CLEAN CODE II

Entornos de Desarrollo

IES Alonso de Avellaneda
DAM 1 diurno

Martín Tadeo, María

Contenido

Código Original

Main

El código a continuación es del cual partirá nuestra práctica Clean Code II coincidiendo con la finalización de la primera parte de este ya una vez implementados los cambios necesarios y las buenas prácticas de programación vistas en clase. Partiendo de este punto continuaremos con estas buenas prácticas que explicaremos de forma detallada aplicándolo a este caso concreto.

```
package cleancodeev2;

public class CleanCodeEv2 {
    public static void main(String[] args) {
        //Creación de un Objeto Coche vacío
        Coche cocheVacio = new Coche();
        //Creación de Objetos Coche con atributos predeterminados
        Coche primerCoche = new Coche("BMW", "negro", 430.47f, 0.84f, 6.5f);
        Coche segundoCoche = new Coche("Mercedes", "gris", 150.95f, 1.02f, 6.5f);
        Coche tercerCoche = new Coche("VW", "blanco", 93.62f, 0.65f, 6.5f);
        Coche cuartoCoche = new Coche("Fiat", "rojo", 694.22f, 0.58f, 6.5f);

        //Impresión de los atributos de cada Objeto coche y el coste del viaje completo de cada uno
            //Primer Orbjeto coche
        cocheVacio.imprimirCoche(primerCoche);
        System.out.printf("Gasto total del viaje: %19.2f \n", cocheVacio.gastosGasolinaViaje(primerCoche));
            //Segundo Orbjeto coche
        cocheVacio.imprimirCoche(segundoCoche);
        System.out.printf("Gasto total del viaje: %19.2f \n", cocheVacio.gastosGasolinaViaje(segundoCoche));
            //Tercer Orbjeto coche
        cocheVacio.imprimirCoche(tercerCoche);
        System.out.printf("Gasto total del viaje: %19.2f \n", cocheVacio.gastosGasolinaViaje(tercerCoche));
            //Cuarto Orbjeto coche
        cocheVacio.imprimirCoche(cuartoCoche);
        System.out.printf("Gasto total del viaje: %19.2f \n", cocheVacio.gastosGasolinaViaje(cuartoCoche));

    }
}
```

Clase Coche

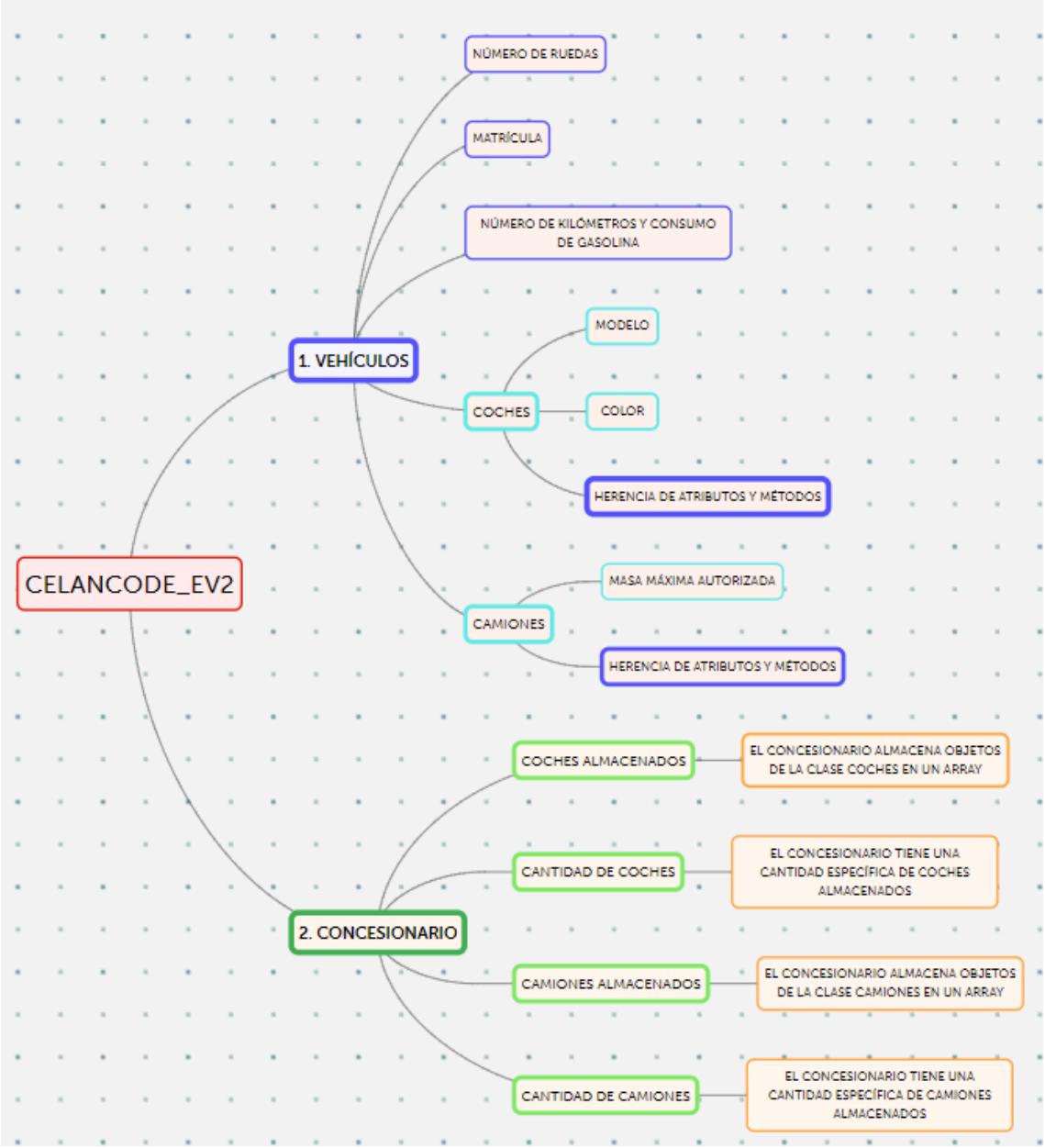
```
package cleancodeev2;

public class Coche {
    //Atributos de la clase
    String modelo;
    String color;
    float nroKilometros;
    float precioGasolina;
    float consumo;

    //Cosntructor de la clase
    public Coche(String modelo, String color, float nroKilometros, float precioGasolina, float consumo) {
        this.modelo = modelo;
        this.color = color;
        this.nroKilometros = nroKilometros;
        this.precioGasolina = precioGasolina;
        this.consumo = consumo;
    }
    //Cosntructor vacío de la clase
    public Coche(){
        modelo = "";
        color = "";
        nroKilometros = 0f;
        precioGasolina = 0f;
        consumo = 0f;
    }

    //Getter y Setter de los atributos de la clase
    public String getModelo() {
        return modelo;
    }
```

```java
public void setModelo(String modelo) {
    this.modelo = modelo;
}
public String getColor() {
    return color;
}
public void setColor(String color) {
    this.color = color;
}
public float getNroKilometros() {
    return nroKilometros;
}
public void setNroKilometros(float nroKilometros) {
    this.nroKilometros = nroKilometros;
}
public float getPrecioGasolina() {
    return precioGasolina;
}
public void setPrecioGasolina(float precioGasolina) {
    this.precioGasolina = precioGasolina;
}
public float getConsumo() {
    return consumo;
}
public void setConsumo(float consumo) {
    this.consumo = consumo;
}

@Override
public String toString() {
    return "Coche{" + "modelo=" + modelo + ", color=" + color + ", nroKilometros=" + nroKilometros +
            ", precioGasolina=" + precioGasolina + ", consumo=" + consumo + '}';
}
```

```java
//Método para calcular el coste total de la gasolina de un coche
public double gastosGasolinaViaje(Coche vehiculo){
    float costeTotal= (vehiculo.nroKilometros * 2 / vehiculo.consumo) * vehiculo.precioGasolina;
    return costeTotal;
}

//Método para imprimir los atributos de un Objeto de la clase
public void imprimirCoche(Coche coche){
    System.out.printf("CARACTERISTICAS: \n\tModelo: %26s \n\tColor: %27s \n\tNumero de Kilometros ida: %8s \n\tPrecio combustible: %14s \n"
            , coche.modelo, coche.color, String.valueOf(coche.nroKilometros), String.valueOf(coche.precioGasolina));
}
```

En esta práctica ampliaremos considerablemente y editaremos el programa anterior para poder tratar los bloques del Clean Code correspondientes a esta segunda parte.

Para tener una visión más clara de cómo estará compuesto nuestro programa hemos realizado un diagrama esquemático donde se pueden ver las clases que existen (`Vehiculos()` y `Concesionario()`), las clases hijas (`Coches()` y `Camiones()`) y además los atributos de cada una de ellas. En todas ellas aplicaremos las técnicas de buena práctica en el código vistas en la práctica uno y las correspondientes a esta.

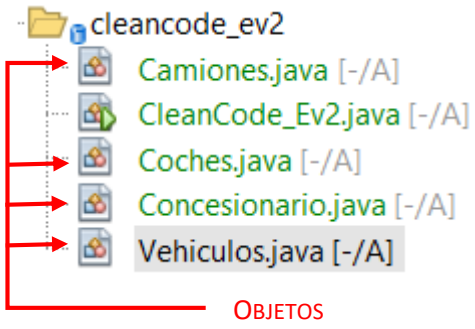


Bloque 4: Objetos y estructuras de datos

Diferencias entre objetos y estructuras de datos

Una estructura de datos se trata de un elemento sencillo que únicamente contiene información y expone sus datos pero sin tener funciones lógicas, mientras que un objeto es una abstracción que expone su funcionalidad para operar con los datos que contiene. A los objetos es más habitual pedirles que hagan cosas y a las estructuras que nos devuelvan datos. Los objetos definen comportamientos mientas que las estructuras de datos almacenan estado.

Veremos de forma práctica como se aplica a nuestro proyecto:




```java
public class Concesionario {
    //Atributos de la clase
    private Coches[] coches;
    private int cantidadCoches;
    private Camiones[] camiones;
    private int cantidadCamiones;
```

ESTRUCTURAS DE DATOS

Nuestro proyecto está formado por cuatro objetos con sus correspondientes atributos y métodos definidos dentro de ella. En la clase Concesionario podemos ver que en ella hay estructuras de datos siendo estos arrays de objetos; es decir es una estructura de dato que almacena objetos.

La Ley de Demeter

Es un mecanismo de detección de acoplamiento; es decir, debemos intentar que nuestros objetos no conozcan el interior de otros objetos con los que tiene interacción. Esta ley se cumple cuando la función comprometida es llamada por la propia clase en la que está definida, un objeto es creado a partir de esa función, un objeto es pasado como argumento a la función o un objeto está almacenado en el campo del objeto donde está definida la función. Un indicio de que esto no se cumple es el hecho de ver muchas llamadas concatenadas.

A continuación, trataremos varias formas de poder solventar este problema y asi cumplir con esta ley.

1. **Añadir métodos extra:** En este punto se trata de añadir nuevos métodos a los objetos implicados en los que se hará la subllamada correspondiente. No es muy recomendable, pero hay ocasiones en las que es necesario.
2. **Arquitectura:** La elaboración de una arquitectura adecuada nos ayuda para reducir significativamente la posibilidad de no cumplir con esta ley. Una buena arquitectura se caracteriza por tener varias interfaces con las que se comunica cada capa de nuestro programa y al tener la implementación oculta evitamos problemas de este tipo.

```java
public class Vehiculos {
    //Atributos de la clase
    private int nroRuedas;
    private String matricula;
    private float nroKilometros;
    private float precioGasolina;
    private float consumo;
```

CLASE VEHÍCULOS

```java
public class Coches extends Vehiculos{
    //Atributos de la clase
    private String modelo;
    private String color;
```

CLASE COCHES

```java
public class Concesionario {
    //Atributos de la clase
    private Coches[] coches;
    private int cantidadCoches;
    private Camiones[] camiones;
    private int cantidadCamiones;
```

CLASE CONCESIONARIOS

```java
public class Camiones extends Vehiculos {
    //Atributos de la clase
    private int masaMaxAutorizada;
```

Clase CAMIONES

La encapsulación es una de las mejores técnicas para ocultar los atributos y de los métodos de una clase para hacerlos únicamente accesibles por la propia clase y las que heredan de ella. Todos nuestros atributos deben ser privados y su accesibilidad será a través de los getters y los setters.

GETTER Y SETTERS

```java
//Getter y Setter de los atributos de la clase
public String getModelo() {
    return modelo;
}
public void setModelo(String modelo) {
    this.modelo = modelo;
}
public String getColor() {
    return color;
}
public void setColor(String color) {
    this.color = color;
}
```

CLASE COCHES

```java
//Método para calcular el coste total de la gasolina de un coche
public double gastosGasolinaViaje(){
    double costeTotal= (this.getNroKilometros() * 2 / this.getConsumo()) * this.getPrecioGasolina();
    return costeTotal;
}
```

METODO GASTOSGASOLINAVIAJE() DE LA CLASE VEHICULOS

Las imágenes previas son dos ejemplos de código de nuestro proyecto donde se ve claramente su aplicación, pero esto se da en todas y cada una de nuestras clases, la definición de los getters y los setters para su uso en los métodos definidos dentro de las clases.

Bloque 5: Manejo de errores

El tratamiento de errores nos permite poder tratar todos los casos posibles en los que nuestro programa puede fallar gracias a las excepciones y evitar así un posible colapso al no haber tenido en cuenta ciertos puntos concretos de vulnerabilidad. La estructura de las excepciones es definir claramente el try-catch-finally y lo hemos aplicado a todos aquellos métodos en los que era necesario tener en cuenta este manejo de errores para así poder controlarlo y evitar el posible colapso de nuestro programa.

MÉTODOS DE LAS CLASE CONCESIONARIO

```java
/**
 * Método que añade un objeto Coche en el array
 * @param car a añadir
 */
public void agregarCoche(Coches car) {
    try {
        if (cantidadCoches < coches.length && car.comprobarMatricula()) {
            coches[cantidadCoches] = car;
            cantidadCoches++;
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
```

```java
/**
 * Método que añade un objeto Coche en el array
 * @param truck a añadir
 */
public void agregarCamion(Camiones truck) {
    try {
        if (cantidadCamiones < camiones.length && truck.comprobarMatricula() == true) {
            camiones[cantidadCamiones] = truck;
            cantidadCamiones++;
        } else {
            System.out.println("Matrícula no válida");
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
```

```java
/**
 * Método que borrar un objeto coche del array. Objeto identificado por la matricula
 * @param matricula del estudiante
 */
public void borrarCoche(String matricula) {
    try {
        for (int i = 0; i < cantidadCoches; i++) {
            if (coches[i].getMatricula().equals(matricula)) {
                // Mover los elementos siguientes hacia atrás
                System.arraycopy(coches, i + 1, coches, i, cantidadCoches - i - 1);
                coches[cantidadCoches - 1] = null; // Limpiar el último elemento
                cantidadCoches--;
                break;
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
```

```java
/**
 * Método que borrar un objeto camion del array. Objeto identificado por la matricula
 * @param matricula del camion
 */
public void borrarCamion(String matricula) {
    try {
        for (int i = 0; i < cantidadCamiones; i++) {
            if (camiones[i].getMatricula().equals(matricula)) {
                // Mover los elementos siguientes hacia atrás
                System.arraycopy(camiones, i + 1, camiones, i, cantidadCamiones - i - 1);
                camiones[cantidadCamiones - 1] = null; // Limpiar el último elemento
                cantidadCamiones--;
                break;
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
```

Bloque 6: Test unitarios

El uso de tests unitarios nos ayudan a saber si nuestras funciones y métodos definidos cumplen realmente con lo que queremos que hagan y así poder elaborarlos con mayor precisión para ajustarse aún más con lo deseado. Es importante que se mantengan limpios para una mayor comprensión y si fuese necesario modificarlos o añadir nuevos poder hacerlos sin dificultad.

Hay que tener muy presente el uso de un único *assert* por test ya que nos verificará correctamente si algo falla o no ya que al unir varios en un mismo test sería mucho más tedioso saber con certeza el motivo al igual que únicamente deben cubrir y comprobar una única cosa.

Para poder ver esto claramente trataremos varios tests en nuestro programa para las clases Vehiculos y Concesionario creando un test por cada uno de los métodos definidos en las clases.

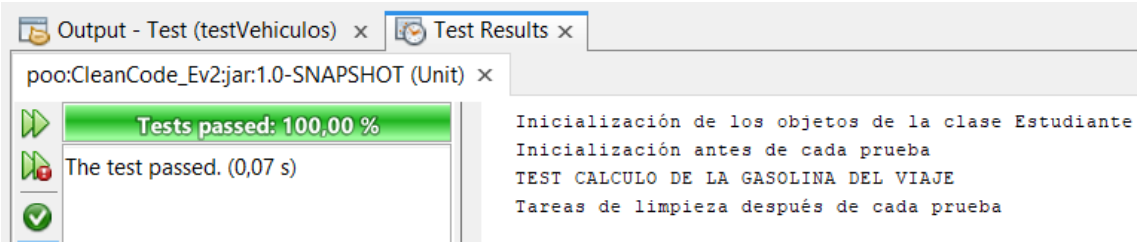
Test clase Vehiculos

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.DisplayName;
import poo.cleancode_ev2.Vehiculos;


public class testVehiculos {
    private static Vehiculos nuevo0;
    private static Vehiculos nuevo1;
    private static Vehiculos nuevo2;

    @BeforeAll
    //inicialización de los objetos de la clase Vehiculo
    public static void setUpAll() {
        nuevo0 = new Vehiculos(4, "1234ABC", 430.47f, 0.84f, 6.5f);
        nuevo1 = new Vehiculos(4, "2589PLK", 150.95f, 1.02f, 7.6f);
        nuevo2 = new Vehiculos(10,"3690FGR",93.62f, 0.65f, 6.5f);
        System.out.println("Inicialización de los objetos de la clase Estudiante");
    }

    @BeforeEach
    public void setUp() {
        System.out.println("Inicialización antes de cada prueba");
    }

    @AfterEach
    public void tearDown() {
        System.out.println("Tareas de limpieza después de cada prueba");
    }

    @AfterAll
    public static void tearDownAll() {
        nuevo0 = null;
        nuevo1 = null;
        nuevo2 = null;
        System.out.println("Final de las pruebas JUnit");
    }

}
```

testCalculoGasolinaViaje

Agrupación en un asserAll() de varios assertEquals()

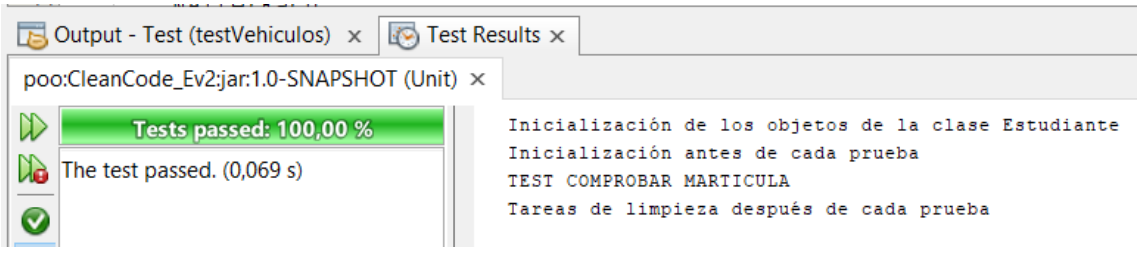
```
@Test
@DisplayName ("Test de verificar que se calcula correctamente el gasto de la gasolina del viaje")
public void testCalculoGasolinaViaje() {
    System.out.println("TEST CALCULO DE LA GASOLINA DEL VIAJE");
    assertAll(
        () -> assertEquals(111.26 , nuevo0.gastosGasolinaViaje(), 0.1),
        () -> assertEquals(40.52 , nuevo1.gastosGasolinaViaje(), 0.1),
        () -> assertEquals(18.72 , nuevo2.gastosGasolinaViaje(), 0.1)
    );
}
```



testComprobarMatricula

Agrupación en un asserAll() de varios assertEquals()

```
@Test
@DisplayName ("Test de verificar que se introducen bien las matriculas de los vehículos")
public void testComprobarMatricula() {
    System.out.println("TEST COMPROBAR MARTICULA");
    Vehiculos nuevo3 = new Vehiculos(18, "D1111CV", 694.22f, 0.58f, 6.5f);
    assertAll(
            () -> assertTrue(nuevo0.comprobarMatricula()),
            () -> assertTrue(nuevo1.comprobarMatricula()),
            () -> assertTrue(nuevo2.comprobarMatricula()),
            () -> assertFalse(nuevo3.comprobarMatricula())
    );
}
```



Test clase Concesionario

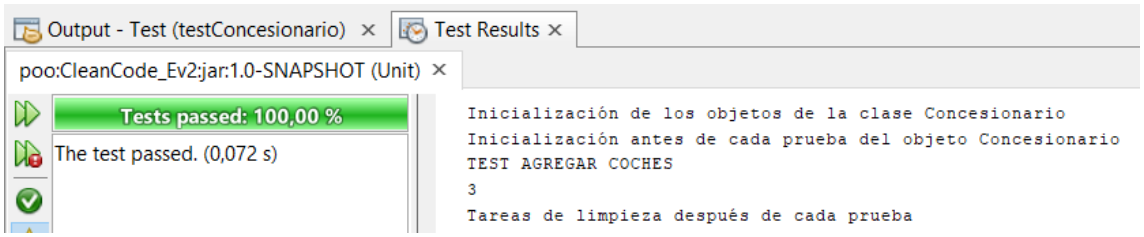
```java
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
//import java.util.Arrays;
import poo.cleancode_ev2.Camiones;
import poo.cleancode_ev2.Coches;
import poo.cleancode_ev2.Concesionario;
//import poo.cleancode_ev2.Vehiculos;

public class testConcesionario {
    //creación del objeto concesionario, el array resCoches,resCamiones y los objetos coches y camiones
    private static Concesionario concesionario;
    private static Coches[] coches;
    private static Camiones[] camiones;
    private static Coches[] resCoches;
    private static Camiones[] resCamiones;
```

```java
    @BeforeAll
    //Inicialización de los objetos de la clase Concesionario
    public static void setUpAll() {
        coches = new Coches[]{
            new Coches(4, "1234ABC", 430.47f, 0.84f, 6.5f, "BMW", "negro"),
            new Coches(4, "2589PLK", 150.95f, 1.02f, 7.6f, "Mercedes", "gris")
        };
        camiones = new Camiones[]{
            new Camiones(10,"3690FGR",93.62f, 0.65f, 6.5f, 7500),
            new Camiones(18, "1568SDC", 694.22f, 0.58f, 6.5f, 9800)
        };
        concesionario = new Concesionario(5,5);
        resCoches = concesionario.obtenerCoches();
        resCamiones = concesionario.obtenerCamiones();

        System.out.println("Inicialización de los objetos de la clase Concesionario");
    }

    @BeforeEach
    public void setUp() {
        concesionario = new Concesionario(5,5);
        //agregar al concesionario todos los Coches
        for (Coches car : coches) {
            concesionario.agregarCoche(car);
        }
        //agregar al Concesionario todos los Camiones
        for (Camiones truck : camiones) {
            concesionario.agregarCamion(truck);
        }

        System.out.println("Inicialización antes de cada prueba del objeto Concesionario");
    }
```

```
    @AfterEach
    public void tearDown() {
        System.out.println("Tareas de limpieza después de cada prueba");
    }

    @AfterAll
    //todos los valores a null
    public static void tearDownAll() {
        concesionario = null;
        coches = null;
        camiones = null;
        resCoches = null;
        resCamiones = null;
        System.out.println("Final de las pruebas JUnit");
    }
}
```

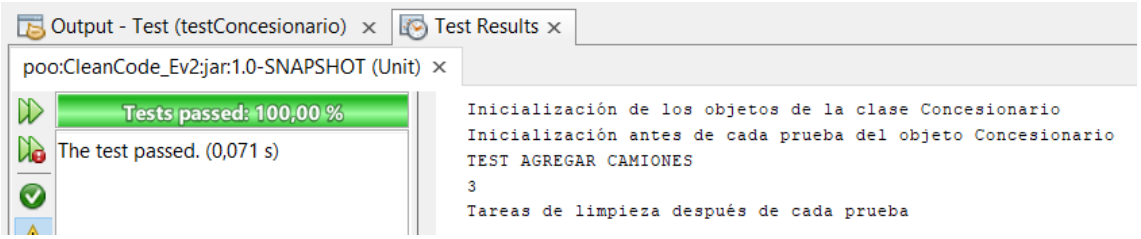
testAgregarCoche

```
@Test
@DisplayName ("Test de verificar que se agrega un objeto coche al array del concerionario coches")
public void testAgregarCoche(){
    System.out.println("TEST AGREGAR COCHES");
    //crear un estudiante nuevo
    Coches cocheNuevo = new Coches(4, "4669CED", 723.9f, 0.94f, 6.8f, "VW", "blanco");
    //agregar al gestor el estudiante nuevo
    concesionario.agregarCoche(cocheNuevo);
    //Obtener el array resultado tras agregar un Estudiante
    resCoches = concesionario.obtenerCoches();
    //crear el array esperado
    Coches[] arrayEsperado = new Coches[]{
        new Coches(4, "1234ABC", 430.47f, 0.84f, 6.5f, "BMW", "negro"),
        new Coches(4, "2589PLK", 150.95f, 1.02f, 7.6f, "Mercedes", "gris"),
        cocheNuevo
    };
    //comparar arrays (assertArrayEquals)
    assertArrayEquals(arrayEsperado,resCoches);
    System.out.println(resCoches.length);
}
```



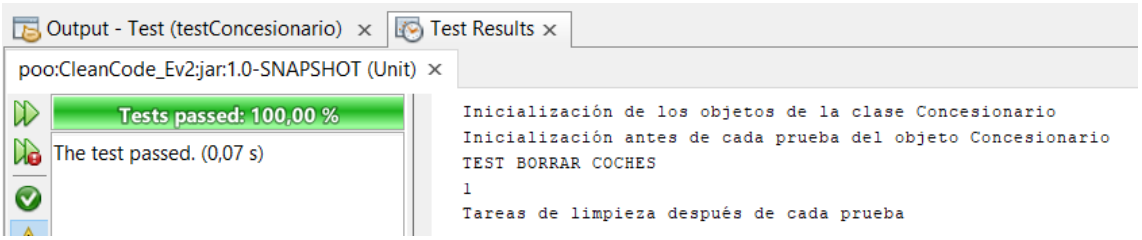
testAgregarCamion

```
@Test
@DisplayName ("Test de verificar que se agrega un objeto camion al array del concerionario camiones")
public void testAgregarCamion(){
    System.out.println("TEST AGREGAR CAMIONES");
    //crear un estudiante nuevo
    Camiones camionNuevo = new Camiones(18, "1568SDC", 694.22f, 0.58f, 6.5f, 9800);
    //agregar al gestor el estudiante nuevo
    concesionario.agregarCamion(camionNuevo);
    //Obtener el array resultado tras agregar un Estudiante
    resCamiones = concesionario.obtenerCamiones();
    //crear el array esperado
    Camiones[] arrayEsperado = new Camiones[]{
        new Camiones(10,"3690FGR",93.62f, 0.65f, 6.5f, 7500),
        new Camiones(18, "1568SDC", 694.22f, 0.58f, 6.5f, 9800),
        camionNuevo
    };
    //comparar arrays (assertArrayEquals)
    assertArrayEquals(arrayEsperado, resCamiones);
    System.out.println(resCamiones.length);
}
```



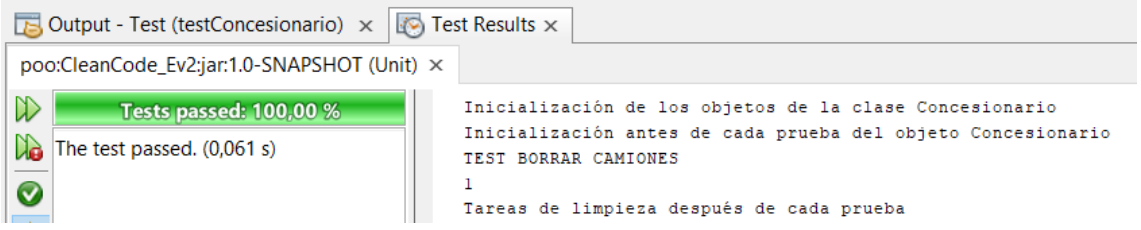
testBorrarCoche

```
@Test
@DisplayName ("Test de verificar que se borra un objeto coche del array del concerionario coches")
public void testBorrarCoche(){
    System.out.println("TEST BORRAR COCHES");
    //borrar estudiante
    concesionario.borrarCoche("1234ABC");
    //obtener el array resultado tras borrar el estudiante
    resCoches = concesionario.obtenerCoches();
    //crear el array esperado
    Coches[] arrayEsperado = new Coches[]{
        new Coches(4, "2589PLK", 150.95f, 1.02f, 7.6f, "Mercedes", "gris"),
    };
    //comparar arrays (assertArrayEquals)
    assertArrayEquals(arrayEsperado, resCoches);
    System.out.println(resCoches.length);
}
```



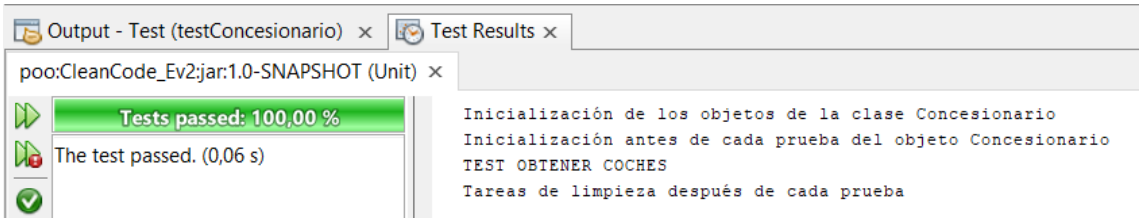
testBorrarCamion

```
@Test
@DisplayName ("Test de verificar que se borra un objeto camion del array del concerionario camiones")
public void testBorrarCamion(){
    System.out.println("TEST BORRAR CAMIONES");
    //borrar estudiante
    concesionario.borrarCamion("1568SDC");
    //obtener el array resultado tras borrar el estudiante
    resCamiones = concesionario.obtenerCamiones();
    //crear el array esperado
    Camiones[] arrayEsperado = new Camiones[]{
        new Camiones(10,"3690FGR",93.62f, 0.65f, 6.5f, 7500)
    };
    //comparar arrays (assertArrayEquals)
    assertArrayEquals(arrayEsperado, resCamiones);
    System.out.println(resCamiones.length);
}
```



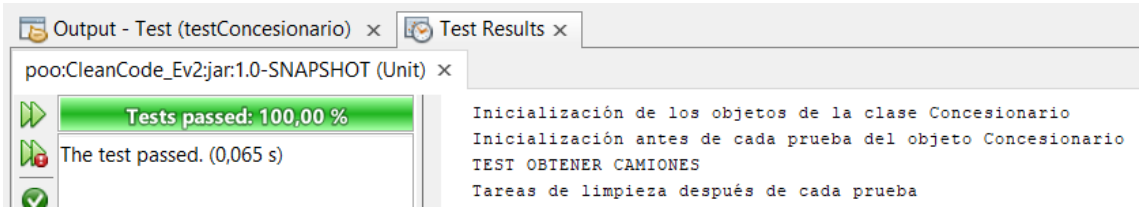
testObtenerCoche

```
@Test
@DisplayName ("Test de verificar que se obtiene el array con los objetos coches del concesionario")
public void testObtenerCoche(){
    System.out.println("TEST OBTENER COCHES");
    resCoches = concesionario.obtenerCoches();
    Coches[] arrayEsperado = new Coches[]{
        new Coches(4, "1234ABC", 430.47f, 0.84f, 6.5f, "BMW", "negro"),
        new Coches(4, "2589PLK", 150.95f, 1.02f, 7.6f, "Mercedes", "gris")
    };
    assertArrayEquals(arrayEsperado, resCoches);
}
```



testObtenerCamion

```
@Test
@DisplayName ("Test de verificar que se obtiene el array con los objetos camiones del concesionario")
public void testObtenerCamion(){
    System.out.println("TEST OBTENER CAMIONES");
    resCamiones = concesionario.obtenerCamiones();
    Camiones[] arrayEsperado = new Camiones[]{
        new Camiones(10,"3690FGR",93.62f, 0.65f, 6.5f, 7500),
        new Camiones(18, "1568SDC", 694.22f, 0.58f, 6.5f, 9800)
    };
    assertArrayEquals(arrayEsperado, resCamiones);
}
```



Bloque 7: Clases

Como hemos visto a lo largo de toda la práctica y como se comentó al inicio de ella mediante un gráfico disponemos de cuatro clases y dos de ellas heredan de la misma por lo que se da una jerarquía de clases. Cada una tiene sus propios atributos definidos y encapsulados y sus métodos. En el caso de las clases hijas aparte de heredar los atributos de la clase padre tienes los suyos propios junto con los métodos heredados sobrescritos.

Clase Vehículo

Primero encontramos la definición de los atributos de la clase junto con su constructor y el constructor vacío.

```
package poo.cleancode_ev2;

public class Vehiculos {
    //Atributos de la clase
    private int nroRuedas;
    private String matricula;
    private float nroKilometros;
    private float precioGasolina;
    private float consumo;

    //Constructor
    public Vehiculos(int nroRuedas, String matricula, float nroKilometros, float precioGasolina, float consumo) {
        this.nroRuedas = nroRuedas;
        this.matricula = matricula;
        this.nroKilometros = nroKilometros;
        this.precioGasolina = precioGasolina;
        this.consumo = consumo;
    }
    //Constructor vacio
    public Vehiculos(){
        nroRuedas = 0;
        matricula = "";
        nroKilometros = 0;
        precioGasolina = 0;
        consumo = 0;
    }
```

Seguidos de ellos están las declaraciones de todos los getters y setter de los atributos (encapsulación)

```
    public int getNroRuedas() {
        return nroRuedas;
    }
    public void setNroRuedas(int nroRuedas) {
        this.nroRuedas = nroRuedas;
    }
    public String getMatricula() {
        return matricula;
    }
    public void setPeso(String matricula) {
        this.matricula = matricula;
    }
    public float getNroKilometros() {
        return nroKilometros;
    }
    public void setNroKilometros(float nroKilometros) {
        this.nroKilometros = nroKilometros;
    }

    public float getPrecioGasolina() {
        return precioGasolina;
    }
    public void setPrecioGasolina(float precioGasolina) {
        this.precioGasolina = precioGasolina;
    }
    public float getConsumo() {
        return consumo;
    }
    public void setConsumo(float consumo) {
        this.consumo = consumo;
    }
```

El método toString es definido por defecto por el NetBeans. Los métodos hasCode e equals han sido necesarios sobrescribirlos para adaptarlos a este programa y asi poder hacer uso de ellos en los test con las aserciones.

```java
@Override
public String toString() {
    return "Vehiculos{" + "nroRuedas=" + this.getNroRuedas() + ", matricula=" + this.getMatricula() + ", nroKilometros=" + this.getNroKilometros() +
            ", precioGasolina=" + this.getPrecioGasolina() + ", consumo=" + this.getConsumo() + '}';
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((this.getMatricula() == null) ? 0 : this.getMatricula().hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Vehiculos other = (Vehiculos) obj;
    if (this.getMatricula() == null) {
        if (other.getMatricula() != null)
            return false;
    } else if (!this.getMatricula().equals(other.getMatricula()))
        return false;
    return true;
}
```

Por último se encuentran los métodos propios de la clase que serán heredados por sus clases hijas en las que no es necesario definirlos de nuevo a no ser que se quiera aplicar algún cambio en él.

```java
//Método para calcular el coste total de la gasolina de un coche
public double gastosGasolinaViaje(){
    double costeTotal= (this.getNroKilometros() * 2 / this.getConsumo()) * this.getPrecioGasolina();
    return costeTotal;
}

//Método para imprimir los atributos de un Objeto de la clase
public void imprimirVehiculo(){
    System.out.printf("CARACTERISTICAS: \n\tNumero de ruedas: %16d \n\tMatricula: %23s \n\tNumero de Kilometros ida: %8.3f "
            + "\n\tPrecio combustible: %14.2f \n\tConsumo: %25.2f"
            , this.getNroRuedas(), this.getMatricula(), this.getNroKilometros(), this.getPrecioGasolina(), this.getConsumo());
}

//Método para comprobar que la matricula es correcta
public boolean comprobarMatricula(){
    return this.getMatricula().toUpperCase().matches("^[0-9]{4}[A-Z]{3}$"); //expresiones regulares: max 4 num. del 0 al 9 y max 3 letras de la A a la Z
}
```

Clase Coches

Declaración de los atributos, el constructor y el constructor vacío. Como esta clase es hija de la clase Vehiculos en el constructor debe hacerse referencia a aquellos atributos que hereda de la clase padre con `super`.

```java
package poo.cleancode_ev2;

public class Coches extends Vehiculos{
    //Atributos de la clase
    private String modelo;
    private String color;

    //Cosntructor de la clase
    public Coches(int nroRuedas, String matricula,float nroKilometros, float precioGasolina, float consumo, String modelo, String color) {
        super(nroRuedas, matricula, nroKilometros, precioGasolina, consumo);
        this.modelo = modelo;
        this.color = color;
    }
    //Cosntructor vacio de la clase
    public Coches(){
        super();
        modelo = "";
        color = "";
    }
```

Únicamente se deben definir los getters y setter de los atributos propios de esta clase, pero no de los atributos heredados ya que eso se ha hecho previamente en la clase padre.

```
    //Getter y Setter de los atributos de la clase
    public String getModelo() {
        return modelo;
    }
    public void setModelo(String modelo) {
        this.modelo = modelo;
    }
    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
```

El método creado por defecto por NetBeans no incluye el de la clase padre por lo que lo incluimos con `super.toString()`.

```
    @Override
    public String toString() {
        return super.toString() + "Coches{" + "modelo=" + this.getModelo() + ", color=" + this.getColor() + '}';
    }
```

Este método heredado de la clase padre puede ser modificado por la clase hijo como hemos comentado anteriormente indicándolo gracias al `@Override`. En este caso estamos llamamos al método de la clase padre llamado igual y añadimos contenido al de la clase hija

```
    @Override
    //Método para imprimir los atributos de un Objeto de la clase
    public void imprimirVehiculo(){
        super.imprimirVehiculo();
        System.out.printf("\n\tModelo: %26s \n\tColor: %27s \n"
                , this.getModelo(), this.getColor());
    }
}
```

Clase Camiones

Al ser también esta clase hija de Vehiculos su comportamiento es igual que la clase anterior, tanto el acceso a los métodos heredados como a los atributos y la definición de los nuevos.

```
package poo.cleancode_ev2;

public class Camiones extends Vehiculos {
    //Atributos de la clase
    private int masaMaxAutorizada;

    public Camiones(int nroRuedas, String matricula, float nroKilometros, float precioGasolina, float consumo, int masaMaxAutorizada) {
        super(nroRuedas, matricula, nroKilometros, precioGasolina, consumo);
        this.masaMaxAutorizada = masaMaxAutorizada;
    }

    public Camiones(float masaMaxAutorizada) {
        super();
        masaMaxAutorizada = 0;
    }
```

```
    public int getMasaMaxAutorizada() {
        return masaMaxAutorizada;
    }
    public void setMasaMaxAutorizada(int masaMaxAutorizada) {
        this.masaMaxAutorizada = masaMaxAutorizada;
    }
```

```
    @Override
    public String toString() {
        return super.toString() + "Camiones{" + "masaMaxAutorizada=" + this.getMasaMaxAutorizada() + '}';
    }
```

```
    @Override
    //Método para imprimir los atributos de un Objeto de la clase
    public void imprimirVehiculo(){
        super.imprimirVehiculo();
        System.out.printf("\n\tMasa Maxima Autorizada: %10d\n"
                , this.getMasaMaxAutorizada());
    }

}
```

Clase Concesionario

La clase concesionario la curiosidad que se puede destacar de ella es que está formada por arrays de objetos de las clases Coches y Camiones. Su comportamiento es como las demas con sus atributos y métodos propios.

```
package poo.cleancode_ev2;

import java.util.Arrays;

public class Concesionario {
    //Atributos de la clase
    private Coches[] coches;
    private int cantidadCoches;
    private Camiones[] camiones;
    private int cantidadCamiones;

    //Constructor
    public Concesionario(int nroCoches, int norCamiones) {
        this.coches = new Coches[nroCoches];
        this.camiones = new Camiones[norCamiones];
        this.cantidadCoches = 0;
        this.cantidadCamiones = 0;
    }

    //Constructor vacio
    public Concesionario() {
        coches = new Coches[0];
        camiones = new Camiones[0];
        cantidadCoches = 0;
        cantidadCamiones = 0;
    }
```

```java
public Coches[] getCoches() {
    return coches;
}
public void setCoches(Coches[] coches) {
    this.coches = coches;
}

public int getCantidadCoches() {
    return cantidadCoches;
}
public void setCantidadCoches(int cantidadCoches) {
    this.cantidadCoches = cantidadCoches;
}

public Camiones[] getCamiones() {
    return camiones;
}
public void setCamiones(Camiones[] camiones) {
    this.camiones = camiones;
}

public int getCantidadCamiones() {
    return cantidadCamiones;
}
public void setCantidadCamiones(int cantidadCamiones) {
    this.cantidadCamiones = cantidadCamiones;
}
```

```java
@Override
public String toString() {
    return "Concesionario{" + "coches=" + this.getCoches() + ", cantidadCoches=" + this.getCantidadCoches() + ", camiones="
            + this.getCamiones() + ", cantidadCamiones=" + this.getCantidadCamiones() + '}';
}
```

Los métodos de esta clase tienen como objetivo añadir un objeto al array, borrarlo de él y obtener el array con todos sus elementos. Al ser arrays que almacenan objetos diferentes ha sido necesario duplicar estos métodos para cada uno de ellos por lo que se podría corresponder con repetición de código.

```java
// Método que imprime los objetos de los array de Concesionario
public void imprimeConcesionario() {
    System.out.println("VEHICULOS:");
    // Imprimir los coches
    for (Coches car : coches) {
        if (car != null) {
            car.imprimirVehiculo();
        }
    }
    // Imprimir los camiones
    for (Camiones truck : camiones) {
        if (truck != null) {
            truck.imprimirVehiculo();
        }
    }
}
```

```java
/**
 * Método que añade un objeto Coche en el array
 * @param car a añadir
 */
public void agregarCoche(Coches car) {
    try {
        if (cantidadCoches < coches.length && car.comprobarMatricula()) {
            coches[cantidadCoches] = car;
            cantidadCoches++;
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}
```

```java
/**
 * Método que añade un objeto Coche en el array
 * @param truck a añadir
 */
public void agregarCamion(Camiones truck) {
    try {
        if (cantidadCamiones < camiones.length && truck.comprobarMatricula() == true) {
            camiones[cantidadCamiones] = truck;
            cantidadCamiones++;
        } else {
            System.out.println("Matrícula no válida");
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}

/**
 * Método que borrar un objeto coche del array. Objeto identificado por la matricula
 * @param matricula del estudiante
 */
public void borrarCoche(String matricula) {
    try {
        for (int i = 0; i < cantidadCoches; i++) {
            if (coches[i].getMatricula().equals(matricula)) {
                // Mover los elementos siguientes hacia atrás
                System.arraycopy(coches, i + 1, coches, i, cantidadCoches - i - 1);
                coches[cantidadCoches - 1] = null; // Limpiar el último elemento
                cantidadCoches--;
                break;
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
```

```java
/**
 * Método que borrar un objeto camion del array. Objeto identificado por la matricula
 * @param matricula del camion
 */
public void borrarCamion(String matricula) {
    try {
        for (int i = 0; i < cantidadCamiones; i++) {
            if (camiones[i].getMatricula().equals(matricula)) {
                // Mover los elementos siguientes hacia atrás
                System.arraycopy(camiones, i + 1, camiones, i, cantidadCamiones - i - 1);
                camiones[cantidadCamiones - 1] = null; // Limpiar el último elemento
                cantidadCamiones--;
                break;
            }
        }
    } catch (Exception e) {
        System.out.println("Error: " + e.toString());
    }
}

/**
 * Método que devuelve el array de objetos coches
 * @return array de coches
 */
public Coches[] obtenerCoches() {
    return Arrays.copyOf(coches, cantidadCoches);
}
```

```java
/**
 * Método que devuelve el array de objetos camiones
 * @return array de camiones
 */
public Camiones[] obtenerCamiones() {
    return Arrays.copyOf(camiones, cantidadCamiones);
}
```

Código Final: Clean Code

```java
public class CleanCode_Ev2 {
    public static void main(String[] args) {
        //Creacion del objeto Concesionario
        Concesionario concesionario = new Concesionario(2, 2);
        //Creación de Objetos Coches y Camiones con atributos predeterminados
        Coches coche1 = new Coches(4, "1234ABC", 430.47f, 0.84f, 6.5f, "BMW", "negro"); //Primer Orbjeto Vehiculos -> Coches
        Coches coche2 = new Coches(4, "2589PLK", 150.95f, 1.02f, 7.6f, "Mercedes", "gris"); //Segundo Orbjeto Vehiculos -> Coches
        Camiones camion1 = new Camiones(10,"3690FGR",93.62f, 0.65f, 6.5f, 7500); //Tercer Orbjeto Vehiculos -> Camiones
        Camiones camion2 = new Camiones(18, "1568SDC", 694.22f, 0.58f, 6.5f, 9800); //Cuarto Orbjeto Vehiculos -> Camiones

        //Agregar los objetos a los arrays del concesionario
        concesionario.agregarCoche(coche1);
        concesionario.agregarCoche(coche2);
        concesionario.agregarCamion(camion1);
        concesionario.agregarCamion(camion2);
        //Impresion el contenido de los arrays del concesionario
        concesionario.imprimeConcesionario();

        //Borrar objeto de los arrays del concesionario
        concesionario.borrarCoche("1234ABC");
        concesionario.borrarCamion("1568SDC");

        //Impresion el contenido de los arrays del concesionario
        concesionario.imprimeConcesionario();

        //Calculo de los gastos de los viajes
        System.out.printf("Gasto total del viaje: %19.2f \n", coche1.gastosGasolinaViaje());
        System.out.println("-------------------------");
        System.out.printf("Gasto total del viaje: %19.2f \n", coche2.gastosGasolinaViaje());
        System.out.println("-------------------------");
        System.out.printf("Gasto total del viaje: %19.2f \n", camion1.gastosGasolinaViaje());
        System.out.println("-------------------------");
        System.out.printf("Gasto total del viaje: %19.2f \n", camion2.gastosGasolinaViaje());
```