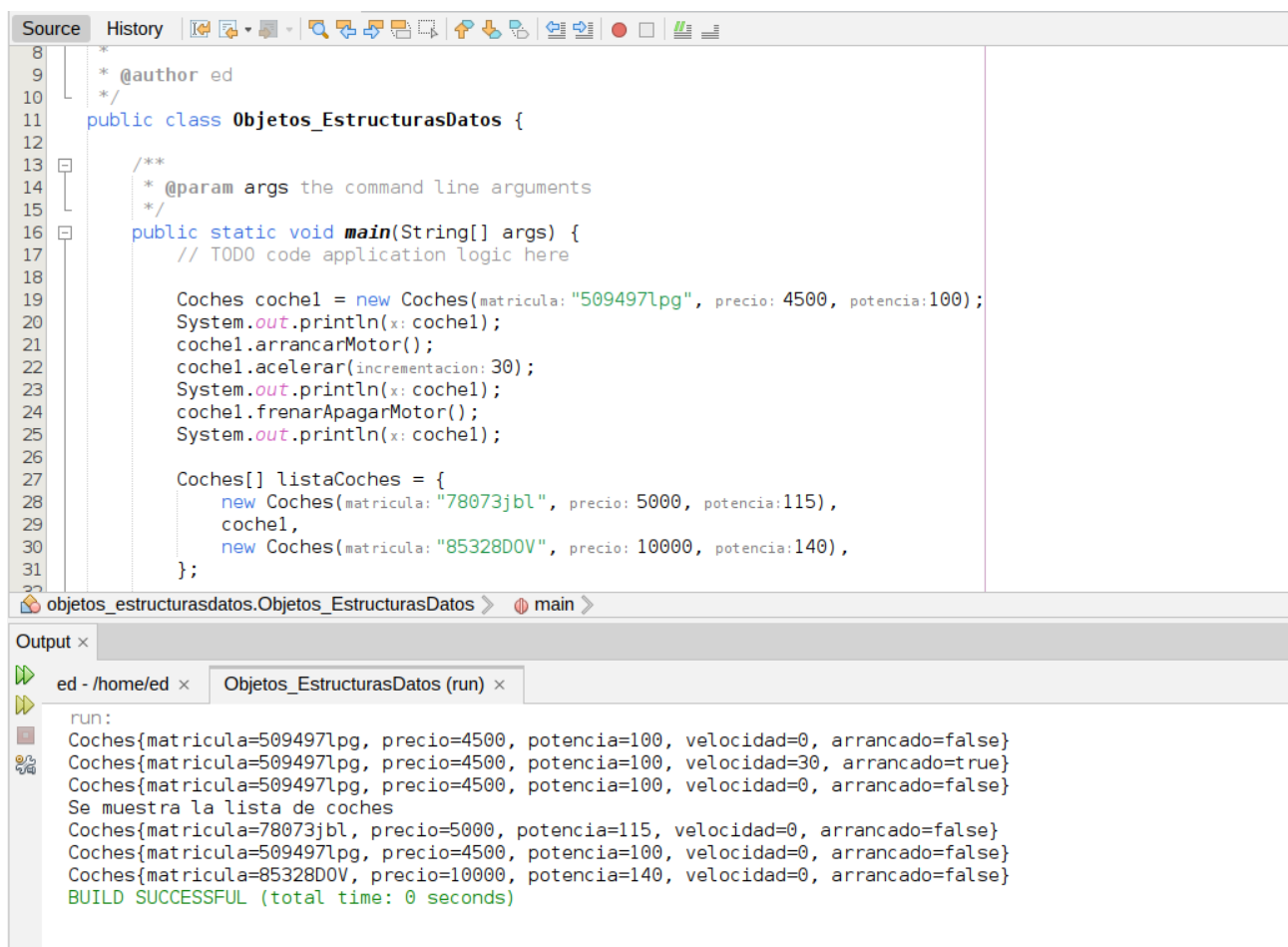


# Clean Code EV2

## Bloque 4. Objetos y estructuras de datos

### Punto 17

En el ejemplo se observa claramente la diferencia entre objetos y estructuras de datos, en este caso, un array. Creamos un objeto de la clase Coches, trabajamos con el y con sus metodos, imprimiendo los cambios realizados por pantalla, hacemos uso de sus funciones lógicas, mientras que el array lo utilizamos simplemente para exponer los datos de los objetos almacenados, no podemos hacer operaciones logicas con el.



The screenshot shows a code editor with a C# program. The code defines a class `Objetos_EstructurasDatos` with a `main` method. In `main`, a `Coches` object is created, its state is modified, and it is added to an array. The array is then printed. The output window shows the execution results, including the state of the `Coches` object before and after modifications, and the list of cars.

```
8  /**
9  * @author ed
10 * */
11 public class Objetos_EstructurasDatos {
12
13     /**
14     * @param args the command line arguments
15     */
16     public static void main(String[] args) {
17         // TODO code application logic here
18
19         Coches coche1 = new Coches(matricula: "509497lpg", precio: 4500, potencia:100);
20         System.out.println(x: coche1);
21         coche1.arrancarMotor();
22         coche1.acelerar(incrementacion: 30);
23         System.out.println(x: coche1);
24         coche1.frenarApagarMotor();
25         System.out.println(x: coche1);
26
27         Coches[] listaCoches = {
28             new Coches(matricula: "78073jbl", precio: 5000, potencia:115),
29             coche1,
30             new Coches(matricula: "85328D0V", precio: 10000, potencia:140),
31         };
32     }
33 }
```

Output:

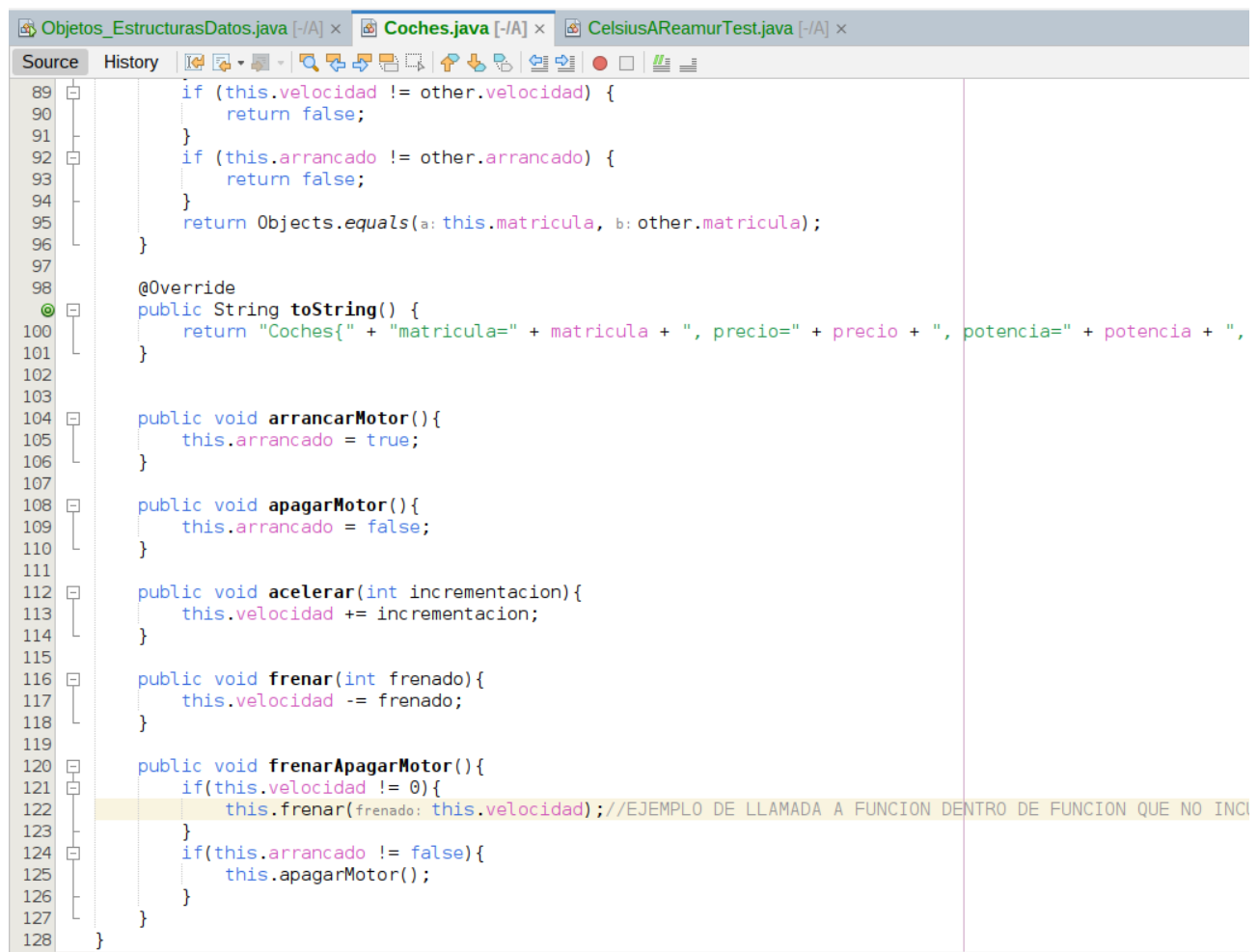
```
run:
Coches{matricula=509497lpg, precio=4500, potencia=100, velocidad=0, arrancado=false}
Coches{matricula=509497lpg, precio=4500, potencia=100, velocidad=30, arrancado=true}
Coches{matricula=509497lpg, precio=4500, potencia=100, velocidad=0, arrancado=false}
Se muestra la lista de coches
Coches{matricula=78073jbl, precio=5000, potencia=115, velocidad=0, arrancado=false}
Coches{matricula=509497lpg, precio=4500, potencia=100, velocidad=0, arrancado=false}
Coches{matricula=85328D0V, precio=10000, potencia=140, velocidad=0, arrancado=false}
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 1: Ejemplo del punto 17

### Punto 18

Este metodo cumpliria la ley de Demeter, que se basa en la premisa de que no se debe conocer las entrañas de objeto con el que interactua. Lo mas importante de la ley de Demeter es no comprometer la funcionalidad del código con la estructura de clases, ya que si esta cambia habra que refactorizar mucho código. Cumple la premisa de teniendo una función f de una clase C, esa

función sólo llama a funciones de : C, un objeto creado por f, un objeto pasado como argumento a f, un objeto almacenado en campo de C.



```
89     if (this.velocidad != other.velocidad) {
90         return false;
91     }
92     if (this.arrancado != other.arrancado) {
93         return false;
94     }
95     return Objects.equals(a: this.matricula, b: other.matricula);
96 }
97
98 @Override
99 public String toString() {
100     return "Coches{" + "matricula=" + matricula + ", precio=" + precio + ", potencia=" + potencia + ",
101 }
102
103
104 public void arrancarMotor(){
105     this.arrancado = true;
106 }
107
108 public void apagarMotor(){
109     this.arrancado = false;
110 }
111
112 public void acelerar(int incrementacion){
113     this.velocidad += incrementacion;
114 }
115
116 public void frenar(int frenado){
117     this.velocidad -= frenado;
118 }
119
120 public void frenarApagarMotor(){
121     if(this.velocidad != 0){
122         this.frenar(frenado: this.velocidad); //EJEMPLO DE LLAMADA A FUNCION DENTRO DE FUNCION QUE NO INCI
123     }
124     if(this.arrancado != false){
125         this.apagarMotor();
126     }
127 }
128 }
```

Figure 2: Ejemplo Ley Demeter

## Bloque 5. Errores

Podemos apreciar que en este ejemplo se utiliza excepciones en lugar de códigos de retorno, se escribe primero el bloque try-catch-finally para manejar las excepciones de manera adecuada, se utilizan excepciones no verificadas (unchecked) para evitar la propagación excesiva de excepciones en el código y no devuelve valores nulos para evitar problemas de NullPointerException.

```
9  * @author ed
10 */
11 public class EjemploExcepciones {
12
13     /**
14     * @param args the command line arguments
15     */
16     public static void main(String[] args) {
17         try {
18             printHelloWorld();
19         } catch (Exception e) {
20             System.out.println("An error occurred: " + e.getMessage());
21         } finally {
22             System.out.println(x: "This will always be executed, regardless of whether an exception was thrown or not.");
23         }
24     }
25
26     public static void printHelloWorld() throws Exception {
27         // Simulamos una situación en la que podríamos lanzar una excepción.
28         if (Math.random() < 0.5) {
29             throw new Exception(message: "An error occurred while printing Hello, World!");
30         }
31         System.out.println(x: "Hello, World!");
32     }
33 }
```

ejemploexcepciones.EjemploExcepciones

Output x

ed - /home/ed x EjemploExcepciones (run) x

run:  
An error occurred: An error occurred while printing Hello, World!  
This will always be executed, regardless of whether an exception was thrown or not.  
BUILD SUCCESSFUL (total time: 0 seconds)

## Bloque 6. Test Unitarios

Generamos los test a partir del código del bloque 4. Se mantienen los tests limpios, siguiendo las reglas de código limpio para que sean legibles y mantenibles. cada test realiza un único Assert para comprobar un único concepto y cumplen con la regla F.I.R.S.T.

Objetos\_EstructurasDatos1.java [-/A] x Coches.java [-/A] x CochesTest.java [-/A] x

Source History

37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62

```
@Test
public void testArrancarMotor() {
    Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia:150);
    coche.arrancarMotor();
    assertTrue(condition: coche.isArrancado());
}

@Test
public void testApagarMotor() {
    Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia:150);
    coche.arrancarMotor();
    coche.apagarMotor();
    assertFalse(condition: coche.isArrancado());
}

@Test
public void testAcelerar() {
    Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia:150);
    coche.arrancarMotor();
    coche.acelerar(incrementacion: 50);
    assertEquals(expected:50, actual: coche.getVelocidad());
}

@Test
public void testFrenar() {
    Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia:150);
```

CochesTest > testFrenar >

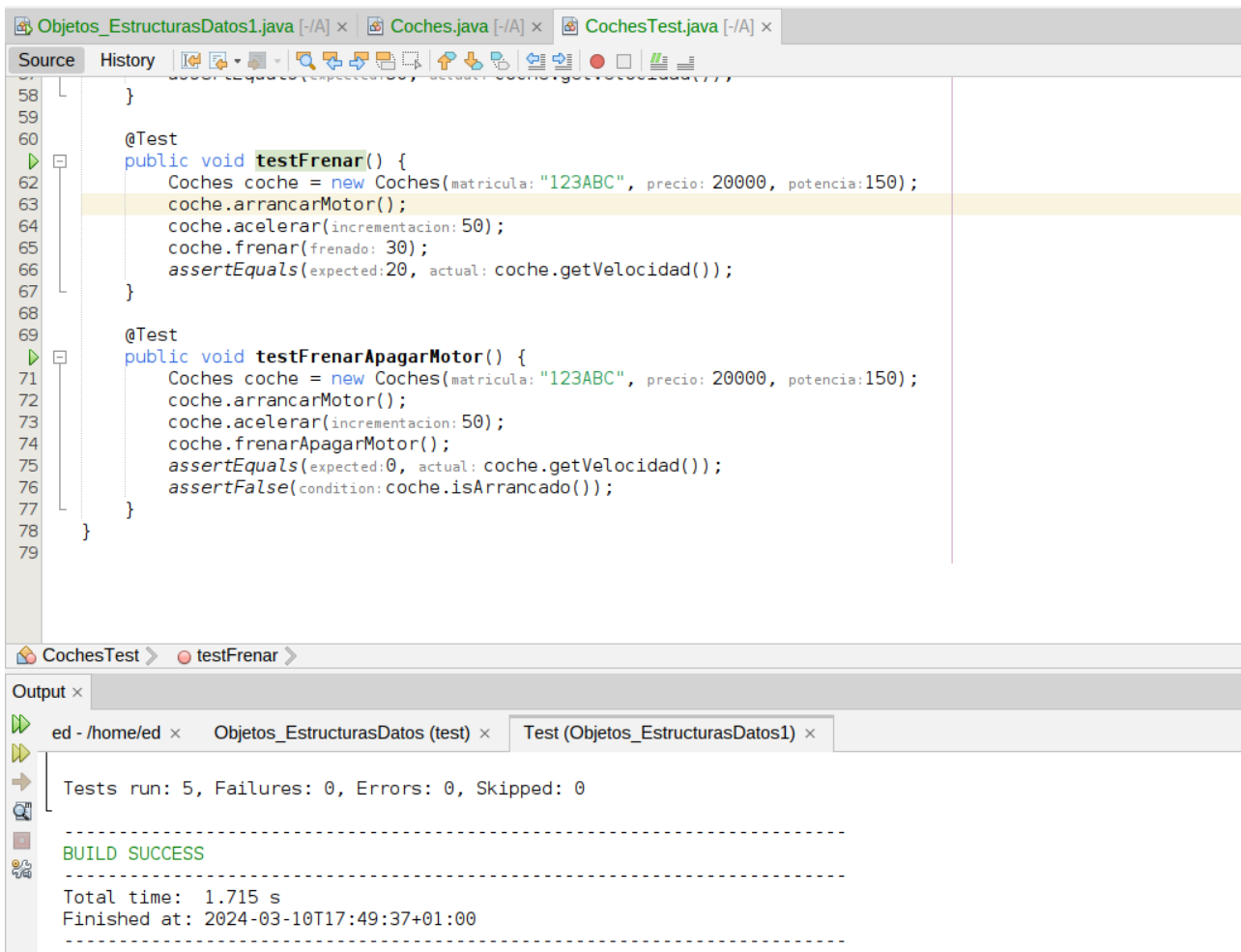
Output x

ed - /home/ed x Objetos\_EstructurasDatos (test) x Test (Objetos\_EstructurasDatos1) x

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESS

Total time: 1.715 s  
Finished at: 2024-03-10T17:49:37+01:00



```
Objetos_EstructurasDatos1.java [-/A] x Coches.java [-/A] x CochesTest.java [-/A] x
Source History
58 }
59
60 @Test
61 public void testFrenar() {
62     Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia: 150);
63     coche.arrancarMotor();
64     coche.acelerar(incrementacion: 50);
65     coche.frenar(frenado: 30);
66     assertEquals(expected: 20, actual: coche.getVelocidad());
67 }
68
69 @Test
70 public void testFrenarApagarMotor() {
71     Coches coche = new Coches(matricula: "123ABC", precio: 20000, potencia: 150);
72     coche.arrancarMotor();
73     coche.acelerar(incrementacion: 50);
74     coche.frenarApagarMotor();
75     assertEquals(expected: 0, actual: coche.getVelocidad());
76     assertFalse(condition: coche.isArrancado());
77 }
78 }
79

CochesTest testFrenar
Output x
ed - /home/ed x Objetos_EstructurasDatos (test) x Test (Objetos_EstructurasDatos1) x
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
-----
BUILD SUCCESS
-----
Total time: 1.715 s
Finished at: 2024-03-10T17:49:37+01:00
-----
```

## Bloque 7.Clases

Hacemos una pequeña modificación en la clase Coches para poder cumplir con todos los puntos de este bloque. Se ha mantenido la cohesión de la clase asegurando que cada método manipule solo una o varias de las variables de instancia, sigue el principio de responsabilidad única, centrándose en la gestión de coches, la construcción y el uso de la clase están separados, y la clase es adecuada para la concurrencia.

Objetos\_EstructurasDatos1.java [-/A] x Coches.java [-/A] x CochesTest.java [-/A] x

Source History

```
10  *
11  * @author ed
12  */
13  public class Coches {
14      // Constantes públicas
15      public static final int VELOCIDAD_MAXIMA = 200;
16
17      // Constantes privadas
18      private static final int VELOCIDAD_MINIMA = 0;
19
20      private String matricula;
21      private int precio;
22      private int potencia;
23      private int velocidad;
24      private boolean arrancado;
25
26      public Coches(String matricula, int precio, int potencia) {
27          this.matricula = matricula;
28          this.precio = precio;
29          this.potencia = potencia;
30          velocidad = 0;
31          arrancado = false;
32      }
33
34      public String getMatricula() {
35          return matricula;
36      }
37  }
```

com.mycompany.objetos\_estructurasdatos1.Coches >

Output x

ed - /home/ed x Objetos\_EstructurasDatos (test) x Test (Objetos\_EstructurasDatos1) x

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

BUILD SUCCESS

Total time: 1.715 s  
Finished at: 2024-03-10T17:49:37+01:00