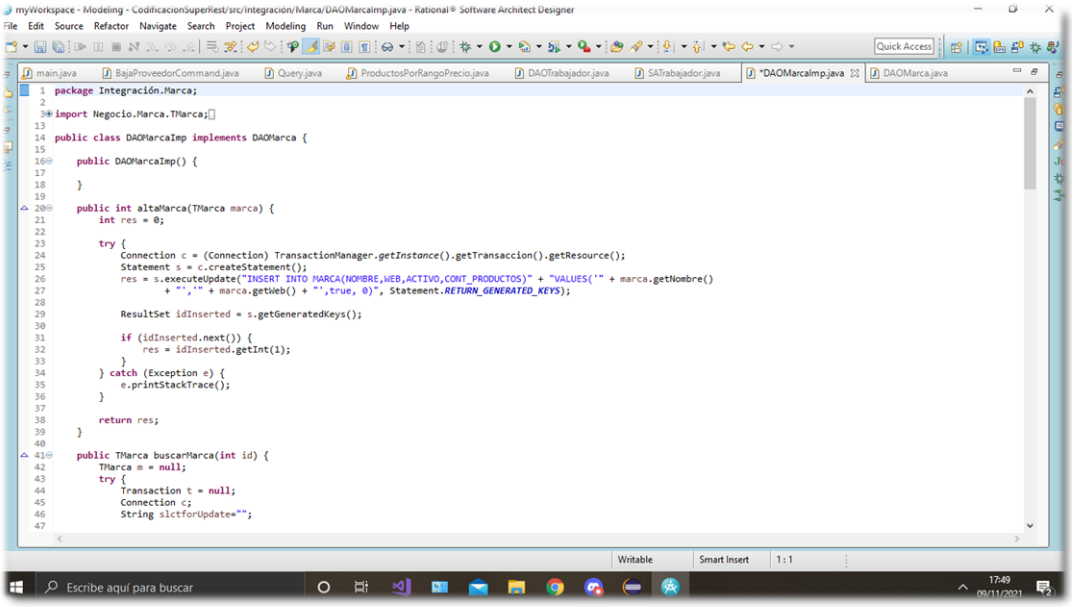


3. **Patrón DAO (Data Access Object).** El cual es utilizado para acceder a la de base de datos y proporciona acceso para el manejo de las peticiones de la capa de presentación, lo utilizamos en la capa de integración.

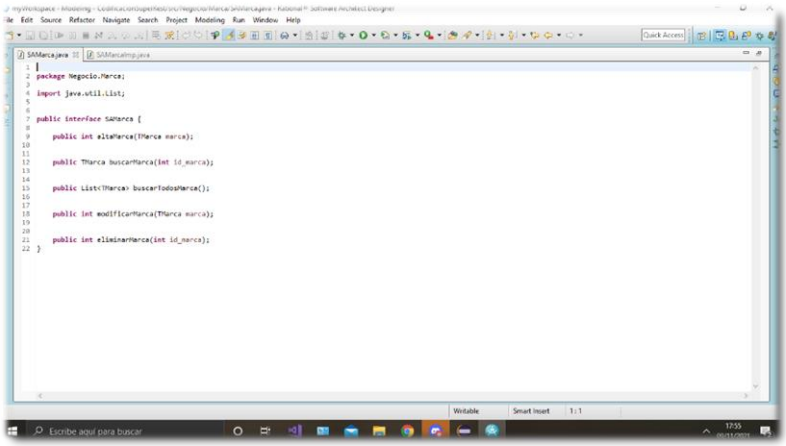


```
package Integración.Marca;

import Negocio.Marca.TMarca;

public interface DAOMarca {

    public int altaMarca(TMarca marca);

    public TMarca buscarMarca(int id);

    public List<TMarca> buscarTodosMarca();

    public int modificarMarca(TMarca marca);

    public int bajaMarca(int id);

    public TMarca buscarMarcaPorNombre(String nombre);

    public int reactivar(int id_marca);
}
```

## Implementación del Patrón DAO

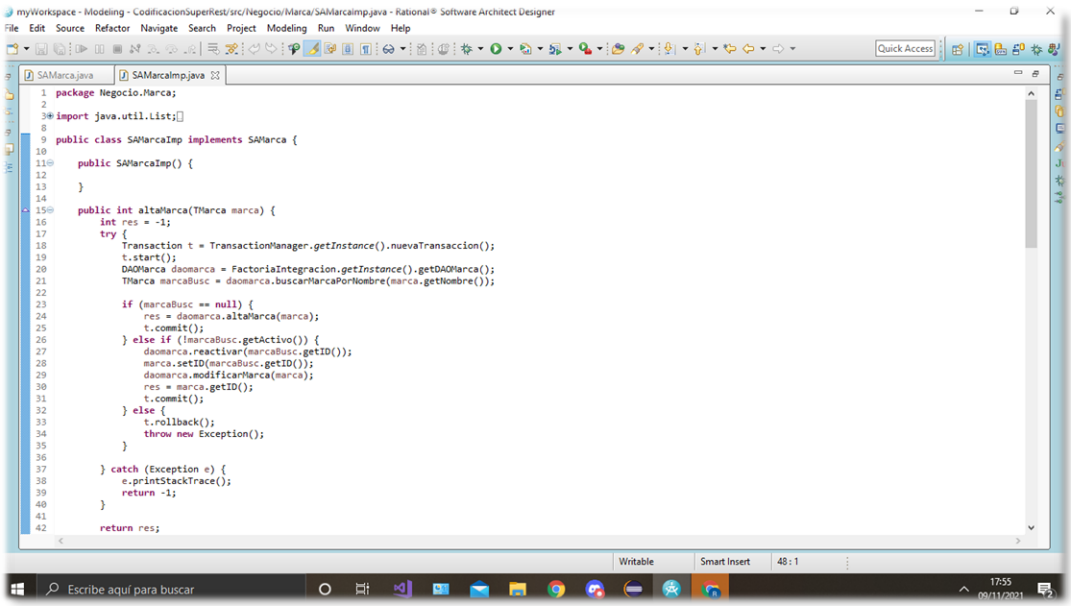


```
package Integración.Marca;

import Negocio.Marca.TMarca;

public class DAOMarcaImp implements DAOMarca {

    public DAOMarcaImp() {

    }

    public int altaMarca(TMarca marca) {
        int res = 0;

        try {
            Connection c = (Connection) TransactionManager.getInstance().getTransaccion().getResource();
            Statement s = c.createStatement();
            res = s.executeUpdate("INSERT INTO MARCA(NOMBRE,WEB,ACTIVO,CONT_PRODUCTOS)" + "VALUES('" + marca.getNombre()
                    + "','" + marca.getWeb() + "',true, 0)", Statement.RETURN_GENERATED_KEYS);

            ResultSet idInserted = s.getGeneratedKeys();

            if (idInserted.next()) {
                res = idInserted.getInt(1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return res;
    }

    public TMarca buscarMarca(int id) {
        TMarca m = null;
        try {
            Transaction t = null;
            Connection c;
            String sIctforUpdate="";
```



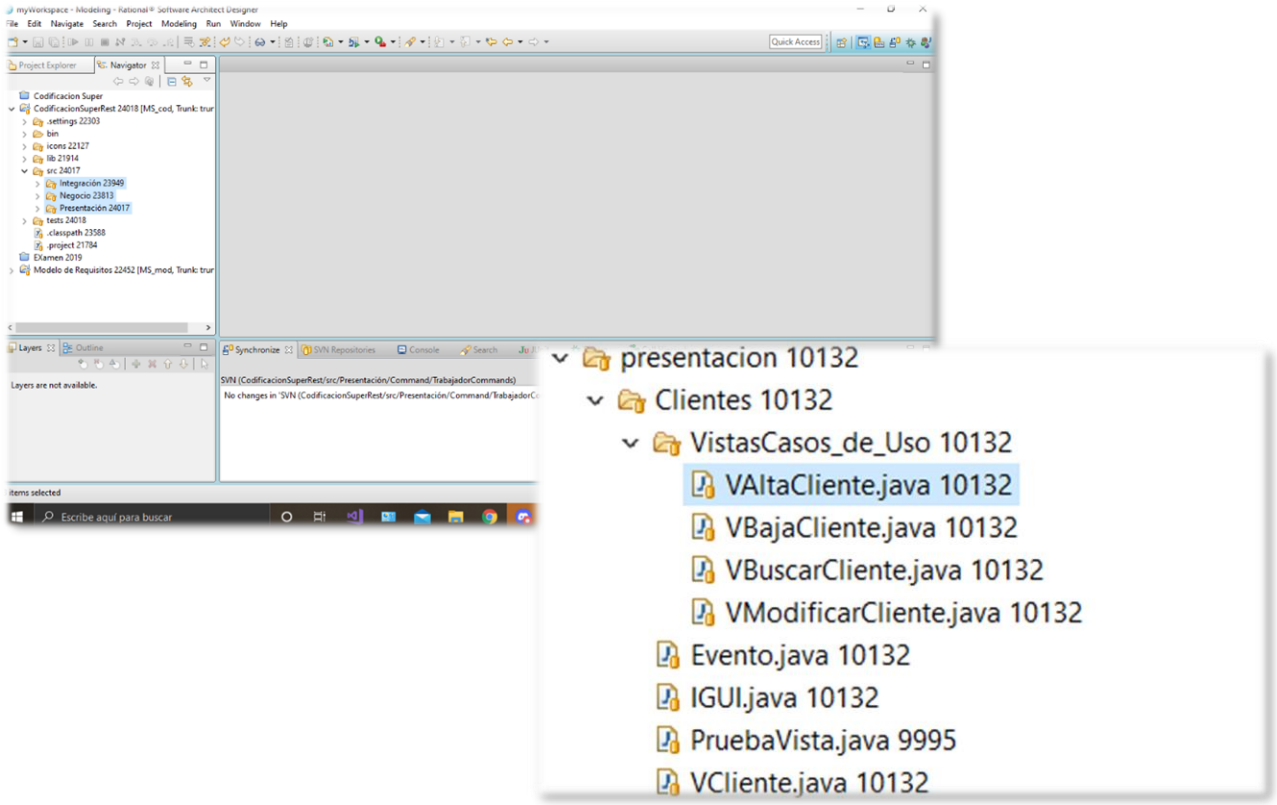
4. **Servicio de Aplicación.** El cual se centraliza en lógica de negocio que la aplicamos a objetos de negocio. La utilizamos en negocio.

## Implementación del patrón Servicio de Aplicación

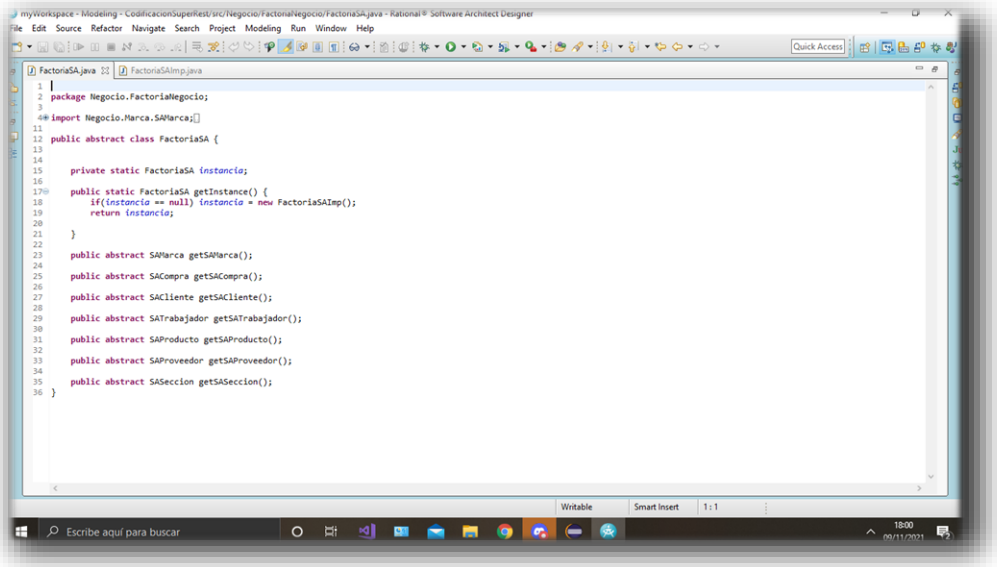


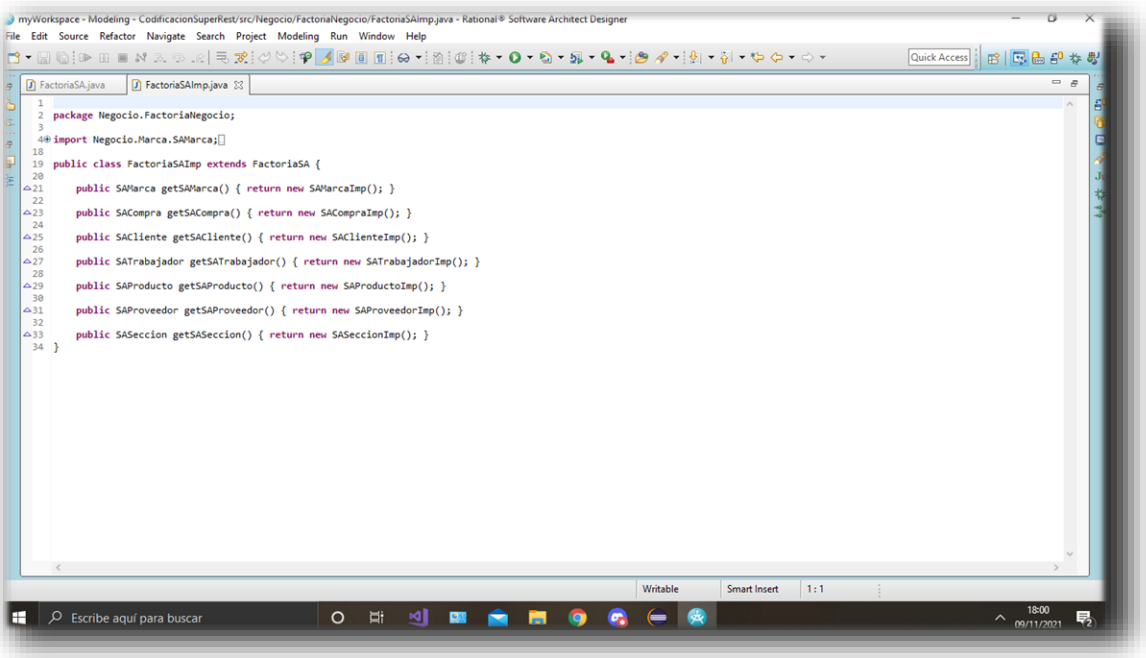
```
package Negocio.Marca;

import java.util.List;

public class SAMarcaImp implements SAMarca {

    public SAMarcaImp() {

    }

    public int altaMarca(TMarca marca) {
        int res = -1;
        try {
            Transaction t = TransactionManager.getInstance().nuevaTransaccion();
            t.start();
            DAOMarca daomarca = FactoriaIntegracion.getInstance().getDAOMarca();
            TMarca marcaBusc = daomarca.buscarMarcaPorNombre(marca.getNombre());

            if (marcaBusc == null) {
                res = daomarca.altaMarca(marca);
                t.commit();
            } else if (!marcaBusc.getActivo()) {
                daomarca.reactivar(marcaBusc.getID());
                marca.setID(marcaBusc.getID());
                daomarca.modificarMarca(marca);
                res = marca.getID();
                t.commit();
            } else {
                t.rollback();
                throw new Exception();
            }

        } catch (Exception e) {
            e.printStackTrace();
            return -1;
        }

        return res;
```

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

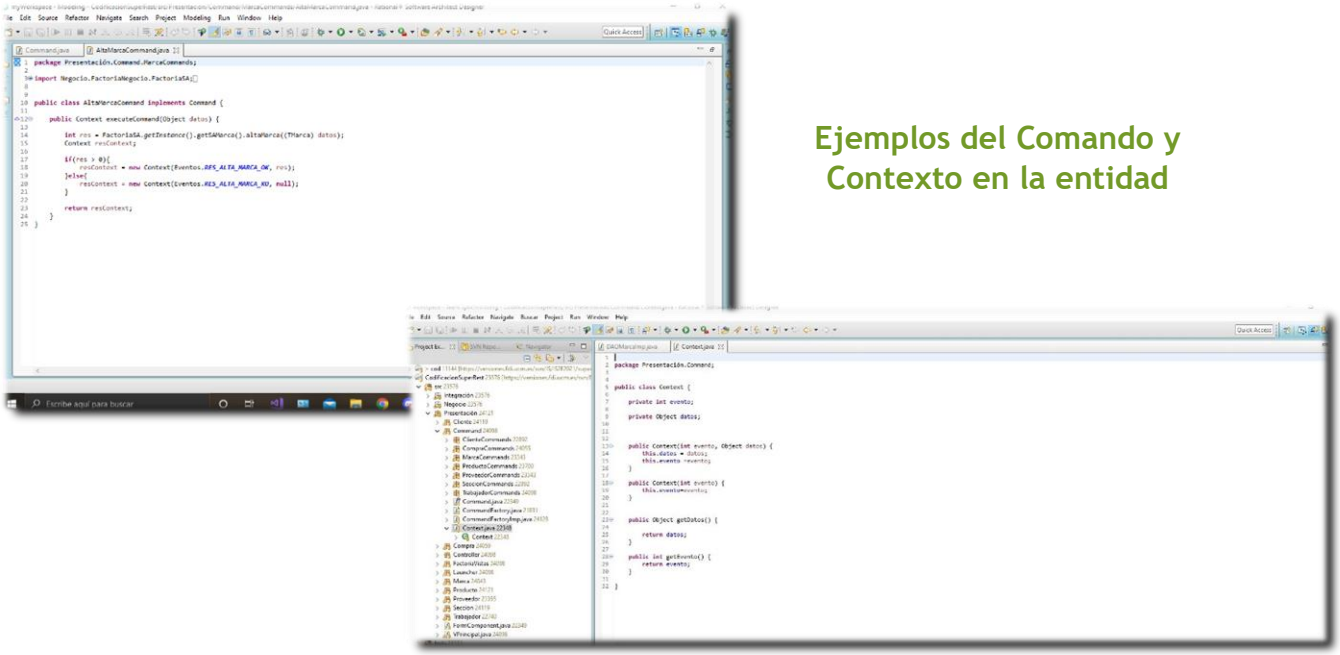



6. **Factoría abstracta.** El cual es utilizado por el DAO y SA y genera familias de objetos relacionados. Crea objetos, pero devuelve interfaces.



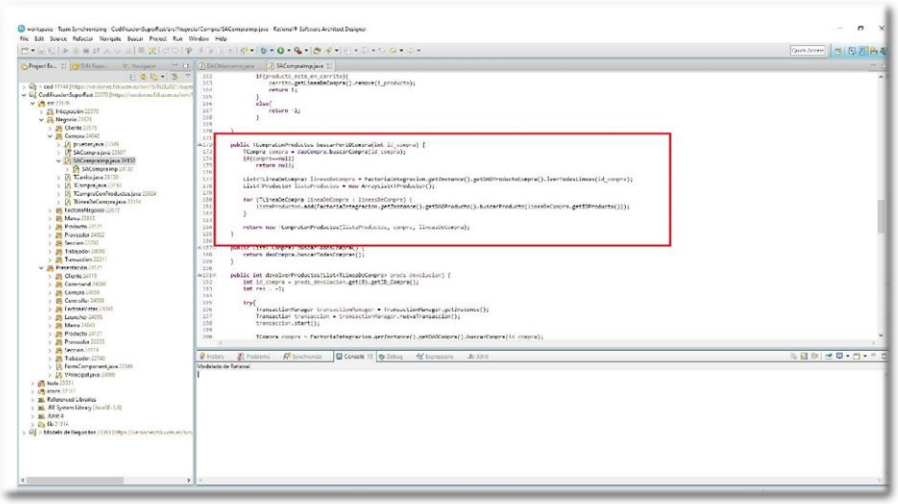



7. **Service To Worker.** Lo implementamos en nuestro proyecto para poder ejecutar lógica de negocio concreta y a partir de la respuesta obtenida generar/actualizar una vista concreta. Para ello nos apoyamos en dos elementos:
   a. **Comando.** Se encarga de llamar a la lógica de negocio y tras recibir la respuesta, mandar ejecutar una u otra vista.
   b. **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, que nos proporciona toda la información encapsulada para poder manipularla sin tener que llamar al SA de producto y de compras y que nos proporcionen uno a uno sus elementos.



## Ponderación de los miembros del equipo

El factor de corrección lo pondremos sobre 1:

LUIS ENRIQUE BARRERO PEÑA = 0.8 Por baja participación.

JOSE LUIS BARTHA DE LAS PEÑAS = 1

BRYAN EDUARDO CÓRDOVA ASCURRA = 1

CARLOS FORRIOL MOLINA = 1

PABLO GAMO GONZÁLEZ = 1.2 Por su imprescindible aportación al desarrollo del proyecto.

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