Clases:

Las clases principales que hemos utilizado durante este proyecto son:

Concesionario

Esta clase la utilizamos como centro neurálgico de la aplicación. Guardamos los datos necesarios en los arrays sobre Clientes, Secciones y Ventas. Además, gestionamos la manipulación de sus datos en esta clase. También se ha creado un método que utilizamos para leer el fichero donde se guardan los datos y e impide que se creen más de un objeto sobre esa clase.

Coche

La clase nos permite mostrar información de los coches. También tiene los métodos para la asignación de una matrícula y el stock.

Cliente

Esta clase nos permite mostrar la información de un cliente. Donde guardaremos su ID, su teléfono y si quiere información aparte.

Sección

Aquí guardaremos los datos sobre los coches que tienen cada una de las secciones. Aparte de los métodos de agregación, eliminación y modelación de los datos de los coches que componen dicha sección.

Venta

Esta clase nos sirve para guardar los datos sobre las ventas. Tanto ID de las ventas como las matrículas de los coches que se venden en el concesionario. Aparte tenemos todos los métodos de asignación de datos.

Patrones:

En esta práctica hemos utilizado tres patrones: Decorator, Composite y el Sigleton.

En el principio del programa encontramos el patrón Singleton introducido en la clase Concesionario. Este patrón nos sirve para que no tengamos creadas más instancias de los datos que utilizaremos a lo largo del proyecto.

Por otro lado, hemos utilizado el patrón Composite para modelar los menús de interacción. Este patrón nos ha sido de gran utilidad. En un principio tenemos la clase padre que se llama Menú, una clase controlador llamada MenuPrincipal y el resto de los menús: MenuCliente, MenuCoche, MenuSeccion....

Para finalizar hemos utilizado el patrón Decorator para modelar las mejoras que tendrán los coches al comprarlos. Aunque ha sido bastante lioso creemos que ha salido una buena práctica en la que hemos podido practicar todo lo aprendido durante esta asignatura.

Código Fuente:

Menu:

```
package Menus;

import Entradas.MyInPut;
import Tienda.*;

public abstract class Menu {
    private Concesionario concesionario;
    public Menu(Concesionario concesionario) {this.concesionario =
concesionario;}

   /**
     * Obtiene el concesionario asociado.
     *
     * @return el objeto {@code Concesionario} correspondiente.
     */
    public Concesionario getConcesionario() {return concesionario;}

    /**
     * Ejecuta un ciclo interactivo para procesar opciones del
usuario.
     *
    * Este método continúa ejecutándose mientras el usuario
introduzca "s"
    * como respuesta en el método {@code ejecutarOpciones}.
     */
    public void ejecutar() {
        String respuesta = "";
        do {
```

```
            respuesta = ejecutarOpciones();

        } while (respuesta.equals("s"));
    }

    /**
     * Ejecuta las opciones disponibles para el usuario y procesa su
selección.
     *
    * Este método presenta las opciones al usuario, captura la
entrada, la procesa
     * y determina la respuesta que controlará el flujo del
programa.
     *
    * @return una cadena que representa la respuesta procesada. Si
no se puede ejecutar
     *         la opción seleccionada, devuelve "s" para indicar que
el ciclo debe continuar.
     */
    public String ejecutarOpciones(){
    opciones();
    String respuesta =MyInPut.readString();
    respuesta = seleccion(respuesta);
    if (respuesta.equals("")) {
        System.out.println("No se puede ejecutar el opcion");
        return"s";
    }else return respuesta;
    }

    /**
     * Procesa y valida la respuesta proporcionada por el usuario.
     *
    * Este método debe ser implementado por las subclases para
definir
    * cómo manejar y transformar la respuesta del usuario según la
lógica específica.
     *
    * @param respuesta la entrada proporcionada por el usuario.
     * @return una cadena que representa la respuesta procesada.
     */
    public abstract String seleccion(String respuesta);
```

```
    /**
     * Muestra o define las opciones disponibles para el usuario.
     *
    * Este método debe ser implementado por las subclases para
especificar
     * las opciones que se presentarán al usuario en el contexto
correspondiente.
     */
    public abstract void opciones();
}
```

MenuPrincipal:

```
package Menus;
import Tienda.Concesionario;
import Entradas.MyInPut;
import java.util.ArrayList;
public class MenuPrincipal extends Menu{
    private ArrayList<Menu> Menus = new ArrayList<>();
    public MenuPrincipal(Concesionario c){
        super(c);
        Menus.add(new MenuSecciones(c));
        Menus.add(new MenuCoche(c));
        Menus.add(new MenuVentas(c));
        Menus.add(new MenuCliente(c));
    }

    /**
     * Muestra las opciones disponibles en el sistema para que el
usuario seleccione una acción.
     *
    * Este método imprime un menú en la consola con las opciones
numeradas,
    * permitiendo al usuario elegir entre diferentes gestiones, como
secciones, coches, ventas y clientes.
     */
    public void opciones(){
        System.out.println("");
        System.out.println("Seleccione una opcion:");
        System.out.println("0.- Salir del programa.");
        System.out.println("1. Gestion de Secciones.");
        System.out.println("2. Gestion de coches.");
```

```java
        //System.out.println("3. Gestion de ventas.");
        System.out.println("4. Gestion de clientes.");
    }

    /**
     * Procesa la cadena proporcionada como entrada y devuelve un
resultado.
     *
    * En esta implementación específica, el método no realiza
ninguna operación
     * y siempre devuelve {@code null}.
     *
    * @param s la cadena de entrada proporcionada por el usuario.
     * @return {@code null}, ya que este método no está
implementado.
     */
    public String seleccion(String s){
    return  null;
    }

   /**
     * Muestra las opciones disponibles, solicita una selección del
usuario y ejecuta la opción seleccionada.
     *
    * Este método valida la entrada del usuario, maneja errores de
formato en caso de que
    * la entrada no sea un número válido, y ejecuta la opción
correspondiente del menú.
     *
    * @return una cadena que controla el flujo del programa:
     *         <ul>
     *         <li>{@code "s"}: para continuar el ciclo
principal.</li>
     *         <li>{@code "n"}: para indicar que el ciclo debe
detenerse.</li>
     *         </ul>
     */
    public String ejecutarOpciones(){
        opciones();
        System.out.println("Elija una opcion: ");
        String opcion = MyInPut.readString();
        int i;
```

```
        try{
            i = Integer.parseInt(opcion);
        }catch(NumberFormatException e){
            System.out.println("El valor de la opcion no es
valido.");
            return "s";
        }
        if(i> 0 && i<=Menus.size()){
            Menus.get(i - 1).ejecutar();
            return "s";
        }else if(i<0 || i>Menus.size()){
            System.out.println("Opcion no valida.");
            return "s";
        }else return "n";
    }
}
```

MenuSecciones:

```
package Menus;

import Entradas.MyInPut;
import Tienda.Coche;
import Tienda.Concesionario;
import Tienda.Seccion;

import java.util.ArrayList;
public class MenuSecciones extends Menu{
    public MenuSecciones(Concesionario c){super(c);}

    /**
     * Agrega una nueva sección al concesionario.
     *
    * Este método solicita al usuario el nombre de la sección. Si la
sección no existe,
    * solicita una descripción adicional, crea la sección y la
agrega al concesionario.
    * Si la sección ya existe, muestra un mensaje de error e invita
al usuario a intentarlo nuevamente.
     *
    * Pasos del método:
     * <ul>
```

```
     *   <li>Solicita al usuario el nombre de la sección.</li>
     *   <li>Verifica si la sección ya existe en el
concesionario.</li>
     *   <li>Si la sección no existe:
     *       <ul>
     *         <li>Solicita una descripción de la sección.</li>
     *         <li>Crea la nueva sección.</li>
     *         <li>Agrega la sección al concesionario.</li>
     *       </ul>
     *   </li>
     *   <li>Si la sección ya existe, muestra un mensaje de
error.</li>
     * </ul>
     */
    public void agregarSeccion(){
        System.out.println("Escriba el nombre de la seccion: ");
        String s = MyInPut.readString();
        Seccion se = getConcesionario().recuperarSeccion(s);
        if (se ==null){
            System.out.println("Introduzca una descripcion de la
seccion: ");
            String desc = MyInPut.readString();
            se = new Seccion(s,desc);
            getConcesionario().agregarSeccion(se);
        }else{
            System.out.println("El seccion ya existe.");
            System.out.println("Vuelva a intentarlo.");
        }
    }

    /**
     * Elimina una sección del concesionario si no contiene coches.
     *
    * Este método solicita al usuario el nombre de la sección a
eliminar. Si la sección existe y no
    * contiene coches, se elimina del concesionario. Si la sección
contiene coches, muestra un mensaje
    * indicando que no puede ser eliminada. Si la sección no existe,
invoca el método {@code excepcion()}
    * para manejar la situación.
     *
    * Pasos del método:
```

```
     * <ul>
     *   <li>Solicita al usuario el nombre de la sección.</li>
     *   <li>Recupera la sección del concesionario.</li>
     *   <li>Si la sección existe:
     *       <ul>
     *         <li>Comprueba si la lista de coches en la sección
está vacía.</li>
     *         <li>Si está vacía, elimina la sección del
concesionario.</li>
     *         <li>Si no está vacía, muestra un mensaje indicando
que no puede eliminarse.</li>
     *       </ul>
     *   </li>
     *   <li>Si la sección no existe, llama al método {@code
excepcion()}.</li>
     * </ul>
     */
    public void EliminarSeccion(){
        System.out.println("Escriba el nombre de la seccion: ");
        String s = MyInPut.readString();
        Seccion se = getConcesionario().recuperarSeccion(s);
        ArrayList<Coche> coches = se.getCoches();
        if (se !=null) {
            if (coches.size() == 0) {
                getConcesionario().eliminarSeccion(se);
            }else{
                System.out.println("La seccion contiene coches.");
                System.out.println("No se puede eliminar.");
            }
        }else{
            excepcion();
        }
    }

    /**
     * Muestra un mensaje de error indicando que la sección no
existe en el sistema.
     *
    * Este método informa al usuario que la sección con el nombre
proporcionado no está
    * registrada en el sistema e invita a intentarlo nuevamente.
     */
```

```java
    public void excepcion(){
        System.out.println("La seccion no existe.");
        System.out.println("Vuelva a intentarlo.");
    }

    /**
     * Muestra el menú de opciones para la gestión de secciones en
el concesionario.
     *
    * Este método imprime un conjunto de opciones en la consola,
permitiendo al usuario
    * seleccionar acciones relacionadas con la gestión de secciones.
Las opciones incluyen
    * crear una nueva sección, eliminar una existente o mostrar
todas las secciones registradas.
     *
    * Opciones disponibles:
     * <ul>
     *   <li>1: Crear una nueva sección.</li>
     *   <li>2: Eliminar una sección existente.</li>
     *   <li>3: Mostrar todas las secciones registradas.</li>
     *   <li>0: Salir del menú.</li>
     * </ul>
     */
    public void opciones(){
        System.out.println("Gestion de Secciones");
        System.out.println("Elija una opcion:");
        System.out.println("1. Crear Seccion");
        System.out.println("2. Eliminar Seccion");
        System.out.println("3. Mostrar todas las secciones");
        System.out.println("0. Salir");
    }

    /**
     * Procesa la opción seleccionada por el usuario y ejecuta la
acción correspondiente para la gestión de secciones.
     *
    * Este método utiliza un bloque `switch` para determinar qué
acción realizar en función de la opción
    * seleccionada por el usuario. Las opciones disponibles son:
     * <ul>
     *   <li>"0": Salir del menú y devolver {@code "n"}.</li>
```

```
     *   <li>"1": Llama al método {@code agregarSeccion()} para
crear una nueva sección y devuelve {@code "s"}.</li>
     *   <li>"2": Llama al método {@code EliminarSeccion()} para
eliminar una sección existente y devuelve {@code "s"}.</li>
     *   <li>"3": Llama al método {@code
getConcesionario().ListarSecciones()} para mostrar todas las
secciones y devuelve {@code "s"}.</li>
     * </ul>
     * Si la opción no coincide con ninguna de las anteriores,
devuelve {@code "n"} para salir del ciclo principal.
     *
    * @param opcion la opción seleccionada por el usuario como
cadena.
     * @return una cadena de control:
     *         <ul>
     *           <li>{@code "s"} para continuar en el ciclo
principal.</li>
     *           <li>{@code "n"} para detener el ciclo principal o
para manejar opciones inválidas.</li>
     *         </ul>
     */
    public String seleccion(String opcion){
        switch(opcion){
            case "0": return "n";
            case "1":{
                agregarSeccion();
                return "s";
            }
            case "2":{
                EliminarSeccion();
                return "s";
            }
            case "3":{
            getConcesionario().ListarSecciones();
            return "s";
            }
        }return "n";
    }
}
```

MenuCoches:

```java
package Menus;

import Entradas.MyInPut;
import Tienda.Coche;
import Tienda.Concesionario;
import Tienda.Seccion;

public class MenuCoche extends Menu{
    public MenuCoche(Concesionario c){super(c);}

    /**
     * Agrega un coche a una sección específica del concesionario.
     *
    * Este método solicita información del coche al usuario, como el modelo, año y precio base,
    * y lo agrega a la sección indicada si no existe previamente. Si el coche ya está registrado,
    * muestra un mensaje de error e invita al usuario a intentarlo nuevamente.
     *
    * El ID del coche se genera combinando el modelo y el año.
     *
    * Pasos del método:
     * <ul>
     *   <li>Solicita la sección donde se agregará el coche.</li>
     *   <li>Solicita detalles del coche: modelo, año y precio base.</li>
     *   <li>Verifica si el coche ya existe en la sección especificada.</li>
     *   <li>Si no existe, crea el coche y lo agrega a la sección correspondiente.</li>
     *   <li>Si ya existe, muestra un mensaje de error.</li>
     * </ul>
     */
    public void agregarCoche(){
        System.out.println("Cual es la seccion en la que lo va a meter:");
        String seccion = MyInPut.readString();
        System.out.println("Introduzca el modelo del coche:");
        String modelo = MyInPut.readString();
        System.out.println("Introduzca el año del coche: ");
```

```
        String ano = MyInPut.readString();
        String id = modelo + "-" + ano;
        System.out.println("Hola");
        if(getConcesionario().recuperarSeccion(seccion) == null){
            System.out.println("La seccion no existe.");
        }
        Coche cocheNuevo=
getConcesionario().recuperarSeccion(seccion).RecuperarCoche(id);
        if(cocheNuevo == null){
            System.out.println("Introduzca el precio base del coche:
");
            double precioBase = MyInPut.readDouble();
            cocheNuevo = new Cochebase(id, precioBase);
            Seccion categoria =
getConcesionario().recuperarSeccion(seccion);
            categoria.agregarCoche(cocheNuevo);
        }else{
            System.out.println("El coche ya existe en el sistema");
            System.out.println("Vuelva a intentarlo");
        }
    }

    /**
     * Muestra la información de un coche específico dentro de una
sección del concesionario.
     *
    * Este método solicita al usuario el ID de la sección y el ID
del coche, busca el coche
    * en la sección especificada y, si lo encuentra, muestra su
información llamando al método
    * {@code mostrarInfo()} del coche. Si el coche no existe, invoca
el método {@code excepcion()}
    * para manejar la situación.
     *
    * Pasos del método:
     * <ul>
     *   <li>Solicita al usuario el ID de la sección.</li>
     *   <li>Solicita al usuario el ID del coche.</li>
     *   <li>Busca el coche en la sección indicada.</li>
     *   <li>Si el coche existe, muestra su información.</li>
     *   <li>Si no existe, muestra un mensaje de error llamando a
{@code excepcion()}.</li>
```

```
     * </ul>
     */
    public void MostrarCoche(){
        System.out.println("Introduzca el id de la seccion: ");
        String seccion = MyInPut.readString();
        System.out.println("Introduzca el id del coche: ");
        String id = MyInPut.readString();
        Coche coche =
getConcesionario().recuperarSeccion(seccion).RecuperarCoche(id);
        if(coche != null){
            coche.mostrarInfo();
        }else{
            excepcion();
        }
    }

    /**
     * Muestra un mensaje de error indicando que el coche no existe
en el sistema.
     *
    * Este método informa al usuario que el coche con el ID
proporcionado no está registrado
     * en el sistema e invita a intentarlo nuevamente.
     */
    public void excepcion(){
        System.out.println("El coche no existe en el sistema.");
        System.out.println("Vuelva a intentarlo.");
    }

    /**
     * Agrega o actualiza el stock de un coche específico en una
sección del concesionario.
     *
    * Este método solicita al usuario el ID de la sección y el ID
del coche,
    * busca el coche en la sección especificada y, si lo encuentra,
permite al usuario
    * introducir la cantidad de stock para actualizar. Si el coche
no existe,
    * se invoca el método {@code excepcion()} para manejar la
situación.
     *
```

```java
    * Pasos del método:
     * <ul>
     *   <li>Solicita al usuario el ID de la sección.</li>
     *   <li>Solicita al usuario el ID del coche.</li>
     *   <li>Busca el coche en la sección especificada.</li>
     *   <li>Si el coche existe, solicita la cantidad de stock y
actualiza el stock del coche
    *       usando el método {@code setStock()}.</li>
     *   <li>Si el coche no existe, muestra un mensaje de error
llamando a {@code excepcion()}.</li>
     * </ul>
     */
    public void agregarStock(){
        System.out.println("Introduzca el id de la seccion: ");
        String seccion = MyInPut.readString();
        System.out.println("Introduzca el id del coche: ");
        String id = MyInPut.readString();
        Coche coche =
getConcesionario().recuperarSeccion(seccion).RecuperarCoche(id);
        if(coche != null){
            System.out.println("Introduzca el stock del coche: ");
            int stock = MyInPut.readInt();
            coche.setStock(stock);
        }else{
            excepcion();
        }
    }

    /**
     * Muestra la lista de coches de una sección específica del
concesionario.
     *
    * Este método solicita al usuario el ID de la sección, verifica
si la sección existe
    * y, si es así, llama al método {@code ListarCoches()} de la
sección para mostrar
    * todos los coches que pertenecen a ella. Si la sección no
existe, muestra un mensaje
    * de error indicando que la categoría no está registrada.
     *
    * Pasos del método:
     * <ul>
```

```
     *   <li>Solicita al usuario el ID de la sección.</li>
     *   <li>Busca la sección en el concesionario.</li>
     *   <li>Si la sección existe, muestra la lista de coches
llamando a {@code ListarCoches()}.</li>
     *   <li>Si la sección no existe, muestra un mensaje de error y
solicita al usuario intentarlo nuevamente.</li>
     * </ul>
     */
    public void mostrarCoche(){
        System.out.println("Introduzca el id de la seccion: ");
        String seccion = MyInPut.readString();
        Seccion categoria =
getConcesionario().recuperarSeccion(seccion);
        if(categoria != null){
            categoria.ListarCoches();
        }else{
            System.out.println("La categoria no existe en el
sistema.");
            System.out.println("Vuelva a intentarlo.");
        }
    }

    /**
     * Muestra el menú de opciones para la gestión de coches.
     *
    * Este método imprime un conjunto de opciones en la consola,
permitiendo al usuario
    * seleccionar acciones relacionadas con la gestión de coches en
el concesionario.
    * Las opciones incluyen agregar un coche, mostrar datos de un
coche, aumentar el stock
    * y listar coches en una sección específica.
     */
    public void opciones(){
        System.out.println("");
        System.out.println("Gestion de coches.");
        System.out.println("Elija una opcion:");
        System.out.println("1. Dar alta coche.");
        System.out.println("2. Mostrar datos de un coche.");
        System.out.println("3. Aumentar el stock");
        System.out.println("4. Mostrar los coches de una seccion.");
        System.out.println("0. Salir");
```

```
    }

    /**
     * Procesa la opción seleccionada por el usuario y ejecuta la
acción correspondiente.
     *
    * Este método utiliza un bloque `switch` para determinar qué
acción realizar en función
    * de la opción proporcionada por el usuario. Las opciones
disponibles son:
     * <ul>
     *   <li>"0": Salir del menú y devolver {@code "n"}.</li>
     *   <li>"1": Llama al método {@code agregarCoche()} para dar de
alta un nuevo coche y devuelve {@code "s"}.</li>
     *   <li>"2": Llama al método {@code MostrarCoche()} para
mostrar los datos de un coche específico y devuelve {@code
"s"}.</li>
     *   <li>"3": Llama al método {@code agregarStock()} para
aumentar el stock de un coche y devuelve {@code "s"}.</li>
     *   <li>"4": Llama al método {@code mostrarCoche()} para listar
los coches de una sección y devuelve {@code "s"}.</li>
     * </ul>
     * Si la opción no coincide con ninguna de las anteriores,
devuelve una cadena vacía.
     *
    * @param opcion la opción seleccionada por el usuario como
cadena.
     * @return una cadena de control:
     *         <ul>
     *           <li>{@code "s"} para continuar en el ciclo
principal.</li>
     *           <li>{@code "n"} para detener el ciclo
principal.</li>
     *           <li>Cadena vacía si la opción es inválida.</li>
     *         </ul>
     */
    public String seleccion(String opcion){
        switch(opcion){
            case "0":
                return "n";
            case "1":{
                agregarCoche();
```

```
                return "s";
            }
            case "2":{
                MostrarCoche();
                return "s";
            }
            case "3":{
                agregarStock();
                return "s";
            }
            case "4":{
                mostrarCoche();
                return "s";
            }
        }return "";
    }

}
```

MenuVentas:

```
package Menus;

import Entradas.MyInPut;
import Tienda.*;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Locale;

public class MenuVentas extends Menu{


    public MenuVentas(Concesionario c) {super(c);}


    private String generarIdVenta() {
        int num =getConcesionario().numeroVentas();
        String id = String.format("V%08d",num);
        return id;
    }
```

```
    public void registarVenta() {
        try {
            System.out.println("Registrando la venta");
            System.out.println("Escribe cual es la seccion en donde
quieres comprar el coche:");
            String seccion = MyInPut.readString();
            Seccion s =
getConcesionario().recuperarSeccion(seccion);
            if (s==null){
                throw new Exception("No se encontro el seccion");
            }
            System.out.println("Escribe el id del coche que quieres
comprar:");
            String id = MyInPut.readString();
            Coche c = s.RecuperarCoche(id);
            if (c.getStock() < 0) {
                throw new Exception("No se puede comprar el coche");
            }
            System.out.println("Escribe el DNI/NIF del cliente:");
            String dni = MyInPut.readString();
            Cliente cliente =
getConcesionario().recuperarClientes(dni);
            if (cliente == null) {
                throw new Exception("No se encontro el cliente");
            }
            System.out.println("Elige si quieres mejora o no (1 si
,0 no):");
            int num = MyInPut.readInt();

            if (num == 1) {
                menuMejoras(c,0);
            }
            String id_venta = generarIdVenta();
            String matricula =
generarMatricula(getConcesionario().matriculaCoche());
            LocalDate hoy = LocalDate.now();

            Venta venta =new
Venta(id_venta,cliente,c,matricula,hoy);

            getConcesionario().anadirVentas(venta);
```

```
        }catch (Exception e) {
            System.out.println( e.getMessage());
            System.out.println("No se puede comprar el coche");
        }
    }

    public void  menuMejoras(Coche c , int num){
        System.out.println("Elige tipo de mejora (el numero) /n
1.Calefaccion /n 2.GPS /n 3.Llantas /n 4.Tapiceria de cuero");
        int eleccion = MyInPut.readInt();
        if (eleccion == num){
            System.out.println("No se puede añadir la misma
mejora");
        }else{
            switch (num) {
                case 1:
                    c= new Calefaccion(c);
                    break;
                case 2:
                    c= new GPS(c);
                    break;
                case 3:
                    c= new Llantas(c);
                    break;
                case 4:
                    c=new Tapiceria_Cuero(c);
                    break;
                default:
                    System.out.println("No has elegido ninguna
mejora");
            }
        }
        System.out.println("Quieres Añadir otra mejora: (1 si ,0
no)");
        num = MyInPut.readInt();
        if(num==1){
            menuMejoras(c,eleccion);
        }


    }
```

```java
    /**
     * Obtiene la matrícula del coche vendido.
     *
     * @return El numero de la matricula anterior.
     */
    public String generarMatricula(String matriculaAnterior) {
        // Validar la matrícula anterior
        if (matriculaAnterior == null ||
!matriculaAnterior.matches("\\d{4} [BCDFGHJKLMNPRSTVWXYZ]{3}")) {
            throw new IllegalArgumentException("Formato de matrícula
inválido.");
        }

        // Letras permitidas en las matrículas españolas
        String LETRAS_PERMITIDAS = "BCDFGHJKLMNPRSTVWXYZ";

        // Descomponer la matrícula en números y letras
        String numerosParte = matriculaAnterior.substring(0, 4);
        String letrasParte = matriculaAnterior.substring(5);

        // Incrementar la parte numérica
        int numeroActual = Integer.parseInt(numerosParte);
        numeroActual++;

        // Si se supera el límite, reiniciar los números y avanzar
las letras
        if (numeroActual > 9999) {
            numeroActual = 0;

            // Incrementar las letras
            char[] letrasArray = letrasParte.toCharArray();
            int primeraLetra =
LETRAS_PERMITIDAS.indexOf(letrasArray[0]);
            int segundaLetra =
LETRAS_PERMITIDAS.indexOf(letrasArray[1]);
            int terceraLetra =
LETRAS_PERMITIDAS.indexOf(letrasArray[2]);

            // Avanzar la última letra primero
            terceraLetra++;
            if (terceraLetra == LETRAS_PERMITIDAS.length()) {
```

```
                terceraLetra = 0;
                segundaLetra++;

                if (segundaLetra == LETRAS_PERMITIDAS.length()) {
                    segundaLetra = 0;
                    primeraLetra++;

                    if (primeraLetra == LETRAS_PERMITIDAS.length())
{
                        throw new IllegalArgumentException("Se han
agotado todas las combinaciones posibles de matrículas.");
                    }
                }
            }

            // Reconstruir las letras
            letrasParte = "" +
LETRAS_PERMITIDAS.charAt(primeraLetra) +
                    LETRAS_PERMITIDAS.charAt(segundaLetra) +
                    LETRAS_PERMITIDAS.charAt(terceraLetra);
        }

        // Formatear la parte numérica y generar la nueva matrícula
        return String.format("%04d %s", numeroActual, letrasParte);
    }



    public void mostarIforacionVenta(){
        System.out.println("Escribe el id de la venta");
        String id = MyInPut.readString();
        Venta v = getConcesionario().recuperarVenta(id);
        v.mostrarInformacion();
    }

    @Override
    public String seleccion(String s) {
        switch (s) {
            case "1": {
                registarVenta();
                return "s";
            }
```

```
            case "2": {
                mostarIforacionVenta();
                return "s";
            }
            case "3": {
                getConcesionario().mostarVentas();
                return "s";
            }
        }return "";
    }


    @Override
    public void opciones() {
        System.out.println(" ");
        System.out.println("Gestion de clientes");
        System.out.println("Seleccione una opcion: ");
        System.out.println("1. Registrar una Venta.");
        System.out.println("2. Mostrar Informacion de un venta
concreta.");
        System.out.println("3. Mostrar informacion de todas las
ventas.");
        System.out.println("0. Salir.");

    }
}
```

MenuCliente:

```
package Menus;
import Entradas.MyInPut;
import Tienda.Concesionario;
import Tienda.Cliente;

public class MenuCliente extends Menu{
    public MenuCliente(Concesionario c) { super(c); }

    /**
     * Solicita datos del cliente al usuario y lo agrega al
concesionario si no existe previamente.
     *
    * Este método verifica si un cliente con el ID proporcionado ya
está registrado en el sistema.
```

```java
     * Si no lo está, solicita información adicional al usuario
(nombre, teléfono, preferencia de recibir información)
     * y lo añade al sistema. Si el cliente ya existe, muestra un
mensaje de error.
     */
    private void agregarCliente() {
        System.out.println(" Introduzca el id del cliente: ");
        String id = MyInPut.readString();
        Cliente c =getConcesionario().recuperarClientes(id);
        if (c == null) {
            System.out.println("Introduzca el nombre del cliente:");
            String nombre = MyInPut.readString();
            System.out.println("Introduzca el telefono del
cliente:");
            int telefono = MyInPut.readInt();
            System.out.println("El cliente desea recibir informacion
(1 = Si // 0 = No): ");
            int info = MyInPut.readInt();
            c = new Cliente(id, nombre, telefono, getInfo(info));
            getConcesionario().anadirClientes(c);
        }else{
            System.out.println("El cliente ya existe en el
sistema.");
            System.out.println("Vuelva a intentarlo.");
        }
    }

    /**
     * Convierte un valor entero en un valor booleano.
     *
    * Este método interpreta el entero proporcionado como una
preferencia:
     * <ul>
     *   <li>Si {@code info} es igual a 1, devuelve {@code
true}.</li>
     *   <li>En cualquier otro caso, devuelve {@code false}.</li>
     * </ul>
     *
    * @param info un entero que representa la preferencia (1 = Sí,
otros valores = No).
     * @return {@code true} si el valor es 1, de lo contrario {@code
false}.
```

```java
     */
    public boolean getInfo(int info){
        if (info == 1) {
            return true;
        }
        return false;
    }

    /**
     * Muestra la información de un cliente identificado por su ID.
     *
    * Este método solicita al usuario que introduzca el ID de un
cliente,
    * busca al cliente en el concesionario y, si lo encuentra,
muestra su información
    * llamando al método {@code MostrarInfo} del cliente. Si el
cliente no existe,
    * invoca el método {@code excepcion()} para manejar la
situación.
     */
    public void mostrarCliente(){
        System.out.println("Introduzca el id del cliente: ");
        String id = MyInPut.readString();
        Cliente c = getConcesionario().recuperarClientes(id);
        if (c == null) {
            excepcion();
        }else{
            c.MostrarInfo();
        }
    }

    /**
     * Muestra un mensaje de error indicando que el ID del cliente
no existe.
     *
    * Este método informa al usuario que el cliente con el ID
proporcionado no se encuentra
    * en el sistema e invita a intentarlo nuevamente.
     */
    public void excepcion(){
        System.out.println("El id del cliente no existe");
        System.out.println("Vuelva a intentarlo");
```

```
    }

    /**
     * Muestra el menú de opciones para la gestión de clientes.
     *
    * Este método imprime un conjunto de opciones en la consola,
permitiendo al usuario
    * seleccionar acciones relacionadas con la gestión de clientes,
como dar de alta un nuevo cliente,
    * obtener información de clientes específicos o mostrar
información general.
     */
    public void opciones(){
        System.out.println(" ");
        System.out.println("Gestion de clientes");
        System.out.println("Seleccione una opcion: ");
        System.out.println("1. Dar de alta un nuevo cliente.");
        System.out.println("2. Informacion de un cliente
concreto.");
        System.out.println("3. Mostrar informacion de todos los
clientes.");
        System.out.println("4. Mostrar informacion de los clientes
que quieran publicidad.");
        System.out.println("0. Salir.");
    }


   /**
     * Procesa la opción seleccionada por el usuario y ejecuta la
acción correspondiente.
     *
    * Este método utiliza un bloque `switch` para determinar la
acción a realizar según
    * la opción proporcionada como entrada. Las opciones disponibles
son:
     * <ul>
     *   <li>"1": Agregar un nuevo cliente llamando a {@code
agregarCliente()}.</li>
     *   <li>"2": Mostrar información de un cliente llamando a
{@code mostrarCliente()}.</li>
     *   <li>"3": Listar todos los clientes llamando a {@code
getConcesionario().ListarClientes()}.</li>
```

```
     *   <li>"4": Listar los clientes interesados en recibir
publicidad llamando a
    *       {@code getConcesionario().ListarClientesInfo()}.</li>
     *   <li>"0": Salir del menú devolviendo {@code "n"}.</li>
     * </ul>
     * Si la opción no es válida, devuelve una cadena vacía.
     *
    * @param s la opción seleccionada por el usuario como cadena.
     * @return una cadena de control:
     *         <ul>
     *           <li>{@code "s"} para continuar en el ciclo
principal.</li>
     *           <li>{@code "n"} para detener el ciclo
principal.</li>
     *           <li>Cadena vacía si la opción es inválida.</li>
     *         </ul>
     */
    public String seleccion(String s){
        switch (s) {
            case "1": {
                agregarCliente();
                return "s";
            }
            case "2": {
                mostrarCliente();
                return "s";
            }
            case "3": {
                getConcesionario().ListarClientes();
                return "s";
            }
            case "4": {
                getConcesionario().ListarClientesInfo();
                return "s";
            }
            case "0": {
                return "n";
            }
        }return "";
    }
}
```

Concesionario:

```
package Tienda;

import java.io.*;
import java.util.ArrayList;

public class Concesionario implements Serializable {
    private static Concesionario tienda;
    private ArrayList<Seccion> Secciones;
    private ArrayList<Cliente> Clientes;
    private  ArrayList<Venta> Ventas;

    private Concesionario() {
        Secciones = new ArrayList();
        Clientes = new ArrayList();
        Ventas = new ArrayList();
    }

    /**
     * Obtiene la instancia única del concesionario, implementando
el patrón Singleton.
     *
    * Este método garantiza que solo exista una única instancia de
la clase {@code Concesionario}.
     * Si la instancia no está inicializada:
     * <ul>
     *   <li>Intenta deserializar la instancia desde el archivo
"BackUp.dat".</li>
     *   <li>Si la deserialización falla o el archivo no existe,
crea una nueva instancia de {@code Concesionario}.</li>
     * </ul>
     *
    * @return la instancia única de {@code Concesionario}.
     */
    public static Concesionario getInstance() {
        if(tienda == null) {
            tienda = deserialize("BackUp.dat");
            if (tienda == null) {
                tienda = new Concesionario();
            }
        }
```

```
        return tienda;
    }

    /**
     * Deserializa un objeto desde un archivo especificado.
     *
    * Este método intenta leer y deserializar un objeto genérico
desde un archivo
    * proporcionado como argumento. Si ocurre un error durante la
deserialización,
    * se captura la excepción y se imprime un mensaje de error.
     *
    * @param <A> el tipo de objeto que se deserializará.
     * @param nombreFichero el nombre del archivo desde el cual se
deserializará el objeto.
     * @return el objeto deserializado si la operación es exitosa, o
{@code null} si ocurre un error.
     */
    private static <A> A deserialize(String nombreFichero) {
        try {
            FileInputStream fis = new
FileInputStream(nombreFichero);
            ObjectInputStream iis = new ObjectInputStream(fis);
            return (A) iis.readObject();
        } catch (FileNotFoundException e) {
            System.out.println("El archivo " + nombreFichero + " no
existe. Se creará uno nuevo.");
        } catch (IOException e) {
            System.err.println("Problema al leer el archivo: " + e);
        } catch (ClassNotFoundException e) {
            System.out.println("Clase no encontrada durante la
deserialización: " + e.getMessage());
        }
        return null;
    }

    //Implementaremos las cosas de Clientes
    /**
     * Añade un cliente a la lista de clientes del concesionario.
     *
    * Este método agrega el cliente proporcionado como argumento a
la colección
```

```
     * interna de clientes del concesionario.
     *
     * @param c el objeto {@code Cliente} que se añadirá a la lista
de clientes.
     */
    public void anadirClientes(Cliente c){
        Clientes.add(c);
    }

    /**
     * Recupera un cliente de la lista de clientes basado en su
identificador único.
     *
     * Este método busca en la lista de clientes y devuelve el
cliente cuyo ID coincide
     * con el ID proporcionado. Si no se encuentra ningún cliente con
ese ID, devuelve {@code null}.
     *
     * @param id el identificador único del cliente que se desea
recuperar.
     * @return el objeto {@code Cliente} correspondiente al ID, o
{@code null} si no se encuentra.
     */
   public Cliente recuperarClientes(String id){
        for(Cliente c: Clientes) {
         if(c.getID().equals(id)) {
             return c;
         }
        }
        return null;
    }

    /**
     * Lista todos los clientes registrados en el concesionario.
     *
     * Este método recorre la colección de clientes y muestra la
información de cada cliente,
     * numerándolos de manera consecutiva. Para cada cliente, llama
al método {@code MostrarInfo()}
     * para detallar su información.
     */
    public void ListarClientes(){
```

```
        int i = 1;
        for (Cliente c: Clientes) {
            System.out.println(i + " - ");
            c.MostrarInfo();
            i = i + 1;
        }
    }

    /**
     * Lista todos los clientes que desean recibir información
adicional.
     *
    * Este método recorre la colección de clientes y muestra la
información de aquellos
     * que tienen configurada la preferencia para recibir
información. Numerando de manera consecutiva,
    * llama al método {@code MostrarInfo()} de cada cliente que
cumple con la condición.
     */
    public void ListarClientesInfo(){
        int i = 1;
        for (Cliente c: Clientes) {
            if (c.getInfo()){
                System.out.println(i + ".- ");
                c.MostrarInfo();
                i = i + 1;
            }
        }
    }

    //Desde aqui se implementaran los metodos que tienen que ver con
las Secciones
    /**
     * Agrega una sección a la lista de secciones del concesionario.
     *
    * Este método añade el objeto {@code Seccion} proporcionado a la
colección interna
    * de secciones del concesionario.
     *
    * @param s el objeto {@code Seccion} que se añadirá a la lista
de secciones.
     */
```

```java
    public void agregarSeccion(Seccion s){
        Secciones.add(s);
    }

    /**
     * Recupera una sección de la lista de secciones basada en su identificador único.
     *
    * Este método busca en la lista de secciones y devuelve la sección cuyo ID coincide
    * con el ID proporcionado. Si no se encuentra ninguna sección con ese ID, devuelve {@code null}.
     *
    * @param id el identificador único de la sección que se desea recuperar.
     * @return el objeto {@code Seccion} correspondiente al ID, o {@code null} si no se encuentra.
     */
    public Seccion recuperarSeccion(String id){
        for(Seccion s: Secciones) {
            if(s.getId().equals(id)) {
                return s;
            }
        }return null;
    }

    /**
     * Elimina una sección de la lista de secciones del concesionario.
     *
    * Este método elimina el objeto {@code Seccion} proporcionado de la colección interna
    * de secciones del concesionario. Si la sección no existe en la lista, no se realiza ninguna acción.
     *
    * @param s el objeto {@code Seccion} que se desea eliminar de la lista de secciones.
     */
    public void eliminarSeccion(Seccion s){
        Secciones.remove(s);
    }
```

```java
    /**
     * Lista todas las secciones registradas en el concesionario.
     *
    * Este método recorre la colección de secciones y muestra la
información de cada sección,
    * numerándolas de manera consecutiva. Para cada sección, llama
al método {@code MostrarInfo()}
    * para detallar su información.
     */
    public void ListarSecciones(){
        int i = 1;
        for (Seccion s: Secciones) {
            System.out.println(i + " - ");
            s.MostrarInfo();
            i += 1;
        }
    }

    //Parte de la seccion a ventas
    public void anadirVentas(Venta v){
        Ventas.add(v);
    }

    public int numeroVentas() {
        return this.Ventas.size();
    }
    public String matriculaCoche() {
        Venta v = Ventas.get(this.numeroVentas()-1);
        return v.getMatricula();
    }

    public Venta recuperarVenta(String id) {
        for (Venta v : Ventas) {
            if (id.equals(v.getIdVenta())) {
                return v;
            }
        }
        return null;
    }

    public void mostarVentas() {
        for (Venta v: Ventas){
```

```
            v.mostrarInformacion();
            System.out.println(" ");
        }
    }
}
```

Seccion:

```
package Tienda;

import java.io.Serializable;
import java.util.ArrayList;
public class Seccion implements Serializable {
    //Atributos
    private String id; //Nombre que se le pone a la seccion
    private String descripcion;//Breve descripcion de la seccion
    private ArrayList<Coche> coches;

    //Constructor
    public Seccion(String id, String descripcion) {
        this.id = id;
        this.descripcion = descripcion;
        coches = new ArrayList<>();
    }

    /**
     * Obtiene el identificador único de la sección.
     *
    * Este método devuelve el valor de la variable {@code id}, que
representa
    * el identificador único asociado a la sección.
     *
    * @return una cadena que contiene el identificador único de la
sección.
     */
    public String getId() {return id;}

    /**
     * Agrega un coche a la lista de coches de la sección.
     *
    * Este método añade el objeto {@code Coche} proporcionado a la
colección interna
```

```java
     * de coches pertenecientes a la sección.
     *
     * @param coche el objeto {@code Coche} que se añadirá a la lista
de coches de la sección.
     */
    public void agregarCoche(Coche coche) {
        coches.add(coche);
    }

    /**
     * Obtiene la lista de coches de la sección.
     *
     * Este método devuelve la colección de objetos {@code Coche}
asociados a la sección.
     *
     * @return una lista de coches ({@code ArrayList<Coche>})
pertenecientes a la sección.
     */
    public ArrayList<Coche> getCoches() {return coches;}

    /**
     * Muestra la información detallada de la sección.
     *
     * Este método imprime en la consola los valores de los atributos
de la sección,
     * incluyendo su identificador único ({@code id}) y su
descripción.
     */
    public void MostrarInfo(){
        System.out.println("ID: "+id);
        System.out.println("Descripcion: "+descripcion);
    }

    /**
     * Recupera un coche de la lista de coches de la sección basado
en su identificador único.
     *
     * Este método busca en la lista de coches de la sección y
devuelve el coche cuyo ID coincide
     * con el ID proporcionado. Si no se encuentra ningún coche con
ese ID, devuelve {@code null}.
     *
```

```
    * @param id el identificador único del coche que se desea
recuperar.
     * @return el objeto {@code Coche} correspondiente al ID, o
{@code null} si no se encuentra.
     */
    public Coche RecuperarCoche(String id){
        for (Coche coche : coches) {
            if(coche.getId().equals(id)){
                return coche;
            }
        }return  null;
    }

    /**
     * Lista todos los coches registrados en la sección.
     *
    * Este método recorre la lista de coches de la sección y muestra
la información de cada coche,
    * numerándolos de manera consecutiva. Para cada coche, llama al
método {@code mostrarInfo()}
    * para detallar su información.
     */
    public void ListarCoches(){
        int i = 1;
        for (Coche coche : coches) {
            System.out.println(i + "- ");
            coche.mostrarInfo();
            i = i + 1;
        }
    }
}
```

Coche:

```
package Tienda;

import java.io.Serializable;
public abstract class Coche  {

    //Atributos
    private String id; // Es el modelo + el año de fabricacion.
    private double precioBase;
    private int stock;
```

```java
    /**
     * Constructor de la clase Coche.
     *
     * @param id Identificador del coche
     * @param precioBase Precio base del coche.
     *
     * @throws IllegalArgumentException Si el modelo es nulo o
vacío, el año es inválido,
     *                                  el precio base es negativo o
el stock es negativo.
     */
    public Coche(String id, double precioBase) {
        try {
            if (id == null || id.isEmpty()  || precioBase < 0  ) {
                throw new IllegalArgumentException("Parámetros
inválidos para crear un coche.");
            }
            this.id =id;
            this.precioBase = precioBase;
            this.stock = 1;
        }catch(IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }

    //Metodos

    public void mostrarInfo() {
        System.out.println("ID: " + id);
        System.out.println("Precio: " + calcularPrecioFinal());
        System.out.println("Stock: " + stock);
    }

    /**
     * Obtiene el identificador único del coche.
     *
     * @return El identificador único del coche.
     */
    public String getId() {
```

```java
        return id;
    }

    /**
     * Obtiene el precio base del coche.
     *
     * @return El precio base del coche.
     */
    public double getPrecioBase() {
        return precioBase;
    }

    /**
     * Obtiene la cantidad de coches disponibles en stock.
     *
     * @return La cantidad de coches en stock.
     */
    public int getStock() {
        return stock;
    }

    /**
     * Actualiza la cantidad de coches disponibles en stock.
     *
     * @param stock La nueva cantidad de coches en stock (mínimo 0).
     */
    public void setStock(int stock) {
        try {
            if (stock < 0) {
                throw new IllegalArgumentException("El stock no
puede ser negativo.");
            }

            this.stock = stock;

        }catch(IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
    /**
     * Calcula el precio final del coche, incluyendo posibles
mejoras.
```

```
     * Este método debe ser implementado por las subclases.
     *
     * @return El precio final del coche.
     */
    public abstract double calcularPrecioFinal();


}
```

CocheBase:

```
package Tienda;

public class Cochebase extends Coche{

    public Cochebase(String id, double precio_base) {
        super(id, precio_base);
    }

    @Override
    public double calcularPrecioFinal() {
        return super.getPrecioBase();
    }


}
```

CocheDecorador:

```
package Tienda;

import java.lang.reflect.Constructor;

/*
Clase abstracta para decorar un coche con mejoras.*/
public abstract class CocheDecorador extends Coche {
    protected Coche c;

    /*

    Constructor del decorador de coches.*
```

```
    @param c El coche que será decorado con mejoras.**/
    public CocheDecorador(Coche c) {
        super(c.getId(), c.getPrecioBase());
        this.c = c;

    }

    @Override
    public double calcularPrecioFinal() {
        return c.calcularPrecioFinal();
    }

    @Override
    public void mostrarInfo() {
        c.mostrarInfo();
    }

    public double costeAdicional(double num){
        return num - c.getPrecioBase();
    }


}
```

GPS:

```
package Tienda;

/*
Decorador para añadir un sistema GPS a un coche.*/
public class GPS extends CocheDecorador {
    private static final double PORCENTAJE_INCREMENTO = 0.01;

    /*
    Constructor de la mejora GPS.*
    @param cocheDecorado El coche que será decorado con esta mejora.
    @throws IllegalArgumentException Si el cocheDecorado es nulo.*/
    public GPS(Coche cocheDecorado) {
        super(cocheDecorado);}

    @Override
```

```
    public double calcularPrecioFinal() {
        return super.calcularPrecioFinal() * (1 +
PORCENTAJE_INCREMENTO);
    }

    @Override
    public void mostrarInfo() {
        super.mostrarInfo();
        System.out.println("Mejora Aplicada: GPS" );
        System.out.println("Coste Adicional: "+ costeAdicional());
    }


    public double costeAdicional() {
        return super.costeAdicional( calcularPrecioFinal());
    }
}
```

Calefaccion:

```
package Tienda;

/**
Decorador para añadir calefacción en los asientos a un coche.*/
public class Calefaccion extends CocheDecorador {
    private static final double PORCENTAJE_INCREMENTO = 0.1;

    /**
    Constructor de la mejora Calefacción en los Asientos.*
    @param cocheDecorado El coche que será decorado con esta mejora.
    @throws IllegalArgumentException Si el cocheDecorado es nulo.*/
    public Calefaccion(Coche cocheDecorado) {
        super(cocheDecorado);}

    @Override
    public double calcularPrecioFinal() {
        return super.calcularPrecioFinal() * (1 +
PORCENTAJE_INCREMENTO);
    }
```

```
    @Override
    public void mostrarInfo() {
        super.mostrarInfo();
        System.out.println("Mejora Aplicada : Calefaccion" );
        System.out.println("Coste Adicional: "+ costeAdicional());
    }

    public double costeAdicional() {
        return super.costeAdicional(calcularPrecioFinal());
    }


}
```

Llantas:

```
package Tienda;

/**
Decorador para añadir llantas de aleación a un coche.**/
public class Llantas extends CocheDecorador {
    private static final double PORCENTAJE_INCREMENTO = 0.08;

    /**
    Constructor de la mejora Llantas de Aleación.*
    @param cocheDecorado El coche que será decorado con esta mejora.
    @throws IllegalArgumentException Si el cocheDecorado es nulo.*/
    public Llantas(Coche cocheDecorado) {
        super(cocheDecorado);}

    @Override
    public double calcularPrecioFinal() {
        return super.calcularPrecioFinal() * (1 +
PORCENTAJE_INCREMENTO);
    }

    @Override
    public void mostrarInfo() {
        super.mostrarInfo();
        System.out.println("Mejora Aplicada : LLantas" );
        System.out.println("Coste Adicional: "+ costeAdicional());
    }
```

```
    public double costeAdicional() {
        return super.costeAdicional( calcularPrecioFinal());
    }
}
```

Tapiceria:

```
package Tienda;

/**
Decorador para añadir tapicería de cuero a un coche.*/
public class Tapiceria_Cuero extends CocheDecorador {
    private static final double PORCENTAJE_INCREMENTO = 0.10;

    /**
    Constructor de la mejora Tapicería de Cuero.*
    @param cocheDecorado El coche que será decorado con esta mejora.
    @throws IllegalArgumentException Si el cocheDecorado es nulo.*/
    public Tapiceria_Cuero(Coche cocheDecorado) {
        super(cocheDecorado);}

    @Override
    public double calcularPrecioFinal() {
        return c.calcularPrecioFinal() * (1 +
PORCENTAJE_INCREMENTO);
    }

    @Override
    public void mostrarInfo() {
        super.mostrarInfo();
        System.out.println("Mejora Aplicada : Tapiceria de Cuero" );
        System.out.println("Coste Adicional: "+ costeAdicional());
    }


    public double costeAdicional() {
        return super.costeAdicional( calcularPrecioFinal());
    }
}
```

Cliente:

```
package Tienda;

import java.io.Serializable;
public class Cliente implements Serializable {
    private String ID; //DNI o NIE
    private String Nombre;
    private int Telefono;
    private boolean Info;
    public Cliente(String ID, String Nombre, int Telefono, boolean
Info) {
        this.ID = ID;
        this.Nombre = Nombre;
        this.Telefono = Telefono;
        this.Info = Info;
    }

    /**
     * Obtiene el identificador único del objeto.
     *
    * Este método devuelve el valor de la variable {@code ID}, que
representa
    * el identificador único asociado al objeto.
     *
    * @return una cadena que contiene el identificador único del
objeto.
     */
    public String getID(){
        return ID;
    }

   /**
     * Muestra la información detallada del objeto.
     *
    * Este método imprime en la consola los valores de los atributos
del objeto,
    * incluyendo el identificador único, el nombre, el teléfono y la
preferencia de información.
     *
```

```
    * Atributos mostrados:
     * <ul>
     *   <li>ID: Identificador único del objeto.</li>
     *   <li>Nombre: Nombre asociado al objeto.</li>
     *   <li>Telefono: Número de teléfono asociado al objeto.</li>
     *   <li>Info: Preferencia sobre recibir información.</li>
     * </ul>
     */
    public void MostrarInfo(){
        System.out.println("ID: " + ID);
        System.out.println("Nombre: " + Nombre);
        System.out.println("Telefono: " + Telefono);
        System.out.println("Info: " + Info);
    }

    /**
     * Obtiene la preferencia de información asociada al objeto.
     *
    * Este método devuelve el valor de la variable {@code Info}, que
indica si
    * el objeto está configurado para recibir información.
     *
    * @return {@code true} si el objeto está configurado para
recibir información,
    *         {@code false} en caso contrario.
     */
    public boolean getInfo(){
        return Info;
    }

    public String getNombre(){return Nombre;}
}
```

Venta:

```
package Tienda;

import javax.swing.plaf.synth.SynthTextAreaUI;
import java.io.Serializable;
import java.time.LocalDate;
```

```java
/**
 * Representa una venta realizada en el concesionario.
 */
public class Venta implements Serializable {
    private String idVenta;
    private Cliente cliente;
    private LocalDate fechaVenta;
    private String matricula;
    private Coche coche;
    private double precioFinal;

    /**
     * Constructor de la clase Venta.
     *
     * @param idVenta Identificador único de la venta (9
caracteres).
     * @param cliente Cliente asociado a la venta.
     * @param coche Coche vendido.
     * @throws IllegalArgumentException Si algún parámetro es nulo o
inválido.
     */
    public Venta(String idVenta, Cliente cliente, Coche coche,String
matricula, LocalDate fechaVenta) {

        try {
            if ( cliente == null || coche == null ) {
                throw new IllegalArgumentException("Parámetros
inválidos para crear la venta.");
            }
            this.idVenta = idVenta;
            this.cliente = cliente;
            this.fechaVenta = LocalDate.now();
            this.coche =  coche;
            this.matricula = matricula;
            this.fechaVenta = fechaVenta;

            this.precioFinal = calcularPrecioFinalConIVA();
        }catch(IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
```

```java
    /**
     * Calcula el precio final del coche incluyendo el IVA.
     *
     * @return Precio final con IVA.
     */
    private double calcularPrecioFinalConIVA() {
        return coche.calcularPrecioFinal() * 1.21; // IVA del 21%
    }

    /**
     * Obtiene el identificador de la venta.
     *
     * @return El identificador de la venta.
     */
    public String getIdVenta() {
        return idVenta;
    }

    /**
     * Obtiene el cliente asociado a la venta.
     *
     * @return El cliente de la venta.
     */
    public Cliente getCliente() {
        return cliente;
    }

    /**
     * Obtiene la fecha de la venta.
     *
     * @return La fecha de la venta.
     */
    public LocalDate getFechaVenta() {
        return fechaVenta;
    }

    /**
     * Obtiene el coche vendido.
     *
     * @return El coche vendido.
     */
```

```
    public Coche getCoche() {
        return coche;
    }

    public String getMatricula() {
        return matricula;
    }

    /**
     * Obtiene el precio final de la venta con IVA incluido.
     *
     * @return El precio final de la venta.
     */
    public double getPrecioFinal() {
        return precioFinal;
    }

    public void mostrarInformacion(){
        System.out.println("ID: " + idVenta);
        System.out.println("Nombre completo del cliente : " +
cliente.getNombre());
        System.out.println("Fecha dela venta: " + fechaVenta);
        System.out.println("Info del coche: ");
        coche.mostrarInfo();
        System.out.println("Precio final: " +
calcularPrecioFinalConIVA());
    }

}
```

Matricula:

```
package Tienda;

public class Matricula{

    //Atributos
    private  int numeroActual=0 ; // Contador de números
    private static int indiceLetras = 0; // Contador de letras
    private static final String LETRAS_PERMITIDAS =
"BCDFGHJKLMNPRSTVWXYZ";
```

```
    public Matricula(int numeroActual){
        this.numeroActual=numeroActual;
    }

    // Metodo para generar una matrícula secuencial
    public String generarMatricula() {
        // Formatear los números
        String formatoNumeros = String.format("%04d", numeroActual);

        // Generar las letras correspondientes al índice actual
        int primeraLetra = (indiceLetras / (21 * 21)) % 21;
        int segundaLetra = (indiceLetras / 21) % 21;
        int terceraLetra = indiceLetras % 21;

        String letras = "" + LETRAS_PERMITIDAS.charAt(primeraLetra)
+
                LETRAS_PERMITIDAS.charAt(segundaLetra) +
                LETRAS_PERMITIDAS.charAt(terceraLetra);

        // Incrementar el número y, si se llega al máximo, reiniciar
y avanzar el índice de letras
        numeroActual++;
        if (numeroActual > 9999) {
            numeroActual = 0;
            indiceLetras++;
        }

        return formatoNumeros + " " + letras;
    }
}
```

MAIN:

```
package Tienda;

import Entradas.MyInPut;
import Menus.MenuPrincipal;
```

```java
public class Main {
    public static void main(String[] args) {
        Concesionario tienda = Concesionario.getInstance();
        MenuPrincipal menu = new MenuPrincipal(tienda);
        menu.ejecutar();
        MyInPut.serialize(tienda, "BackUp.dat");
    }
}
```