

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

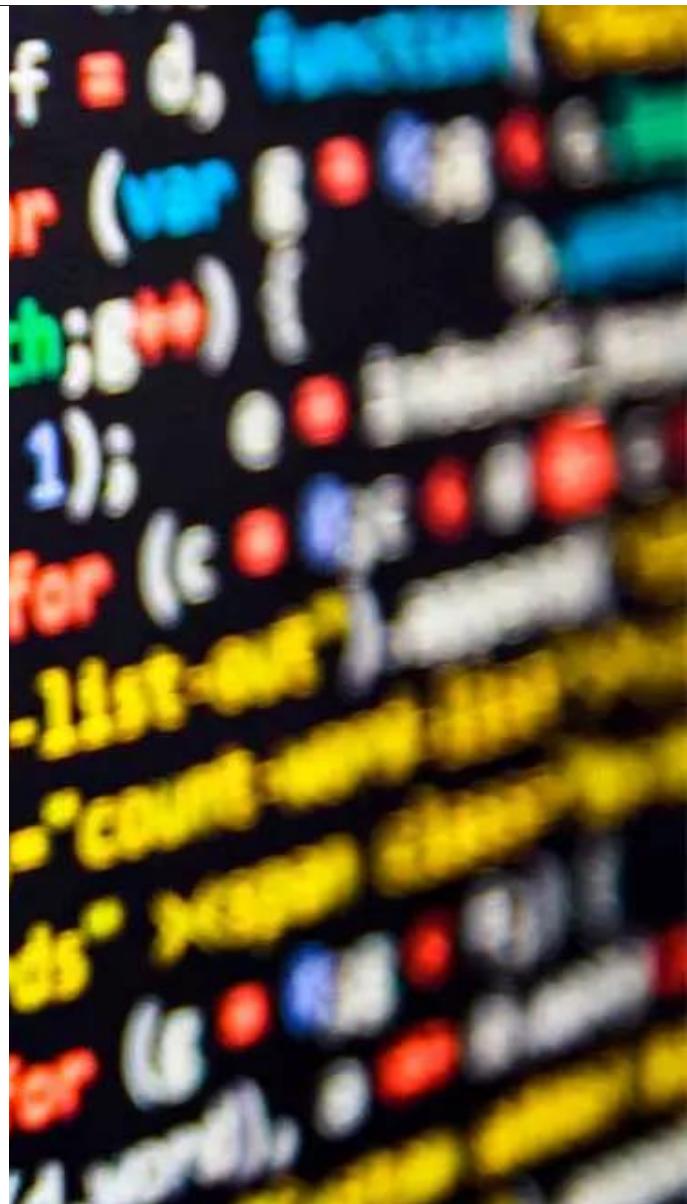
EDER LÓPEZ VILLARREAL

5H T/V

DOCENTE

MTRO. SAUL OLAF
LOAIZA

Portafolio de
evidencia



Fundamentos de POO

Objetivos de cada Practica-Actividades

Práctica 1: Mi primer programa

Crear un proyecto nuevo en el que se realice un código de una clase principal Main y se compruebe su funcionalidad, debe mostrar en pantalla los nombres de los integrantes de equipo y su objetivo: “Aprender a programar en Java, mi primer programa”.

Práctica 2: Subiendo Código a GitHub

Crear un Repositorio en GitHub a través de ciertos pasos para subir nuestro primer código en la plataforma, visualizándose dentro de la carpeta creada para la materia de POO.

Actividad: La historia de UML

Elaboración de una línea del tiempo en la que se describa el contexto de UML a través del tiempo desde su lanzamiento, así como las versiones y el año en el que fue lanzado para los usuarios.

Práctica 3: Obtener datos y conversión de tipos

Crear un programa que realice operaciones con dos números ingresados por el usuario.

Práctica 4: Clase Carro

Crear un programa en donde se realicen los registros de un coche para su mantenimiento, el consumo de gas y sus Km. Recorridos, se deberá de crear su diagrama UML del clasificador del carro, así como en código de la clase carro y la clase principal 3 objetos actualizando a 3 atributos, mostrando la salida en pantalla.

Práctica 5: Clase Carro con sobrecarga

Con el código anterior, realizar el diseño UML de la clase Carro y diseño UML de los objetos de prueba, donde se observe al menos dos tipos de sobrecarga y el conteo de los objetos creados.

Desarrollo clase 1 Clase Temperatura

Objetivo: Realizar el diseño UML de la clase Temperatura y diseño UML de los objetos de prueba, donde se observe al menos dos tipos de sobrecarga. Realizar el diagrama de UML del clasificador de la clase Temperatura (Utilizar dos atributos para identificar la unidad, por ejemplo: tempF y tempC). Realizar el código de la clase Temperatura de acuerdo al diagrama UML de clase. Realizar el diagrama UML de los objetos para verificar la prueba. Realizar en la clase principal con 2 objetos con diferentes inicializaciones y que concuerden con el diagrama de objetos.

Desarrollo clase 1 Clase Restaurante

Realizar el diseño UML de la clase Restaurante y diseño UML de los objetos de prueba. Realizar el diagrama de UML del clasificador de la clase Restaurante (tomar los métodos de acuerdo al análisis). Realizar el código de la clase Restaurante al diagrama UML de clase. Realizar el diagrama UML del objeto para verificar la prueba indicada. Realizar la clase principal el objeto con la prueba solicitada.

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

KAREN LETICIA GARCÍA VÁSQUEZ

EDER LÓPEZ VILLARREAL

JAZMIN PORTILLO MICHICOL

ERIC JHONATHAN ANAYA MÁRQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Practica 1
Mi primer programa

10/01/2024



Código de clase principal Main y prueba de funcionalidad

The screenshot shows a Java development environment with the following details:

- File Explorer:** Shows the project structure with `Main.java` selected.
- Code Editor:** Displays the `Main.java` code:

```
1  /**
2  * UPTx - Fundamentos de programación orientada a objetos
3  * Práctica No. 1     Grupo: 5H
4  * Modificación: 10/ene/2024
5  */
6
7 // Definición de la clase Main
8 public class Main {
9     // Método principal que se ejecuta al correr el programa
10    public static void main(String[] args) {
11        // Imprime el mensaje "Hola Mundo"
12        System.out.println("----- Hola Mundo -----");
13        // Imprime los nombres de los integrantes
14        System.out.println("Integrantes: Eder Lopez Villarreal, Jazmin Portillo Michicol, Eric Jhonathan Anaya Marquez, Karen Garcia Vasquez");
15        // Imprime el objetivo
16        System.out.println("Objetivo: Aprender a programar en Java, Mi primer programa");
17    }
18 }
```
- Run Tab:** Shows the run configuration for `Main`.
- Output Window:** Displays the program's output:

```
----- Hola Mundo -----
Integrantes: Eder Lopez Villarreal, Jazmin Portillo Michicol, Eric Jhonathan Anaya Marquez, Karen Garcia Vasquez
Objetivo: Aprender a programar en Java, Mi primer programa
```
- Bottom Status Bar:** Shows the file path (`Práctica1 > src > Main`), timestamp (`18:2`), and encoding (`LF UTF-8 4 spaces`).

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL

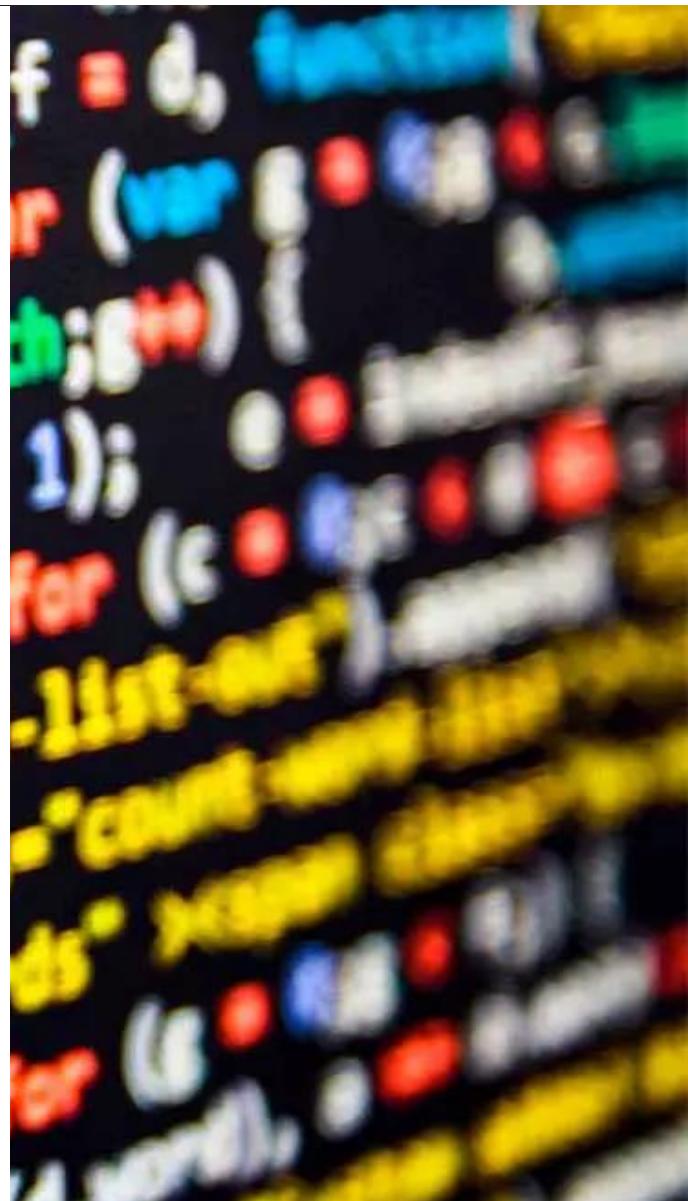
5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Practica 2
Subiendo código a GitHub

11/01/2024



Código de clase principal Main y prueba de funcionalidad

The screenshot shows a Java development environment with the following details:

- File Explorer:** Shows the file `Main.java`.
- Code Editor:** Displays the Java code for the `Main` class, which prints "Hola Mundo" and the names of four students.
- Run Tab:** Shows the output of the program:
 - Hola Mundo -----
 - Integrantes: Eder Lopez Villarreal, Jazmin Portillo Michicol, Eric Jhonathan Anaya Marquez, Karen Garcia Vasquez
 - Objetivo: Aprender a programar en Java, Mi primer programa
- Status Bar:** Shows the file path `Practica1 > src > Main`, and the time `18:2`.

Subida de código a repositorio

The screenshot shows a GitHub repository interface with the following details:

- Repository:** Fundamentos-de-POO/Practica1
- File List:** Shows the file `Practica1.txt`.
- File Content:** Displays the same Java code as the previous screenshot, showing the commit message "EderL04 Update Practica1.txt" by "eae182" made 3 minutes ago.

LINEA DEL TIEMPO DEL CONTEXTO HISTORICO DE UML Y SUS VERSIONES

1997 UML 1.1

En 1997, se lanzó la primera versión oficial del UML, que incluía 9 tipos de diagramas diferentes. Estos diagramas ayudaban a los programadores a visualizar y entender mejor el proceso de desarrollo de software.

Fue adoptado por Object Management Group. Esta fue la primera versión de UML.



1997

1997

1997-OMG UML 1.1(A PARTIR DE ESTA FECHA TODAS LAS VERSIONES SON OMG UML)

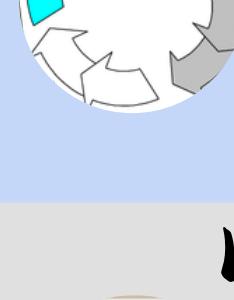


Gracias al trabajo de Booch, Jacobson, Rumbaugh y metodólogos y desarrolladores de varias compañías publican la propuesta (primera versión) para convertirlo en estándar del OMG

El Lenguaje Unificado de Modelado fue adoptado unánimemente por los miembros de OMG como estándar

1998-UML 1.2

Se empezó a trabajar con el estándar UML. En los años sucesivos fueron apareciendo nuevas versiones que introducían mejoras o ampliaban a las anteriores.



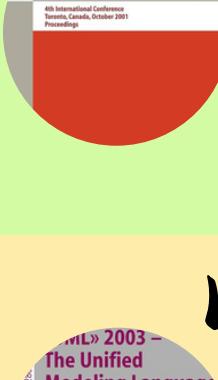
1998

1999



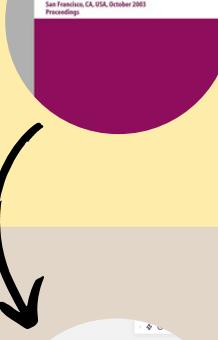
Introdujo mejoras significativas en comparación con las versiones anteriores. Se centró en la integración de los enfoques de modelado y en la mejora de la consistencia y claridad del lenguaje. Ofreció varios tipos de diagramas, incluyendo diagramas de clases, diagramas de secuencia, diagramas de actividad, diagramas de casos de uso, entre otros

2001 La ISO (Organización Internacional de Normalización) es una entidad independiente que desarrolla y publica estándares internacionales para asegurar la calidad, eficiencia y seguridad de productos y servicios en diversas industrias. La historia de la ISO se remonta a principios del siglo XX.



2001

2003



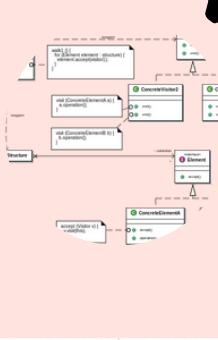
siguió a las ya citadas

En 2005, se lanzó la versión 2.0 del UML, que incluía mejoras en los diagramas existentes y la adición de nuevos diagramas, como el diagrama de comunicación y el diagrama de tiempo.



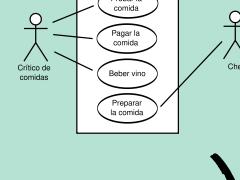
2005

2006



Lanzamiento de UML 2.1 con varias mejoras en la notación y el soporte para modelar aspectos específicos, como el modelado de perfiles y extensiones.

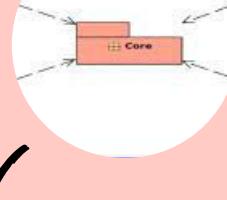
Lanzamiento de actualizaciones menores (2.1.1 y 2.1.2) que generalmente incluyen correcciones de errores y ajustes en la especificación.



2007

Introducción de nuevas características y mejoras, como el soporte mejorado para el modelado de perfiles, mejoras en el soporte para la extensibilidad y la integración con otros estándares.

2009



Continuación de mejoras en la especificación, con posiblemente nuevas características y correcciones de errores.

Si embargo, ten en cuenta que la información detallada sobre cambios específicos en cada versión puede requerir una revisión más detallada de la documentación oficial y las notas de versión.



2010

En 2011, se lanzó oficialmente UML 2.4.1. UML es el Lenguaje Unificado de Modelado, un lenguaje de modelado visual para la arquitectura, el diseño y la implementación de sistemas de software complejos.

2011



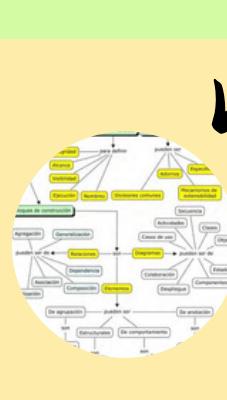
En 2012, no se lanzó una versión oficial de UML. La última versión antes de ese año fue UML 2.4, que se publicó en 2011. En general, el año 2012 no marcó un hito significativo en términos de nuevas versiones de UML o cambios sustanciales en el estándar.



2012

2013

se espera la aprobación del año 2013



REFERENCIAS

LA LÍNEA DEL TIEMPO DE UML. (2023, MARZO 21). LÍNEA DE TIEMPO. [HTTPS://LINEADETIEMPO.NET/LA-LINEA-DEL-TIEMPO-DE-UML/](https://LINEADETIEMPO.NET/LA-LINEA-DEL-TIEMPO-DE-UML/)

SUTORI. (S/F). SUTORI.COM. RECUPERADO EL 13 DE ENERO DE 2024, DE [HTTPS://WWW.SUTORI.COM/ES/HISTORIA/LINEA-DEL-TIEMPO-LENGUAJE-UNIFICADO-DE-MODELADO-UML--CPKUBHBXNWYNFYL9M16C6RRH](https://WWW.SUTORI.COM/ES/HISTORIA/LINEA-DEL-TIEMPO-LENGUAJE-UNIFICADO-DE-MODELADO-UML--CPKUBHBXNWYNFYL9M16C6RRH)

WALKER, A. (2023, DICIEMBRE 29). DIAGRAMAS UML: HISTORIA, TIPOS, CARACTERÍSTICAS, VERSIONES, HERRAMIENTAS. GURU99. [HTTPS://WWW.GURU99.COM/ES/UML-DIAGRAMS.HTML](https://WWW.GURU99.COM/ES/UML-DIAGRAMS.HTML)

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL

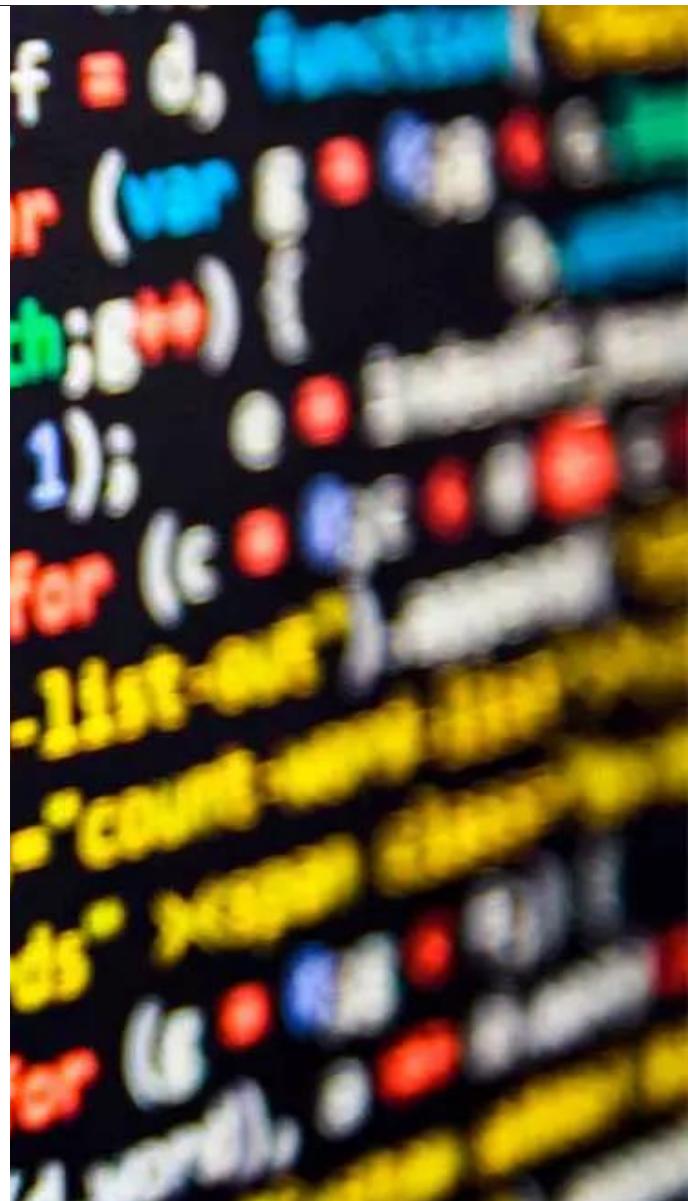
5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Practica 3
Obtener datos y conversión de tipos

13/01/2024



Código de clase principal Main

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Current File, Version control.
- File Explorer:** Shows the project structure with `Main.java` selected.
- Code Editor:** Displays the `Main.java` file content. The code is a simple program that reads two numbers from the user and prints them back. It includes comments explaining the purpose of each section of code.
- Status Bar:** Shows the file size (68.2), line separator (LF), encoding (UTF-8), and indentation (4 spaces).

```
1  /**
2   * UPTx - Fundamentos de Programación Orientada a Objetos
3   * Practica No. 3      Grupo: SH
4   * Tema: Obtener datos y conversión de tipos
5   * Integrantes: Eder López Villarreal
6   * Modificación: 13/ene/2024
7   */
8
9  import java.util.Scanner;
10 public class Main {
11     public static void main(String[] args) {
12         // Declaración de variables
13         int numint;
14         double numdbl, res1, res2, res3, res4, res5, res6;
15
16         //Mensaje de bienvenida
17         System.out.printf("Bienvenidos, favor de seguir las instrucciones\n");
18
19         //Documentar: Creación de un objeto Scanner para leer la entrada del usuario
20         Scanner scanner = new Scanner(System.in);
21
22         //Documentar: Solicita y almacena un número entero que ingresa el usuario
23         System.out.printf("\nDigite un número entero: ");
24
25         //Documentar: Solicita y almacena un número decimal que ingresa el usuario
26         ...
27
28     }
29
30     numint = scanner.nextInt();
31     System.out.printf("\nDigite un número decimal: ");
32     numdbl = scanner.nextDouble();
33
34     //Documentar:Realiza operaciones con los números ingresados por el usuario
35     // Operación 1: Suma de un número decimal y un número entero
36     res1 = numdbl + numint;
37
38     // Operación 2: División de un número entero entre la parte entera de un número decimal
39     // Nota: Se puede producir un error si la parte decimal de numdbl es cero, ya que no es posible dividir por cero.
40     // Se puede producir un error al realizar la conversión de numdbl a int, la parte decimal se truncará. Esto podría
41     // llevar a una pérdida de información si numdbl tiene una parte decimal significativa. La pérdida de información podría
42     // afectar la precisión del resultado.
43     res2 = numint / (int) numdbl;
44
45     // Operación 3: División de un número entero entre un número decimal
46     // Nota: Se puede producir un error si numdbl es cero, ya que no es posible dividir por cero.
47     res3 = numint / numdbl;
48
49     // Operación 4: División de un número decimal entre un número entero
50     // Nota: Se puede producir un error si numint es cero, ya que no es posible dividir por cero.
51     res4 = numdbl / numint;
52
53     // Operación 5: División de la parte entera de un número decimal entre un número entero
54     // Nota: Se puede producir un error si numint es cero, ya que no es posible dividir por cero.
55     // Se puede producir un error al realizar la conversión de numdbl a int, la parte decimal se truncará. Esto podría
56     ...
57 }
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Current File, Version control.
- File Explorer:** Shows the project structure with `Main.java` selected.
- Code Editor:** Displays the `Main.java` file content. This version of the code adds five additional operations (res1 to res5) involving division and conversion between integers and doubles. It includes detailed comments for each operation, noting potential errors and precision issues.
- Status Bar:** Shows the file size (68.2), line separator (LF), encoding (UTF-8), and indentation (4 spaces).

```
27     numint = scanner.nextInt();
28     System.out.printf("\nDigite un número decimal: ");
29     numdbl = scanner.nextDouble();
30
31     //Documentar:Realiza operaciones con los números ingresados por el usuario
32     // Operación 1: Suma de un número decimal y un número entero
33     res1 = numdbl + numint;
34
35     // Operación 2: División de un número entero entre la parte entera de un número decimal
36     // Nota: Se puede producir un error si la parte decimal de numdbl es cero, ya que no es posible dividir por cero.
37     // Se puede producir un error al realizar la conversión de numdbl a int, la parte decimal se truncará. Esto podría
38     // llevar a una pérdida de información si numdbl tiene una parte decimal significativa. La pérdida de información podría
39     // afectar la precisión del resultado.
40     res2 = numint / (int) numdbl;
41
42     // Operación 3: División de un número entero entre un número decimal
43     // Nota: Se puede producir un error si numdbl es cero, ya que no es posible dividir por cero.
44     res3 = numint / numdbl;
45
46     // Operación 4: División de un número decimal entre un número entero
47     // Nota: Se puede producir un error si numint es cero, ya que no es posible dividir por cero.
48     res4 = numdbl / numint;
49
50     // Operación 5: División de la parte entera de un número decimal entre un número entero
51     // Nota: Se puede producir un error si numint es cero, ya que no es posible dividir por cero.
52     // Se puede producir un error al realizar la conversión de numdbl a int, la parte decimal se truncará. Esto podría
53     ...
54 }
```

```
53 // llevar a una pérdida de información si numdbl tiene una parte decimal significativa. La pérdida de información podría
54 // afectar la precisión del resultado.
55 res5 = (int) numdbl / numint;
...
57 // Operación 6: Suma de la parte entera de un número decimal y un número entero
58 res6 = (int) numdbl + numint;
59
60 // Muestra los resultados de las operaciones
61 System.out.printf("\nEl valor de la operación = " + res1);
62 System.out.printf("\nEl valor de la operación = " + res2);
63 System.out.printf("\nEl valor de la operación = " + res3);
64 System.out.printf("\nEl valor de la operación = " + res4);
65 System.out.printf("\nEl valor de la operación = " + res5);
66 System.out.printf("\nEl valor de la operación = " + res6);
67 }
68 }
```

Prueba

```
1 /**
2 * UPTx - Fundamentos de Programación Orientada a Objetos
3 * Práctica No. 3 Grupo: 5H
4 * Tema: Obtener datos y conversión de tipos
5 * Integrantes: Eder López Villarreal
6 * Modificación: 13/ene/2024
7 */
8
9 import java.util.Scanner;
```

Run Main

```
Digité un número entero: 8
Digité un número decimal: 4.5
El valor de la operación = 12.5
El valor de la operación = 2.0
El valor de la operación = 1.7777777777777777
El valor de la operación = 0.5625
El valor de la operación = 0.0
El valor de la operación = 12.0
Process finished with exit code 0
```

Práctica3 > src > Main

UNIVERSIDAD POLITÉCNICA DE TLAXCALA

Ingeniería en tecnologías de la información

5º “H”

Fundamentos de la Programación

-Practica 4 Clase Carro-

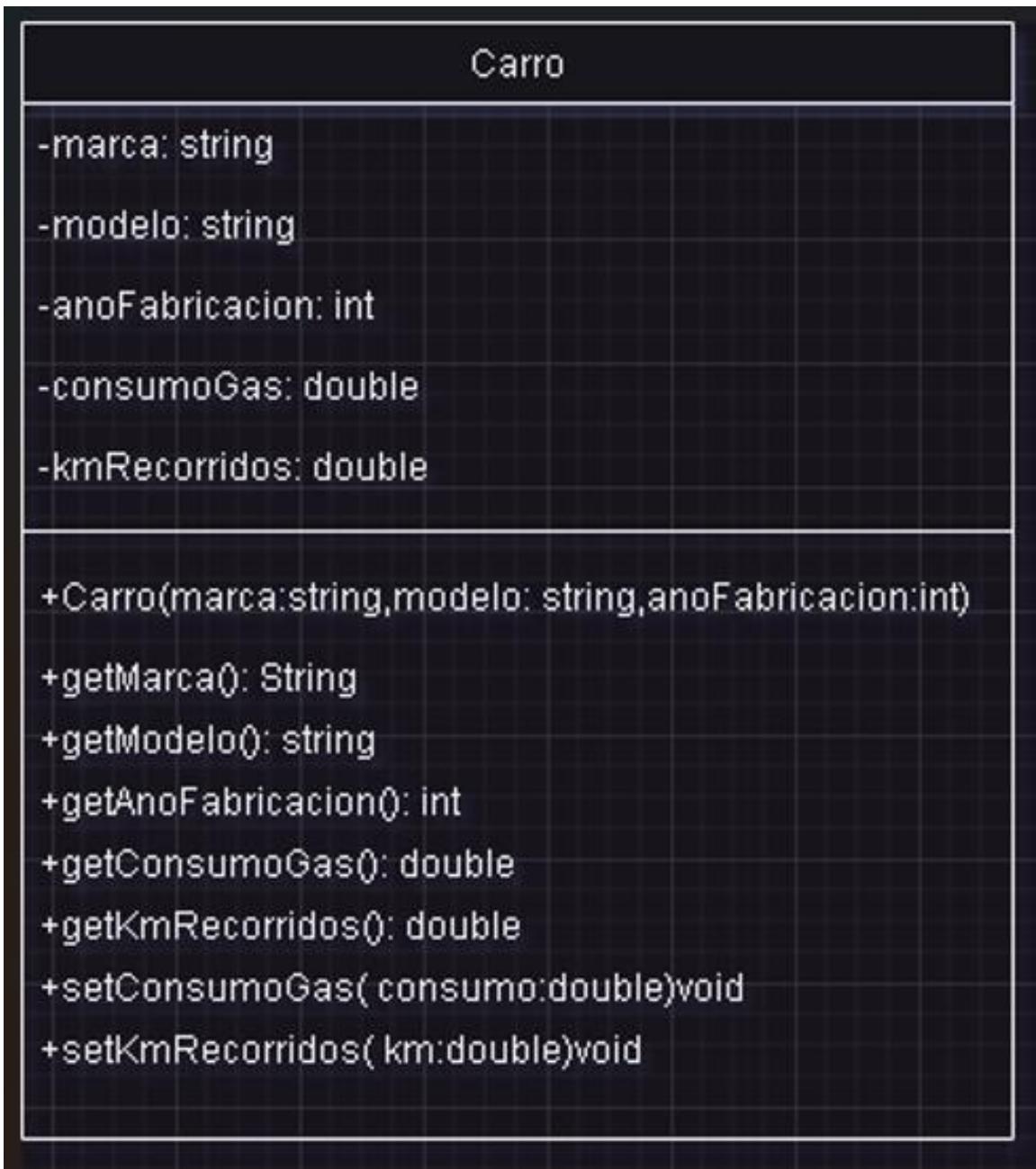
Eric Jhonathan Anaya Márquez
Jazmín Portillo Michicol
Eder López Villarreal
Karen García Leticia Vásquez

18/01/2024

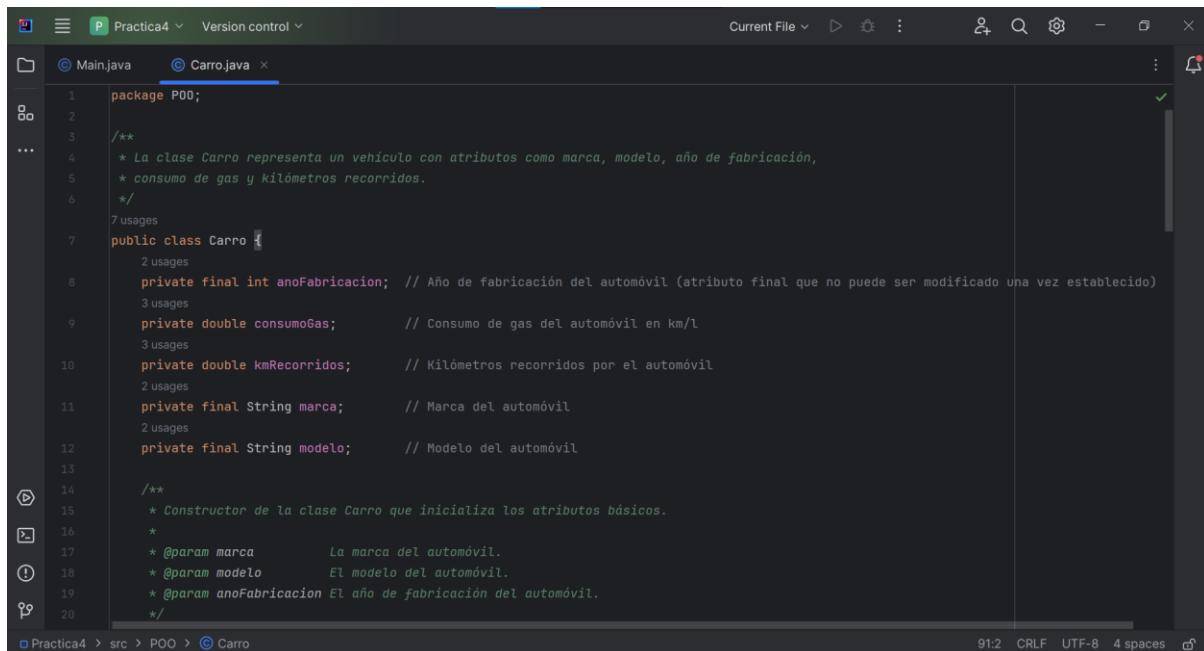
Profesor Saúl Olaf Loaiza



Diagrama UML clasificador del carro



Código de la clase Carro

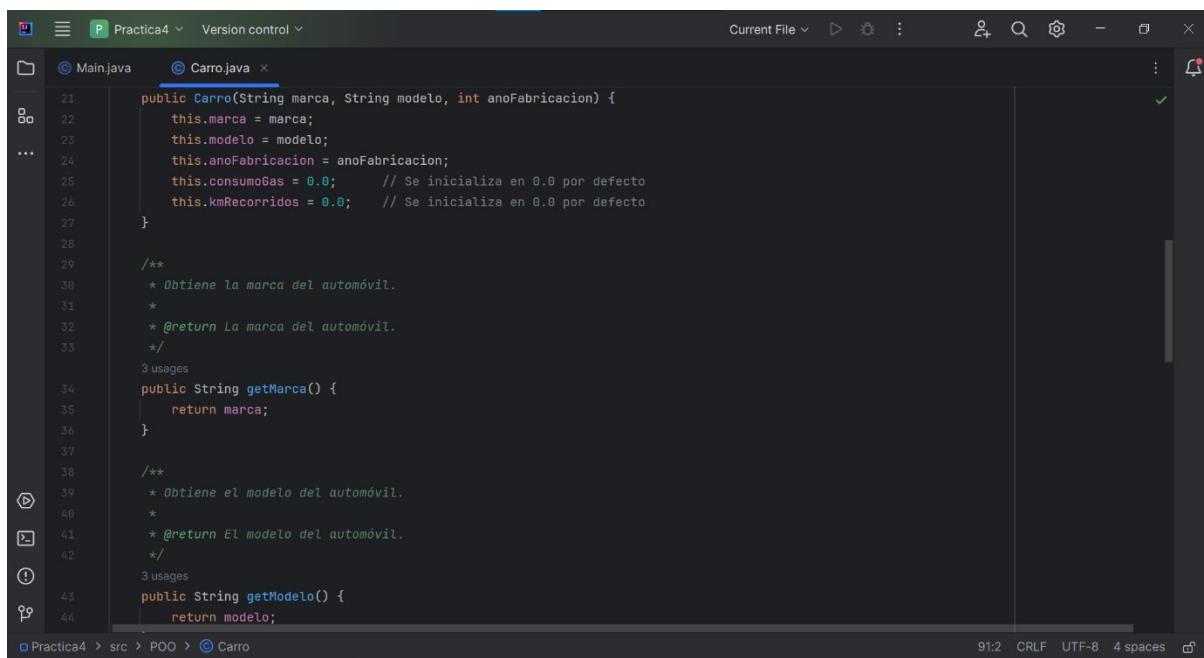


```
package POO;

/**
 * La clase Carro representa un vehículo con atributos como marca, modelo, año de fabricación,
 * consumo de gas y kilómetros recorridos.
 */
7 usages
public class Carro {
    2 usages
    private final int anoFabricacion; // Año de fabricación del automóvil (atributo final que no puede ser modificado una vez establecido)
    3 usages
    private double consumoGas; // Consumo de gas del automóvil en km/l
    3 usages
    private double kmRecorridos; // Kilómetros recorridos por el automóvil
    2 usages
    private final String marca; // Marca del automóvil
    2 usages
    private final String modelo; // Modelo del automóvil

    /**
     * Constructor de la clase Carro que inicializa los atributos básicos.
     *
     * @param marca La marca del automóvil.
     * @param modelo El modelo del automóvil.
     * @param anoFabricacion El año de fabricación del automóvil.
     */
}

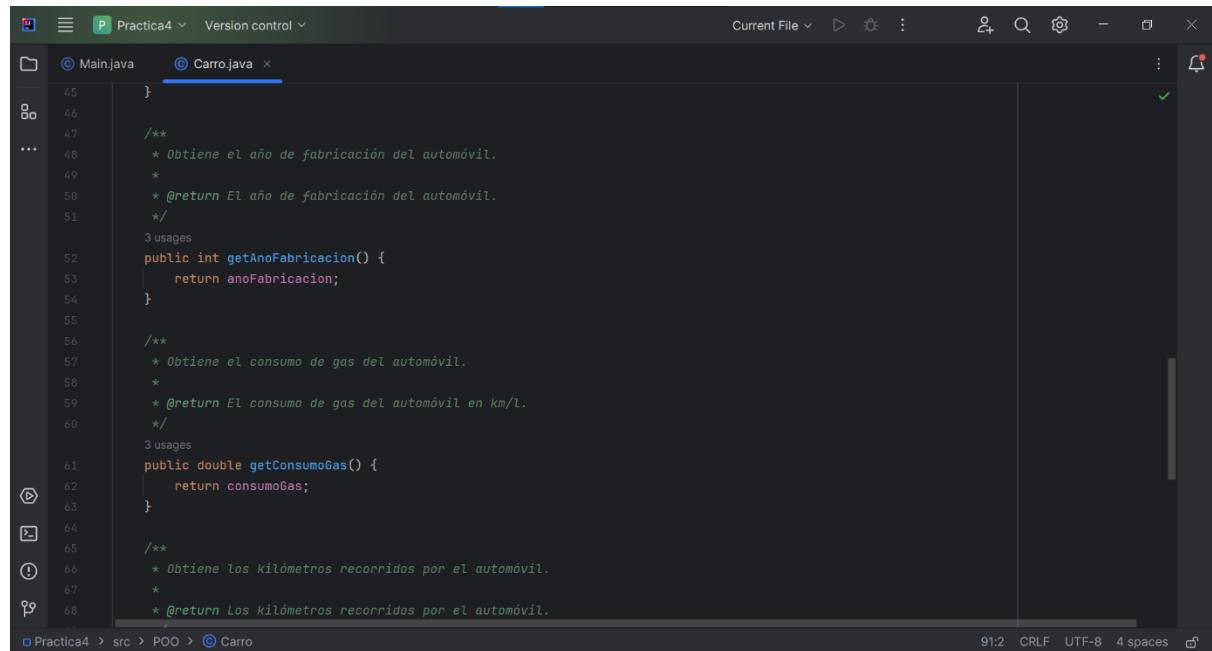
Practica4 > src > POO > Carro
```



```
public Carro(String marca, String modelo, int anoFabricacion) {
    this.marca = marca;
    this.modelo = modelo;
    this.anoFabricacion = anoFabricacion;
    this.consumoGas = 0.0; // Se inicializa en 0.0 por defecto
    this.kmRecorridos = 0.0; // Se inicializa en 0.0 por defecto
}

/**
 * Obtiene la marca del automóvil.
 *
 * @return La marca del automóvil.
 */
3 usages
public String getMarca() {
    return marca;
}

/**
 * Obtiene el modelo del automóvil.
 *
 * @return El modelo del automóvil.
 */
3 usages
public String getModelo() {
    return modelo;
}
```



Practica4 Version control

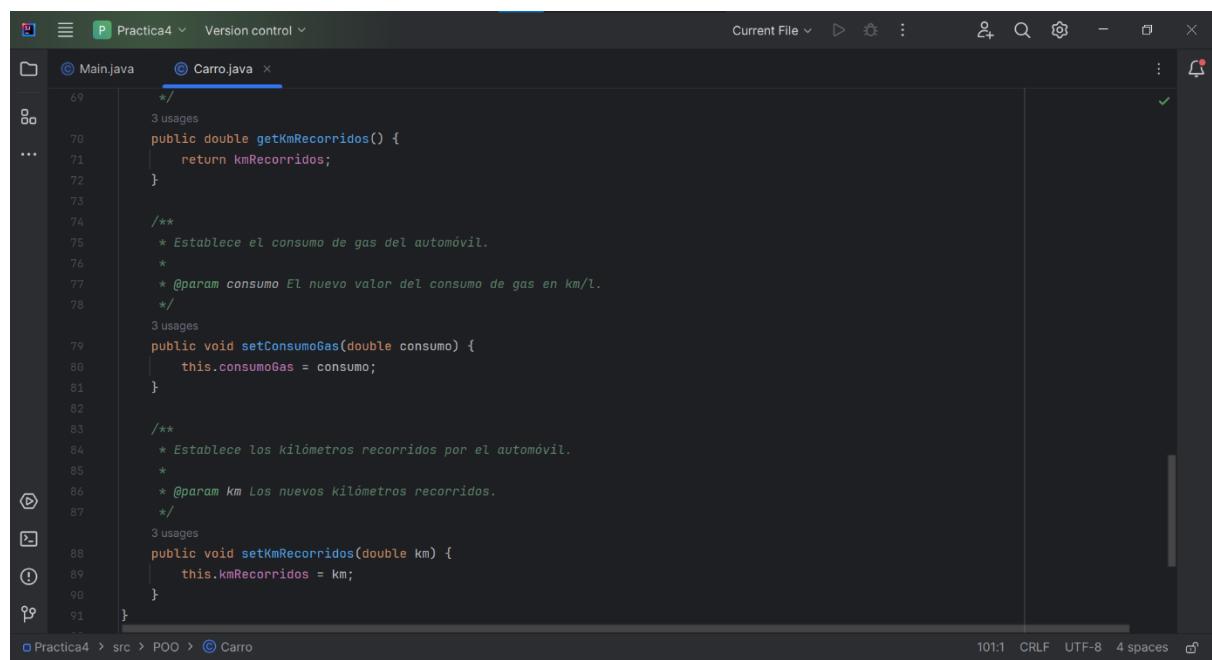
Main.java Carro.java

```
45 }
46 /**
47 * Obtiene el año de fabricación del automóvil.
48 *
49 * @return El año de fabricación del automóvil.
50 */
51 3 usages
52 public int getAnoFabricacion() {
53     return anoFabricacion;
54 }

55 /**
56 * Obtiene el consumo de gas del automóvil.
57 *
58 * @return El consumo de gas del automóvil en km/l.
59 */
60 3 usages
61 public double getConsumoGas() {
62     return consumoGas;
63 }

64 /**
65 * Obtiene los kilómetros recorridos por el automóvil.
66 *
67 * @return Los kilómetros recorridos por el automóvil.
68 */

Practica4 > src > POO > Carro 91:2 CRLF UTF-8 4 spaces ⌂
```



Practica4 Version control

Main.java Carro.java

```
69 */
70 3 usages
71 public double getKmRecorridos() {
72     return kmRecorridos;
73 }

74 /**
75 * Establece el consumo de gas del automóvil.
76 *
77 * @param consumo El nuevo valor del consumo de gas en km/l.
78 */
79 3 usages
80 public void setConsumoGas(double consumo) {
81     this.consumoGas = consumo;
82 }

83 /**
84 * Establece los kilómetros recorridos por el automóvil.
85 *
86 * @param km Los nuevos Kilómetros recorridos.
87 */
88 3 usages
89 public void setKmRecorridos(double km) {
90     this.kmRecorridos = km;
91 }

Practica4 > src > POO > Carro 101:1 CRLF UTF-8 4 spaces ⌂
```

Código de clase principal con tres objetos

Main.java

```

1  /**
2  * UPTx - Fundamentos de Programación Orientada a Objetos
3  * Práctica No. 4      Grupo: 5H
4  * Tema: Clase Carro
5  * Integrantes: Eder López Villarreal, Jazmin Portillo Michicol, Karen Leticia García Vásquez, Eric Jhonathan Anaya Márquez
6  * Objetivo: Realizar una aplicación donde se realice los registros de un coche
7  *             para establecer su mantenimiento, consumo de gas y km recorrido.
8  * Modificación: 18/ene/2024
9  */
10
11 import POO.Carro;
12 public class Main {
13     public static void main(String[] args) {
14
15         // Crea tres objetos Carro
16         Carro carro1 = new Carro("Toyota", "Corolla", 2020);
17         Carro carro2 = new Carro("Honda", "Civic", 2019);
18         Carro carro3 = new Carro("Ford", "Focus", 2021);
19
20         // Actualiza atributos de los carros
21         carro1.setConsumoGas(10.5);
22         carro1.setKmRecorridos(14700.0);
23         carro2.setKmRecorridos(15000.0);
24         carro2.setConsumoGas(12.0); // Establecer el consumo de gas antes de los kilómetros recorridos
25         carro3.setConsumoGas(15.0);
26         carro3.setKmRecorridos(20000.0);

```

Practica4 > src > Main

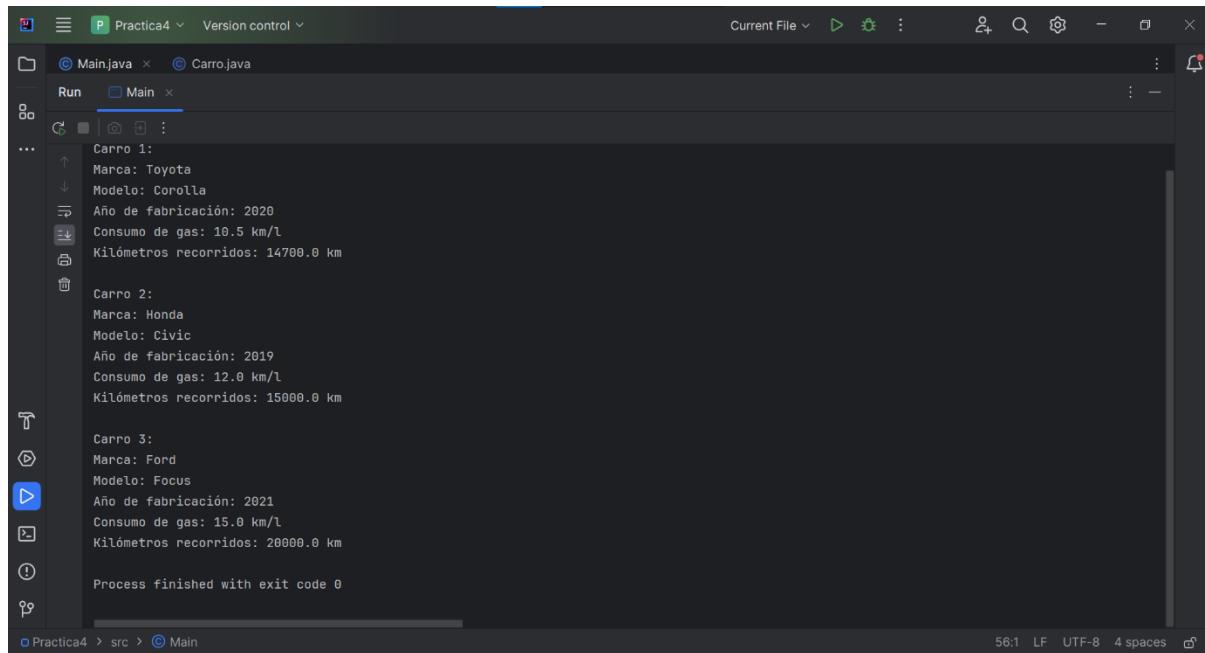
Main.java

```

27
28     // Muestra valores en consola
29     System.out.println("Carro 1:");
30     System.out.println("Marca: " + carro1.getMarca());
31     System.out.println("Modelo: " + carro1.getModelo());
32     System.out.println("Año de fabricación: " + carro1.getAnoFabricacion());
33     System.out.println("Consumo de gas: " + carro1.getConsumoGas() + " km/l");
34     System.out.println("Kilómetros recorridos: " + carro1.getKmRecorridos() + " km\n");
35
36     System.out.println("Carro 2:");
37     System.out.println("Marca: " + carro2.getMarca());
38     System.out.println("Modelo: " + carro2.getModelo());
39     System.out.println("Año de fabricación: " + carro2.getAnoFabricacion());
40     System.out.println("Consumo de gas: " + carro2.getConsumoGas() + " km/l");
41     System.out.println("Kilómetros recorridos: " + carro2.getKmRecorridos() + " km\n");
42
43     System.out.println("Carro 3:");
44     System.out.println("Marca: " + carro3.getMarca());
45     System.out.println("Modelo: " + carro3.getModelo());
46     System.out.println("Año de fabricación: " + carro3.getAnoFabricacion());
47     System.out.println("Consumo de gas: " + carro3.getConsumoGas() + " km/l");
48     System.out.println("Kilómetros recorridos: " + carro3.getKmRecorridos() + " km");
49 }

```

Funcionalidad



```
Carro 1:  
Marca: Toyota  
Modelo: Corolla  
Año de fabricación: 2020  
Consumo de gas: 10.5 km/l  
Kilómetros recorridos: 14700.0 km  
  
Carro 2:  
Marca: Honda  
Modelo: Civic  
Año de fabricación: 2019  
Consumo de gas: 12.0 km/l  
Kilómetros recorridos: 15000.0 km  
  
Carro 3:  
Marca: Ford  
Modelo: Focus  
Año de fabricación: 2021  
Consumo de gas: 15.0 km/l  
Kilómetros recorridos: 20000.0 km  
  
Process finished with exit code 0
```

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

KAREN LETICIA GARCÍA VÁSQUEZ

EDER LÓPEZ VILLARREAL

JAZMIN PORTILLO MICHICOL

ERIC JHONATHAN ANAYA MÁRQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

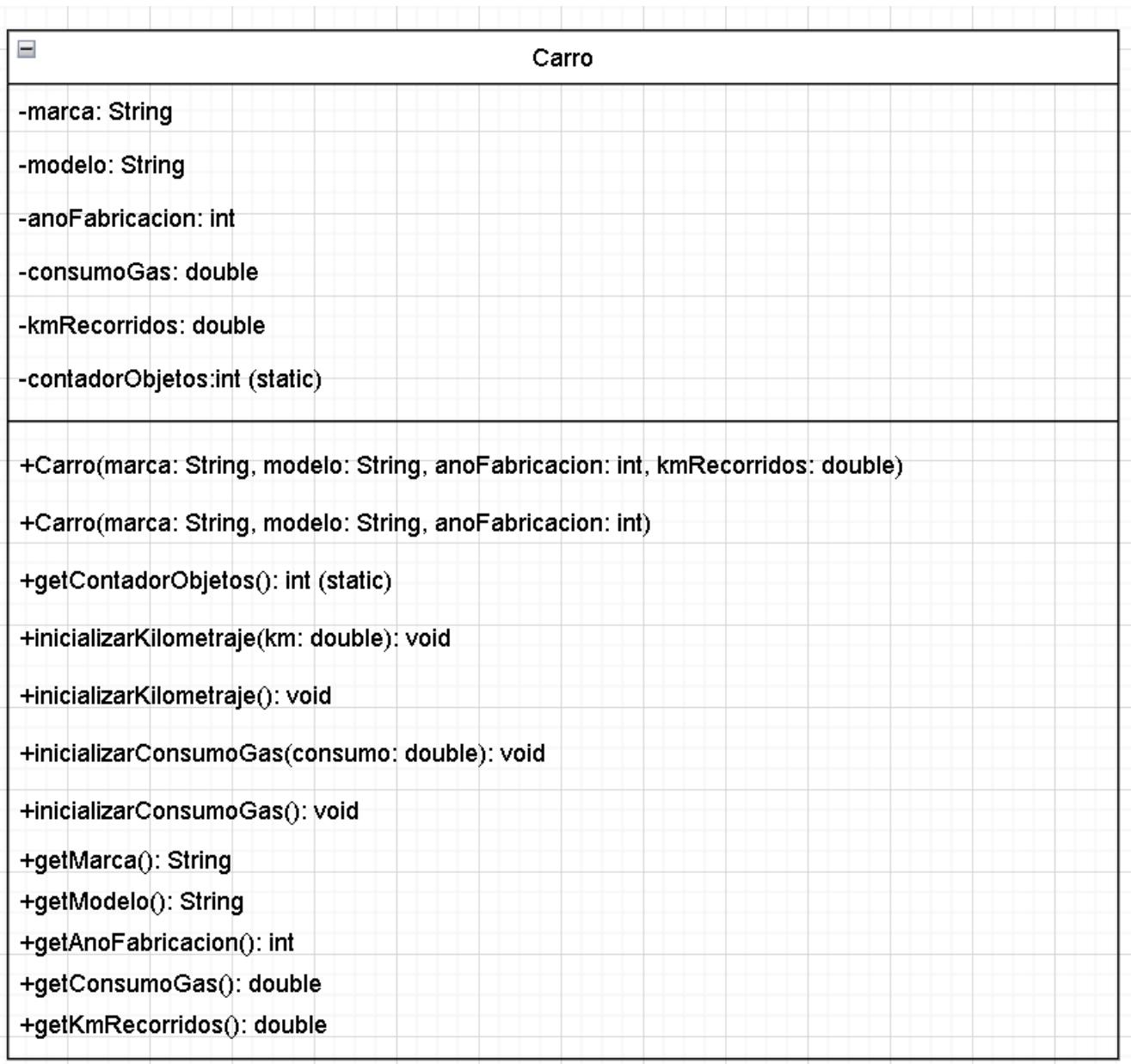
Practica 5

Clase Carro con sobrecarga

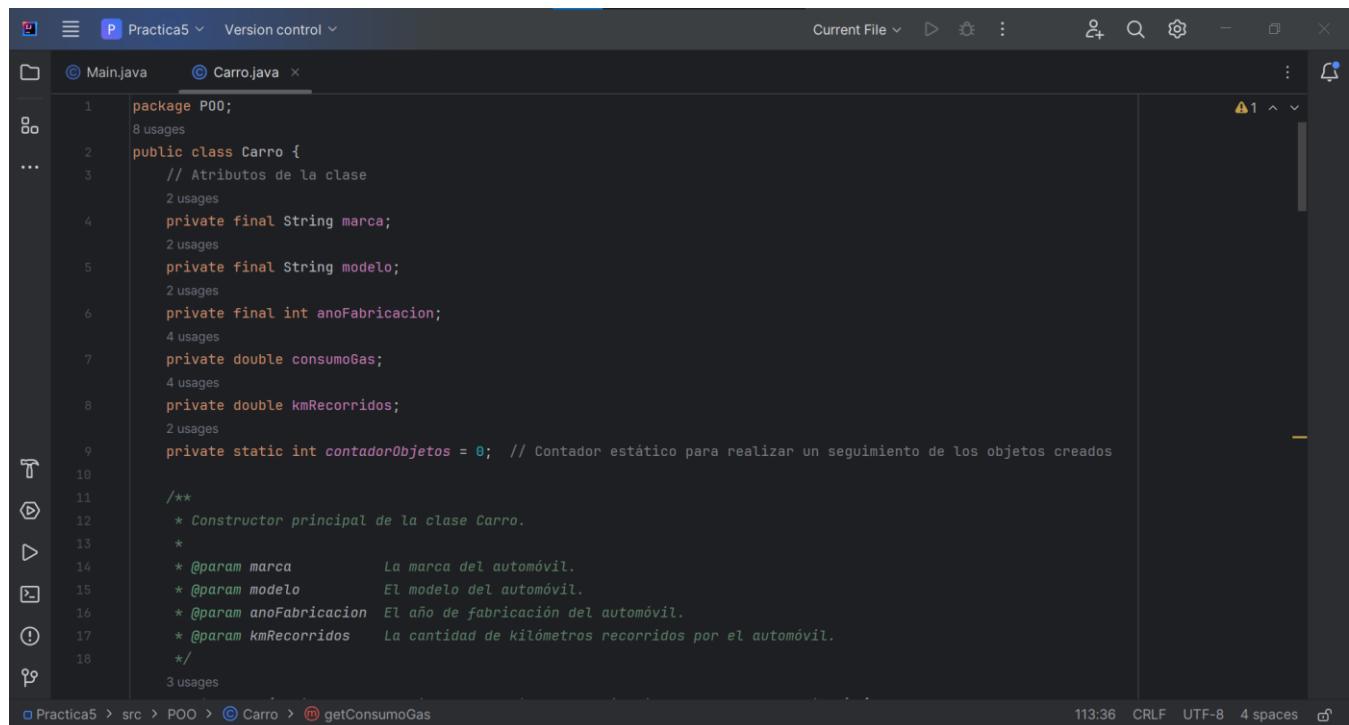
21/01/2024



Diagrama UML del clasificador de la clase Carro



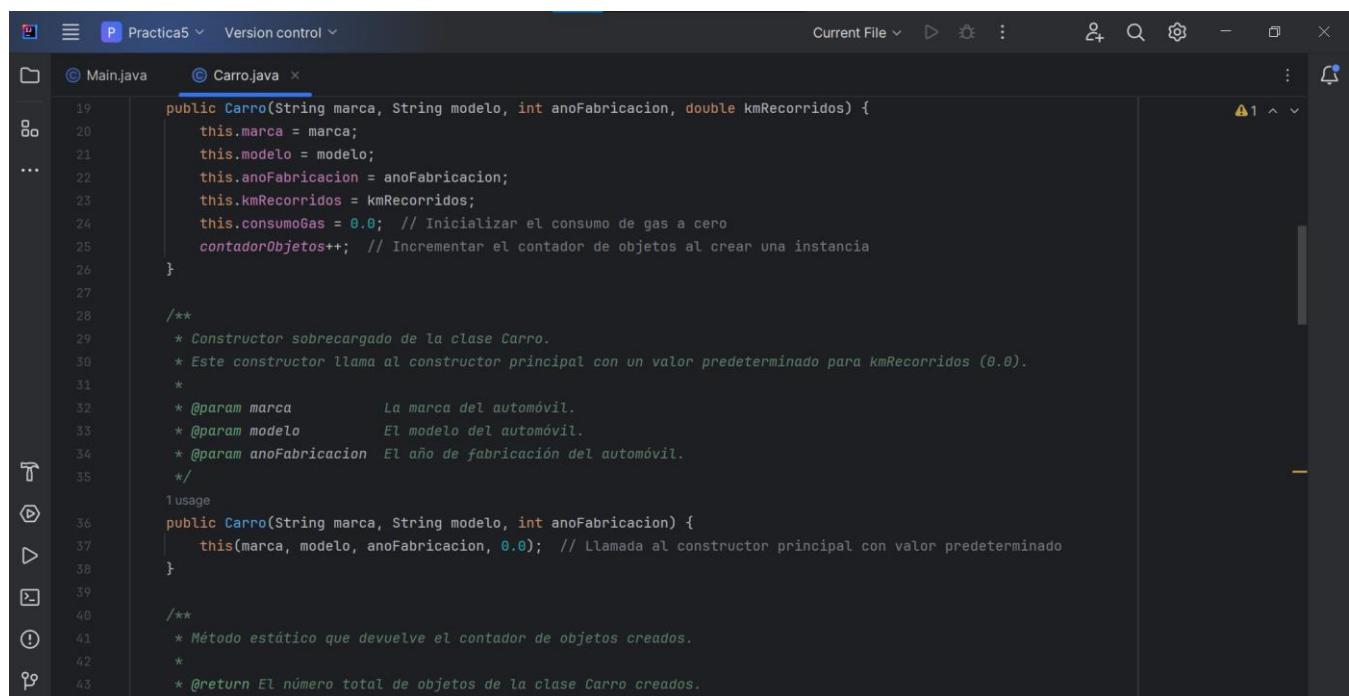
Código de clase Carro



```
package POO;
public class Carro {
    // Atributos de la clase
    private final String marca;
    private final String modelo;
    private final int anoFabricacion;
    private double consumoGas;
    private double kmRecorridos;
    private static int contadorObjetos = 0; // Contador estático para realizar un seguimiento de los objetos creados

    /**
     * Constructor principal de la clase Carro.
     *
     * @param marca La marca del automóvil.
     * @param modelo El modelo del automóvil.
     * @param anoFabricacion El año de fabricación del automóvil.
     * @param kmRecorridos La cantidad de kilómetros recorridos por el automóvil.
     */
    3 usages
}
```

Practica5 > src > POO > Carro > getConsumoGas 113:36 CRLF UTF-8 4 spaces ⌂



```
public Carro(String marca, String modelo, int anoFabricacion, double kmRecorridos) {
    this.marca = marca;
    this.modelo = modelo;
    this.anoFabricacion = anoFabricacion;
    this.kmRecorridos = kmRecorridos;
    this.consumoGas = 0.0; // Inicializar el consumo de gas a cero
    contadorObjetos++; // Incrementar el contador de objetos al crear una instancia
}

/**
 * Constructor sobrecargado de la clase Carro.
 * Este constructor llama al constructor principal con un valor predeterminado para kmRecorridos (0.0).
 *
 * @param marca La marca del automóvil.
 * @param modelo El modelo del automóvil.
 * @param anoFabricacion El año de fabricación del automóvil.
 */
1 usage
public Carro(String marca, String modelo, int anoFabricacion) {
    this(marca, modelo, anoFabricacion, 0.0); // Llamada al constructor principal con valor predeterminado
}

/**
 * Método estático que devuelve el contador de objetos creados.
 *
 * @return El número total de objetos de la clase Carro creados.

```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica5 > Version control
- File List:** Main.java, Carro.java (selected)
- Code Area:** The code for the `Carro` class is displayed, including methods like `getContadorObjetos()`, `inicializarKilometraje(double km)`, `inicializarKilometraje()`, and `inicializarConsumoGas(double consumo)`. Annotations like `@param` and `@return` are present.
- Left Sidebar:** Shows icons for file operations (New, Open, Save, Find, Replace, Delete, Copy, Paste, Undo, Redo), a search bar, and a status bar indicating the current file is `Carro.java`.
- Bottom Status Bar:** Displays the file path (Practica5 > src > POO > Carro), the current time (15:55), and encoding information (CRLF, UTF-8, 4 spaces).

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica5 > Version control
- File List:** Main.java, Carro.java (selected)
- Code Area:** The code for the `Carro` class is displayed, including methods like `getContadorObjetos()`, `inicializarKilometraje(double km)`, `inicializarConsumoGas(double consumo)`, and `getMarca()`. Annotations like `@param` and `@return` are present.
- Left Sidebar:** Shows icons for file operations (New, Open, Save, Find, Replace, Delete, Copy, Paste, Undo, Redo), a search bar, and a status bar indicating the current file is `Carro.java`.
- Bottom Status Bar:** Displays the file path (Practica5 > src > POO > Carro), the current time (126:1), and encoding information (CRLF, UTF-8, 4 spaces).

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Practica5 Version control
- File List:** Main.java, Carro.java (selected)
- Code Area:** Content of Carro.java file.

```
90  /**
91   * Devuelve el modelo del automóvil.
92   *
93   * @return El modelo del automóvil.
94   */
95  /**
96   * Devuelve el año de fabricación del automóvil.
97   *
98   * @return El año de fabricación del automóvil.
99   */
100 /**
101  * Devuelve el consumo de gas del automóvil.
102  *
103  * @return La tasa de consumo de gas del automóvil.
104 /**
105  * Devuelve la cantidad de kilómetros recorridos por el automóvil.
106  *
107  * @return La cantidad de Kilómetros recorridos por el automóvil.
108 /**
109  * Devuelve la cantidad de litros consumidos por el automóvil.
110  *
111  * @return La cantidad de litros consumidos por el automóvil.
112  */
113 public double getConsumoGas() {
114     return consumoGas;
115 }
116 /**
117  * Devuelve la cantidad de kilómetros recorridos por el automóvil.
118  *
119  * @return La cantidad de Kilómetros recorridos por el automóvil.
120  */
121 /**
122  * Devuelve la cantidad de litros consumidos por el automóvil.
123  *
124  * @return La cantidad de litros consumidos por el automóvil.
125 }
```

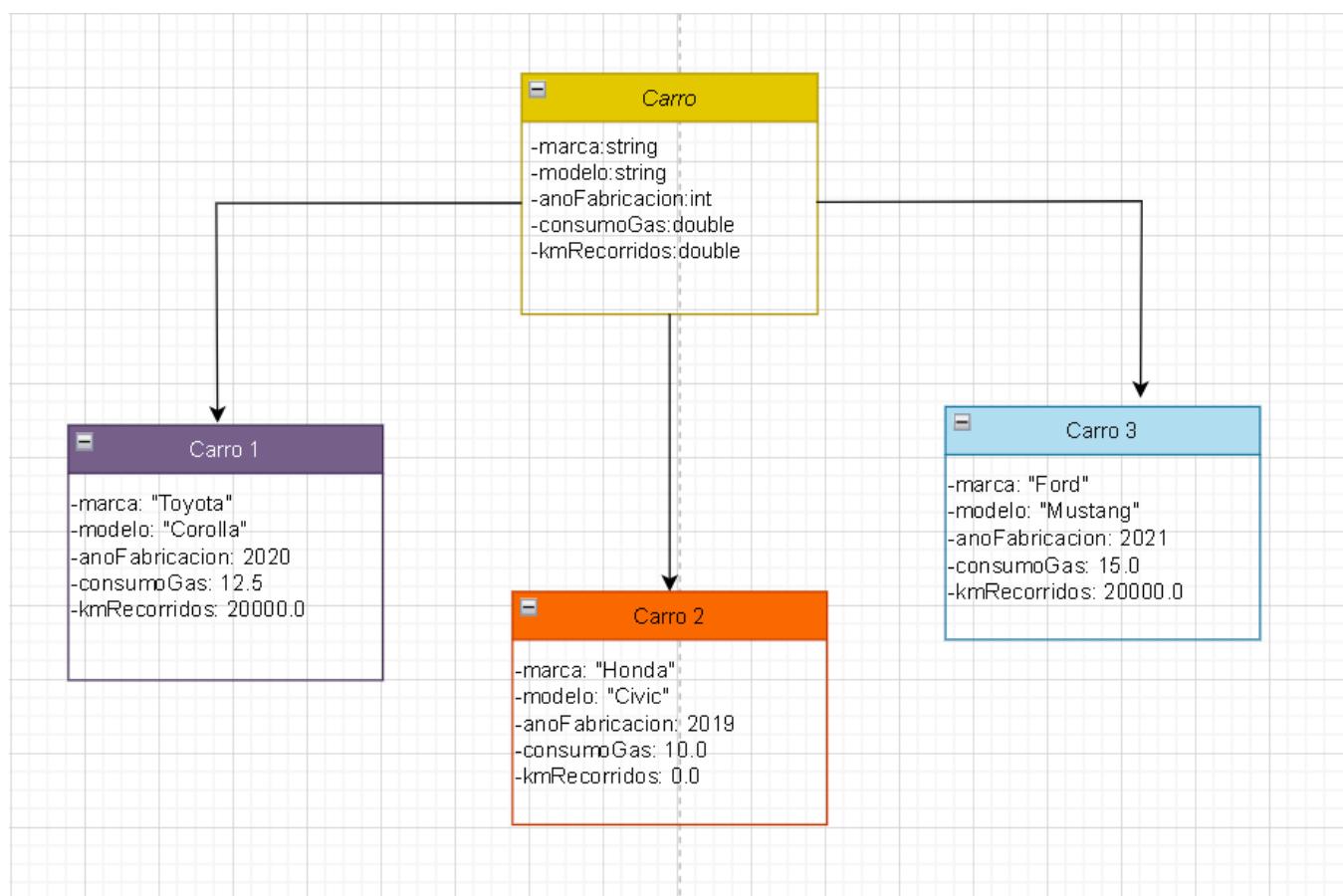
Status Bar: 126:1 CRLF UTF-8 4 spaces

The screenshot shows a Java code editor interface with the following details:

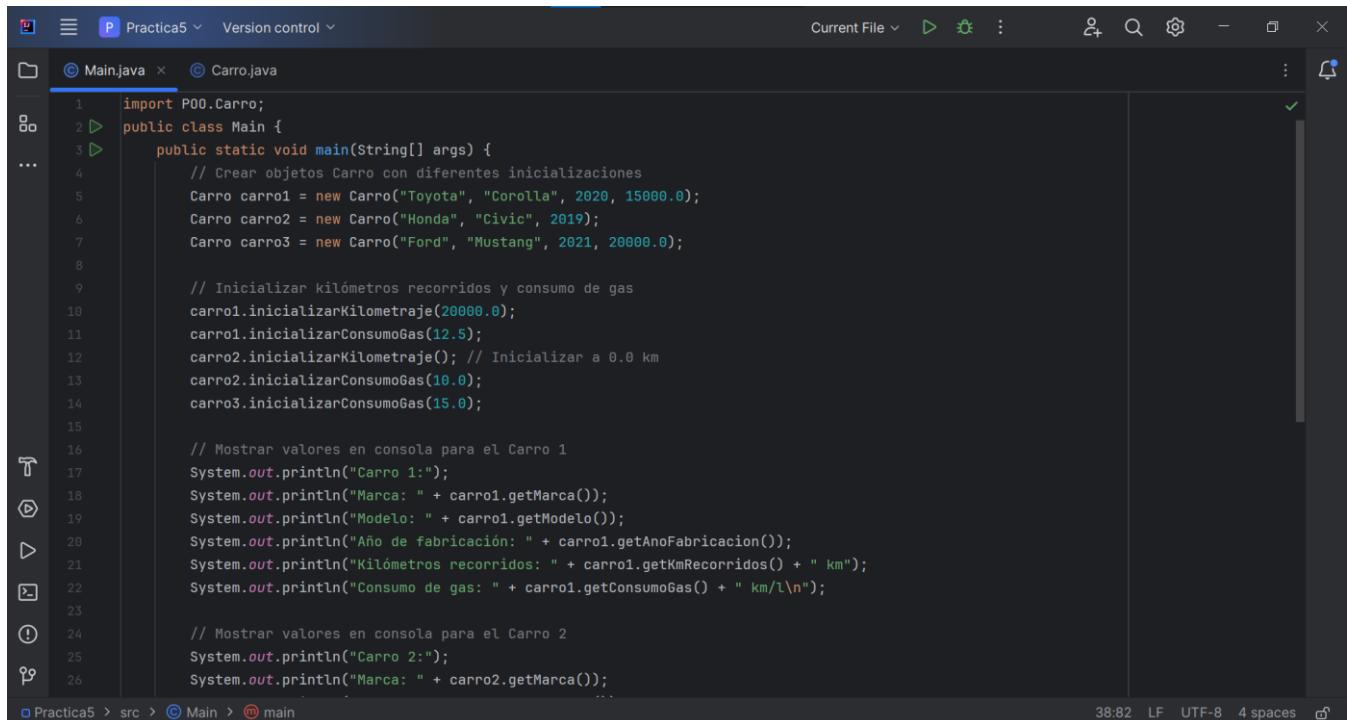
- Title Bar:** Practica5 Version control
- File List:** Main.java, Carro.java (selected)
- Code Area:** Content of Carro.java file.

```
113 public double getConsumoGas() {
114     return consumoGas;
115 }
116 /**
117  * Devuelve la cantidad de kilómetros recorridos por el automóvil.
118  *
119  * @return La cantidad de Kilómetros recorridos por el automóvil.
120  */
121 /**
122  * Devuelve la cantidad de litros consumidos por el automóvil.
123  *
124  * @return La cantidad de litros consumidos por el automóvil.
125 }
```

Diagrama UML de los objetos para verificar la prueba



Clase principal



```
import POO.Carro;
public class Main {
    public static void main(String[] args) {
        // Crear objetos Carro con diferentes inicializaciones
        Carro carro1 = new Carro("Toyota", "Corolla", 2020, 15000.0);
        Carro carro2 = new Carro("Honda", "Civic", 2019);
        Carro carro3 = new Carro("Ford", "Mustang", 2021, 20000.0);

        // Inicializar Kilómetros recorridos y consumo de gas
        carro1.inicializarKilometraje(20000.0);
        carro1.inicializarConsumoGas(12.5);
        carro2.inicializarKilometraje(); // Inicializar a 0.0 km
        carro2.inicializarConsumoGas(10.0);
        carro3.inicializarConsumoGas(15.0);

        // Mostrar valores en consola para el Carro 1
        System.out.println("Carro 1:");
        System.out.println("Marca: " + carro1.getMarca());
        System.out.println("Modelo: " + carro1.getModelo());
        System.out.println("Año de fabricación: " + carro1.getAnoFabricacion());
        System.out.println("Kilómetros recorridos: " + carro1.getKmRecorridos() + " km");
        System.out.println("Consumo de gas: " + carro1.getConsumoGas() + " km/l\n");

        // Mostrar valores en consola para el Carro 2
        System.out.println("Carro 2:");
        System.out.println("Marca: " + carro2.getMarca());
        System.out.println("Modelo: " + carro2.getModelo());
        System.out.println("Año de fabricación: " + carro2.getAnoFabricacion());
        System.out.println("Kilómetros recorridos: " + carro2.getKmRecorridos() + " km");
        System.out.println("Consumo de gas: " + carro2.getConsumoGas() + " km/l\n");

        // Mostrar valores en consola para el Carro 3
        System.out.println("Carro 3:");
        System.out.println("Marca: " + carro3.getMarca());
        System.out.println("Modelo: " + carro3.getModelo());
        System.out.println("Año de fabricación: " + carro3.getAnoFabricacion());
        System.out.println("Kilómetros recorridos: " + carro3.getKmRecorridos() + " km");
        System.out.println("Consumo de gas: " + carro3.getConsumoGas() + " km/l\n");

        // Mostrar el contador de objetos creados
        System.out.println("Contador de objetos: " + Carro.getContadorObjetos());
    }
}
```

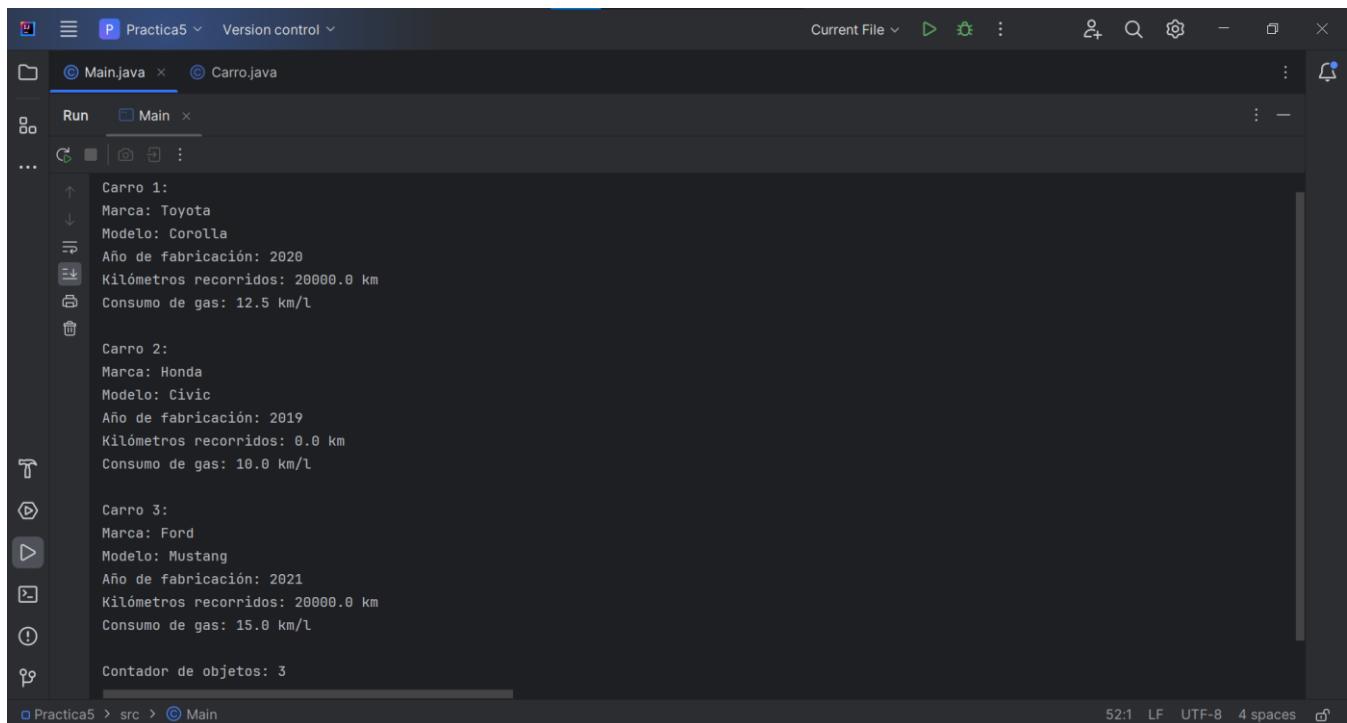


```
System.out.println("Modelo: " + carro2.getModelo());
System.out.println("Año de fabricación: " + carro2.getAnoFabricacion());
System.out.println("Kilómetros recorridos: " + carro2.getKmRecorridos() + " km");
System.out.println("Consumo de gas: " + carro2.getConsumoGas() + " km/l\n");

// Mostrar valores en consola para el Carro 3
System.out.println("Carro 3:");
System.out.println("Marca: " + carro3.getMarca());
System.out.println("Modelo: " + carro3.getModelo());
System.out.println("Año de fabricación: " + carro3.getAnoFabricacion());
System.out.println("Kilómetros recorridos: " + carro3.getKmRecorridos() + " km");
System.out.println("Consumo de gas: " + carro3.getConsumoGas() + " km/l\n");

// Mostrar el contador de objetos creados
System.out.println("Contador de objetos: " + Carro.getContadorObjetos());
```

Prueba



```
Carro 1:  
Marca: Toyota  
Modelo: Corolla  
Año de fabricación: 2020  
Kilómetros recorridos: 20000.0 km  
Consumo de gas: 12.5 km/l  
  
Carro 2:  
Marca: Honda  
Modelo: Civic  
Año de fabricación: 2019  
Kilómetros recorridos: 0.0 km  
Consumo de gas: 10.0 km/l  
  
Carro 3:  
Marca: Ford  
Modelo: Mustang  
Año de fabricación: 2021  
Kilómetros recorridos: 20000.0 km  
Consumo de gas: 15.0 km/l  
  
Contador de objetos: 3
```

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL

ERIC JHONATHAN ANAYA
MARQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Desarrollo clase 1 Clase
Temperatura

26/01/2024

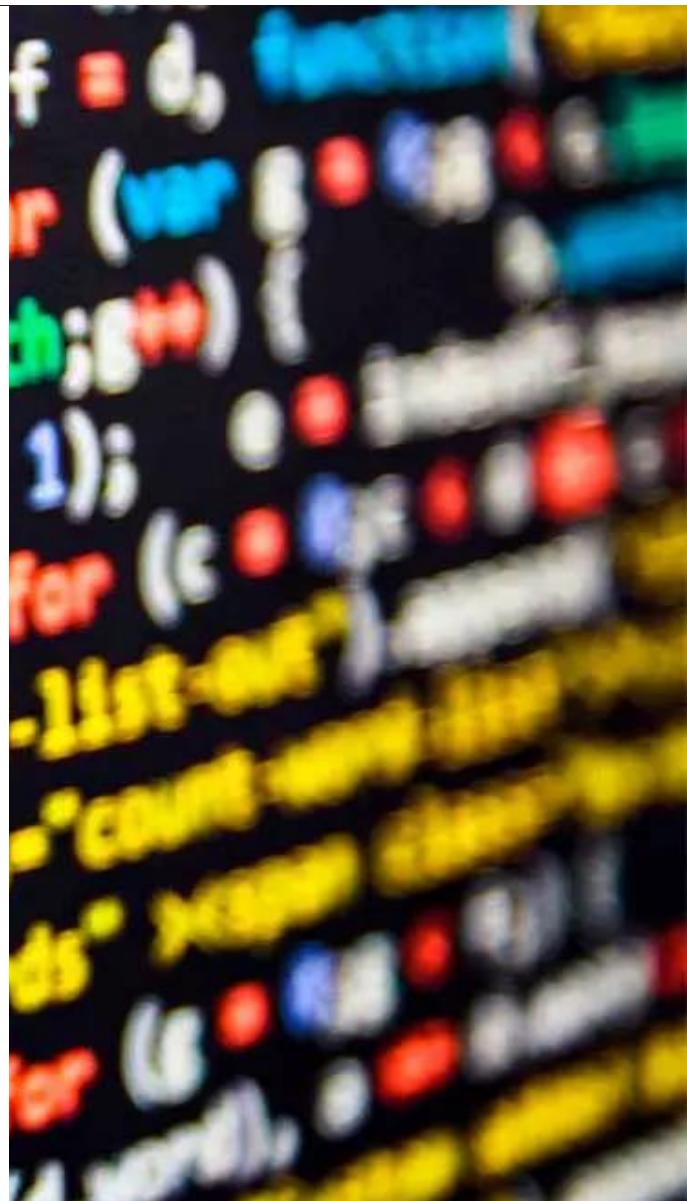
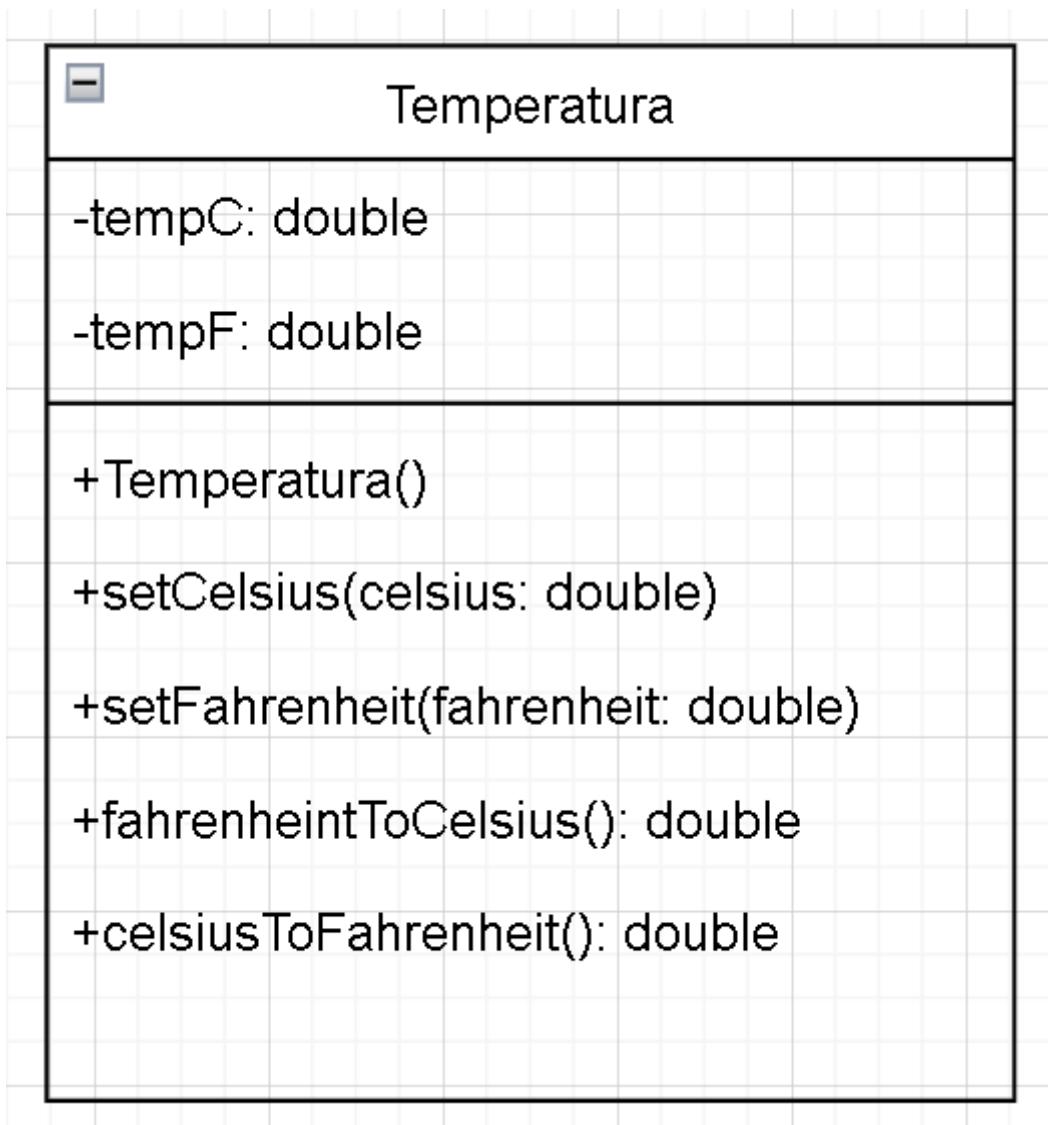


Diagrama UML del clasificador de clase Temperatura



Código de clase Temperatura

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Clase Temperatura > Version control
- File List:** Main.java, Temperatura.java (selected)
- Code Content (Temperatura.java):**

```
1 package POO;
2
3 /**
4  * La clase Temperatura proporciona métodos para convertir entre grados Celsius y Fahrenheit.
5  */
6 public class Temperatura {
7     // Atributos
8     private double tempC; // Temperatura en grados Celsius
9     private double tempF; // Temperatura en grados Fahrenheit
10
11    /**
12     * Constructor por defecto que inicializa los valores de temperatura a 0 grados Celsius y 32 grados Fahrenheit.
13     */
14    public Temperatura() {
15        this.tempC = 0;
16        this.tempF = 32;
17    }
18
19    /**
20     * Establece la temperatura en grados Celsius y actualiza la temperatura en grados Fahrenheit.
21     * @param celsius Temperatura en grados Celsius.
22     */
23 }
```

- Bottom Status Bar:** Checking for JDK updates, 53:1, CRLF, UTF-8, 4 spaces

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Clase Temperatura > Version control
- File List:** Main.java, Temperatura.java (selected)
- Code Content (Temperatura.java):**

```
23 public void setCelsius(double celsius) {
24     this.tempC = celsius;
25     this.tempF = celsiusToFahrenheit(); // Actualizar tempF al establecer tempC
26 }
27
28 /**
29  * Establece la temperatura en grados Fahrenheit y actualiza la temperatura en grados Celsius.
30  * @param fahrenheit Temperatura en grados Fahrenheit.
31  */
32 public void setFahrenheit(double fahrenheit) {
33     this.tempF = fahrenheit;
34     this.tempC = fahrenheitToCelsius(); // Actualizar tempC al establecer tempF
35 }
36
37 /**
38  * Convierte la temperatura de grados Fahrenheit a grados Celsius.
39  * @return Temperatura en grados Celsius.
40  */
41 public double fahrenheitToCelsius() {
42     return (tempF - 32) / 1.8;
43 }
44
45 /**
46  * Convierte la temperatura de grados Celsius a grados Fahrenheit.
47  */
48 }
```

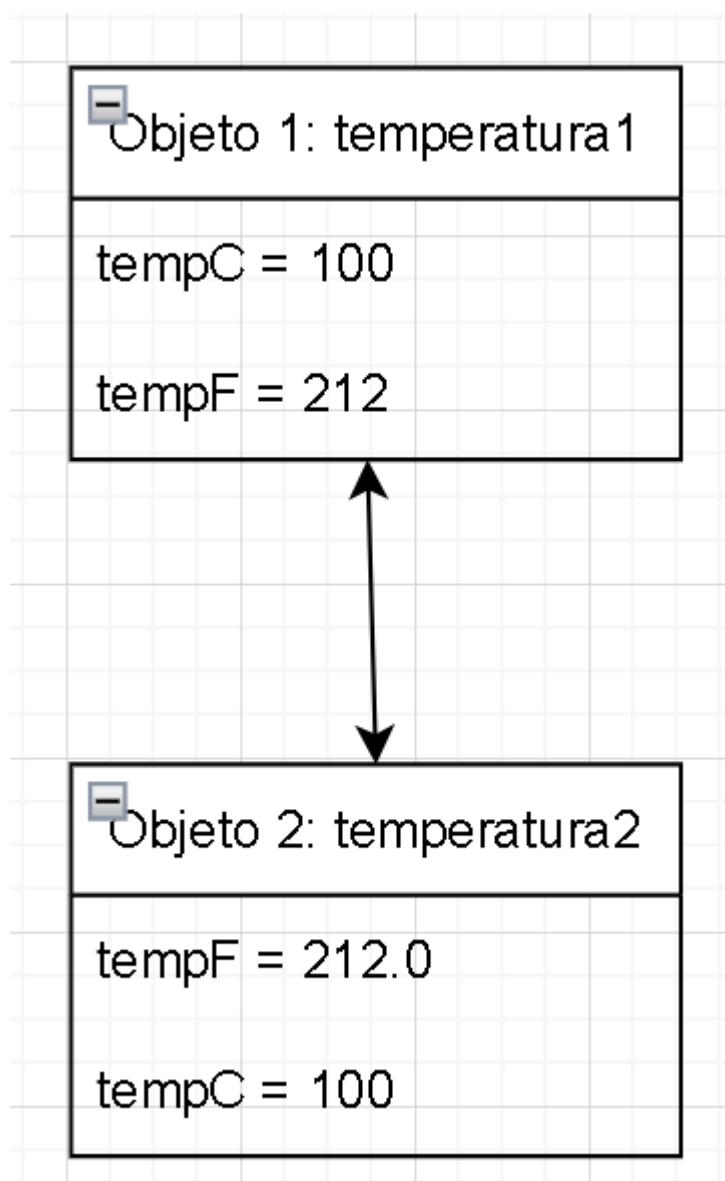
- Bottom Status Bar:** Checking for JDK updates, 53:1, CRLF, UTF-8, 4 spaces

The screenshot shows a Java code editor interface. At the top, there are tabs for "Main.java" and "Temperatura.java". The "Main.java" tab is currently active. Below the tabs, the code for Main.java is displayed:

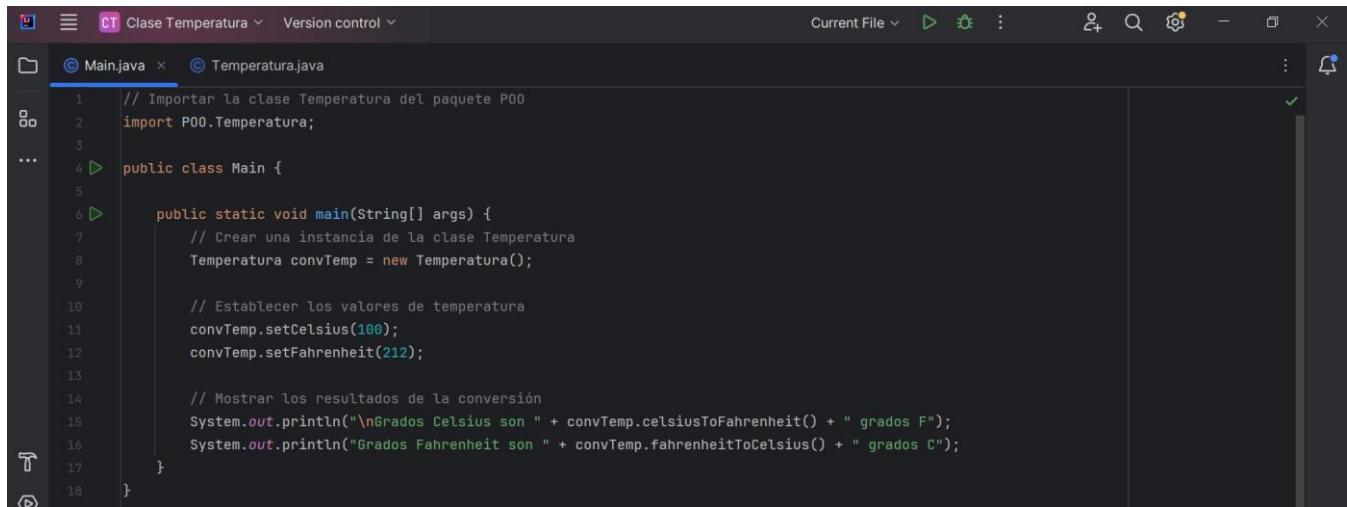
```
47     * @return Temperatura en grados Fahrenheit.  
48     */  
49     public double celsiusToFahrenheit() {  
50         return (1.8 * tempC) + 32;  
51     }  
52 }
```

The code includes a Javadoc comment at the top and a method named `celsiusToFahrenheit()` that converts Celsius to Fahrenheit.

Diagrama UML del objeto para verificar la prueba indicada



Clase principal Main con la prueba solicitada



The screenshot shows a Java code editor interface with two files open:

- Main.java**:

```
1 // Importar la clase Temperatura del paquete POO
2 import POO.Temperatura;
3
4 public class Main {
5
6     public static void main(String[] args) {
7         // Crear una instancia de la clase Temperatura
8         Temperatura convTemp = new Temperatura();
9
10        // Establecer los valores de temperatura
11        convTemp.setCelsius(100);
12        convTemp.setFahrenheit(212);
13
14        // Mostrar los resultados de la conversión
15        System.out.println("\nGrados Celsius son " + convTemp.celsiusToFahrenheit() + " grados F");
16        System.out.println("Grados Fahrenheit son " + convTemp.fahrenheitToCelsius() + " grados C");
17    }
18}
```
- Temperatura.java**: This file is partially visible and contains the implementation of the Temperatura class.

Verificación de prueba

The screenshot shows a Java development environment with the following details:

- Title Bar:** Clase Temperatura
- File Explorer:** Shows two files: Main.java and Temperatura.java.
- Code Editor:** The Main.java file contains the following code:

```
1 // Importar la clase Temperatura del paq
2 import POO.Temperatura;
3
4 public class Main {
```
- Run Tab:** Set to Main.
- Output Window:** Displays the command prompt and the output of the program:

```
"C:\Users\EDER.jdk\openjdk-21.0.2\bin\java -cp . Main
Grados Celsius son 212.0 grados F
Grados Fahrenheit son 100.0 grados C
Process finished with exit code 0
```

Fundamentos de POO

Ingeniería en
Tecnologías de la
Información

EDER LÓPEZ VILLARREAL

ERIC JHONATHAN ANAYA
MARQUEZ

5H T/V

DOCENTE

MTRO. SAUL OLAF LOAIZA

Desarrollo clase 1 Clase
Restaurante

25/01/2024

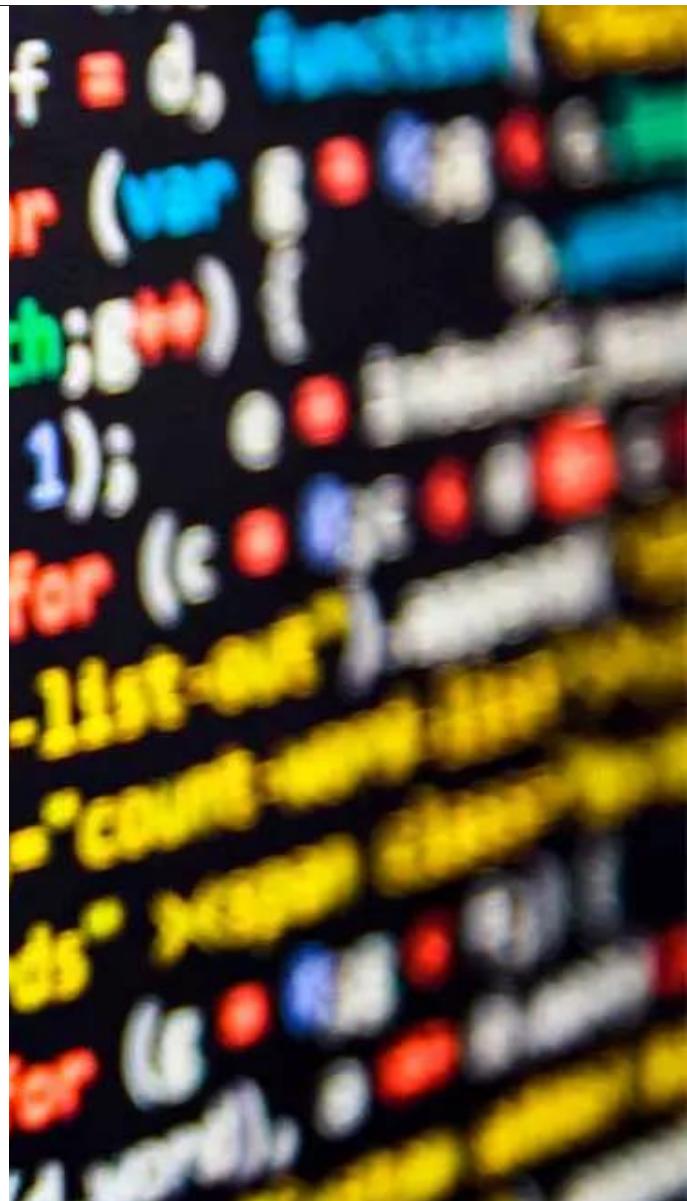
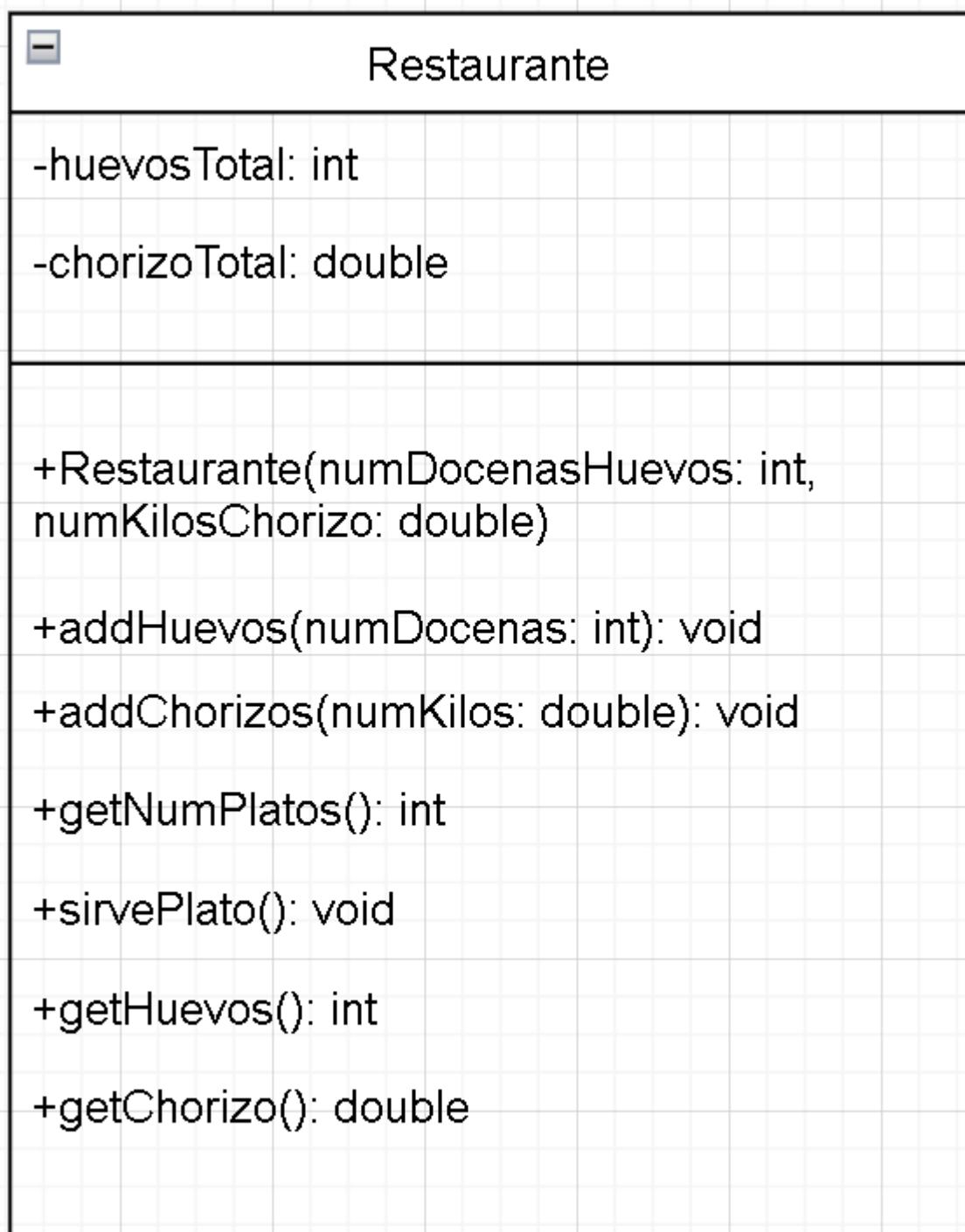
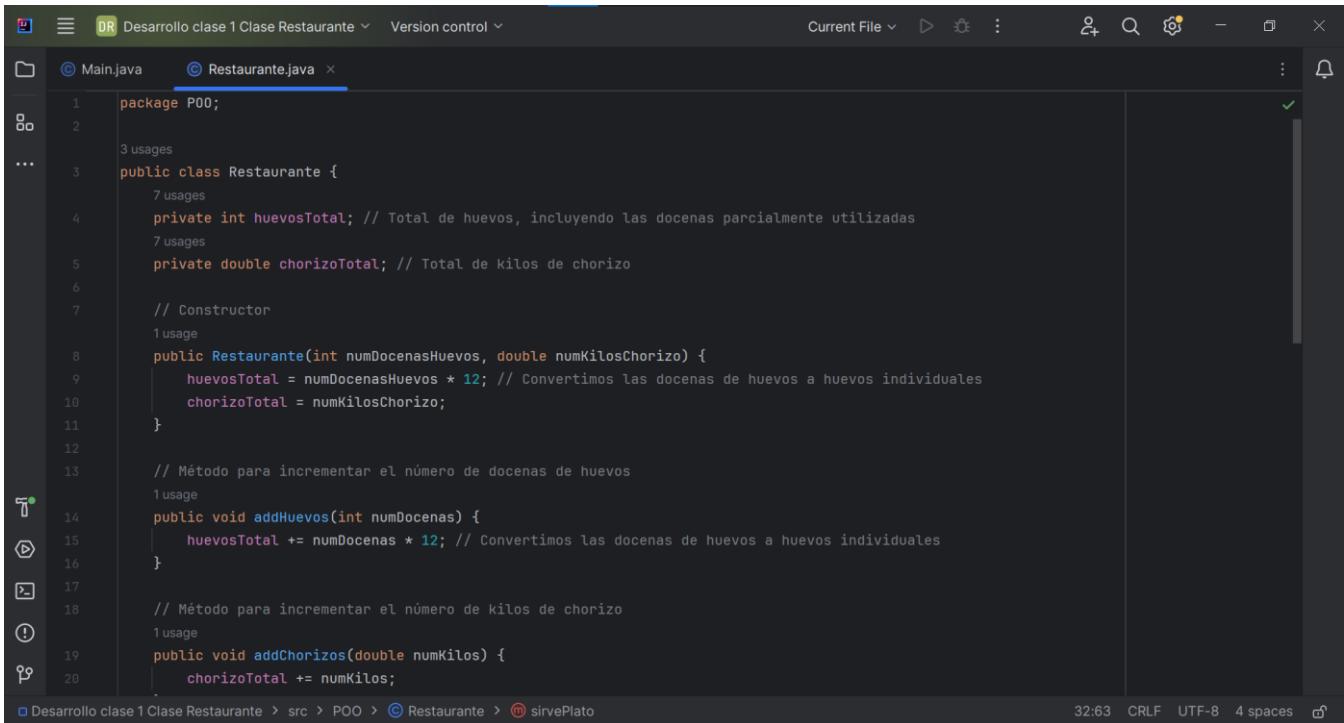


Diagrama UML del clasificador de clase Restaurante



Código de clase Restaurante



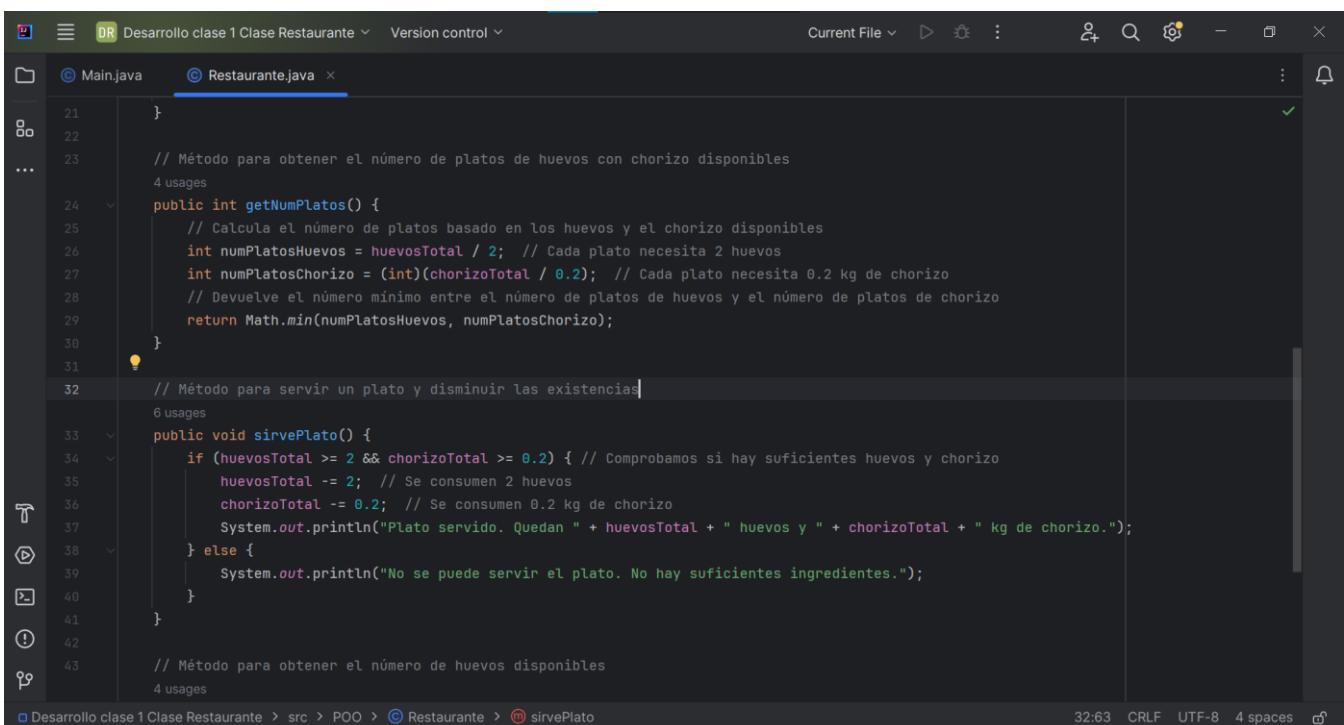
```
package POO;

public class Restaurante {
    private int huevosTotal; // Total de huevos, incluyendo las docenas parcialmente utilizadas
    private double chorizoTotal; // Total de kilos de chorizo

    // Constructor
    public Restaurante(int numDocenasHuevos, double numKilosChorizo) {
        huevosTotal = numDocenasHuevos * 12; // Convertimos las docenas de huevos a huevos individuales
        chorizoTotal = numKilosChorizo;
    }

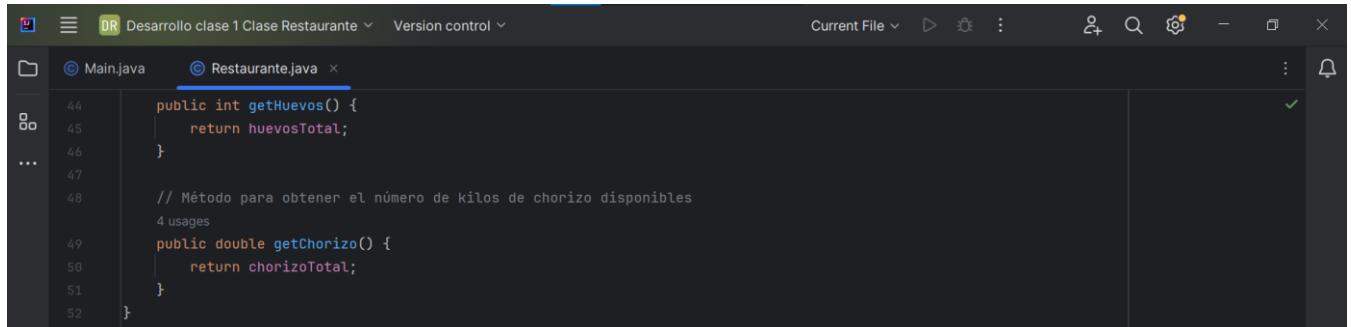
    // Método para incrementar el número de docenas de huevos
    public void addHuevos(int numDocenas) {
        huevosTotal += numDocenas * 12; // Convertimos las docenas de huevos a huevos individuales
    }

    // Método para incrementar el número de kilos de chorizo
    public void addChorizos(double numKilos) {
        chorizoTotal += numKilos;
    }
}
```



```
    // Método para obtener el número de platos de huevos con chorizo disponibles
    public int getNumPlatos() {
        // Calcula el número de platos basado en los huevos y el chorizo disponibles
        int numPlatosHuevos = huevosTotal / 2; // Cada plato necesita 2 huevos
        int numPlatosChorizo = (int)(chorizoTotal / 0.2); // Cada plato necesita 0.2 kg de chorizo
        // Devuelve el número mínimo entre el número de platos de huevos y el número de platos de chorizo
        return Math.min(numPlatosHuevos, numPlatosChorizo);
    }

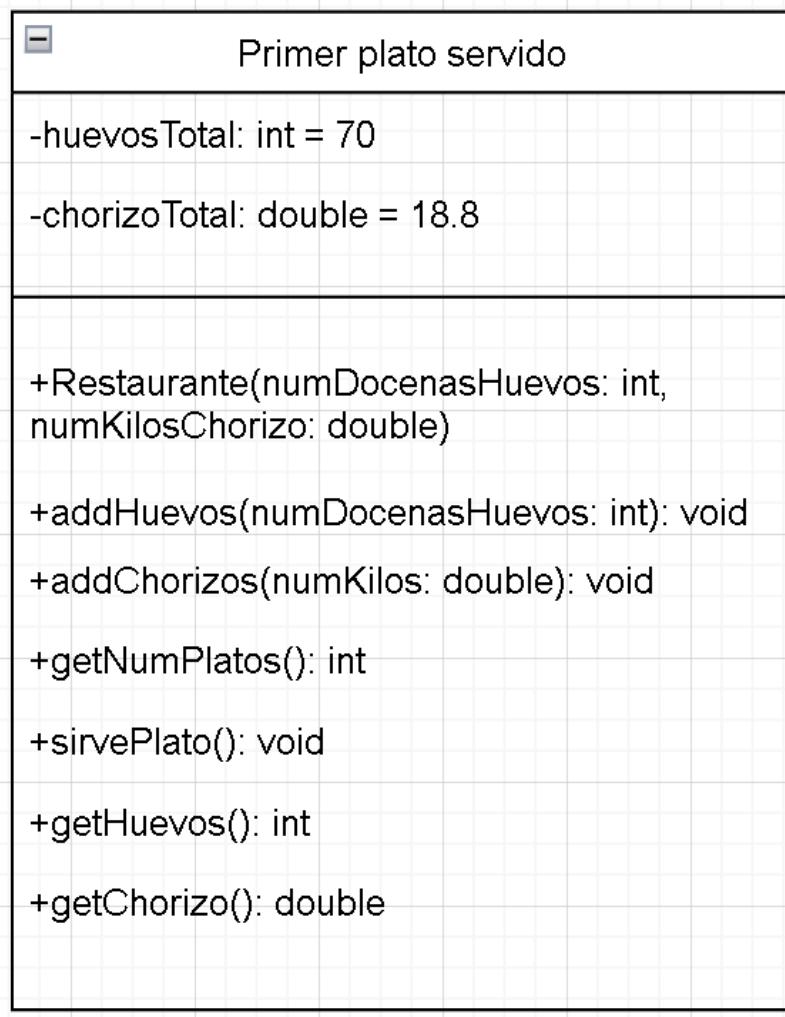
    // Método para servir un plato y disminuir las existencias
    public void sirvePlato() {
        if (huevosTotal >= 2 && chorizoTotal >= 0.2) { // Comprobamos si hay suficientes huevos y chorizo
            huevosTotal -= 2; // Se consumen 2 huevos
            chorizoTotal -= 0.2; // Se consumen 0.2 kg de chorizo
            System.out.println("Plato servido. Quedan " + huevosTotal + " huevos y " + chorizoTotal + " kg de chorizo.");
        } else {
            System.out.println("No se puede servir el plato. No hay suficientes ingredientes.");
        }
    }
}
```



The screenshot shows a Java code editor interface. The title bar indicates the project is "Desarrollo clase 1 Clase Restaurante" and the current file is "Restaurante.java". The code editor displays the following Java code:

```
44 public int getHuevos() {  
45     return huevosTotal;  
46 }  
47  
48 // Método para obtener el número de kilos de chorizo disponibles  
49 public double getChorizo() {  
50     return chorizoTotal;  
51 }  
52 }
```

Diagrama UML del objeto para verificar la prueba indicada



Actualización huevosTotal

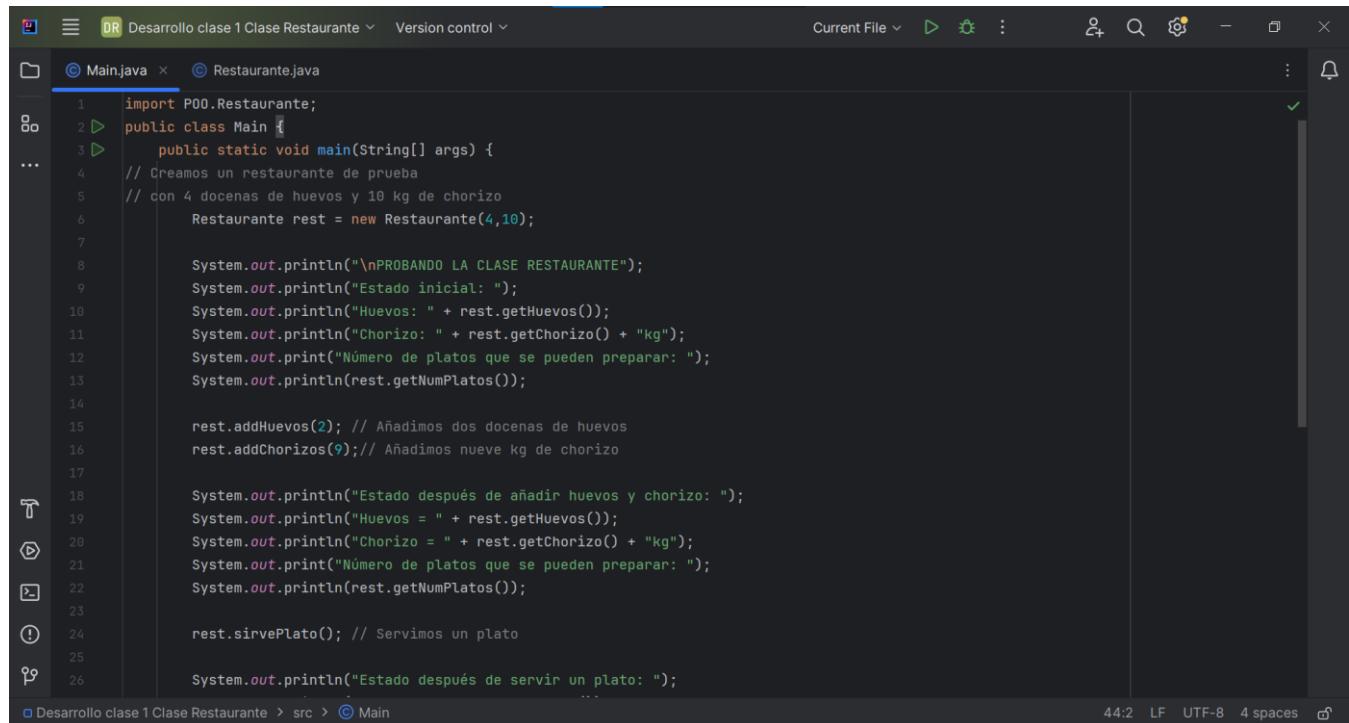
En el constructor, numDocenasHuevos representa el número de docenas de huevos, que se multiplica por 12 para obtener el total de huevos. Por lo tanto, si se pasa 4 como numDocenasHuevos, se calcula $4 * 12 = 48$, lo que establece huevosTotal en 48. Sin embargo, luego se incrementa huevosTotal en 2 docenas más utilizando el método addHuevos(2) en el método main(). Por lo tanto, el valor total de huevosTotal se convierte en $48 + 2 * 12 = 72$. Además, se sirve un plato en el método sirvePlato(), lo que reduce huevosTotal en otros 2, llegando a $72 - 2 = 70$. Y así son las mismas operaciones para cuando se sirven los 5 platos restantes.

Actualización chorizoTotal

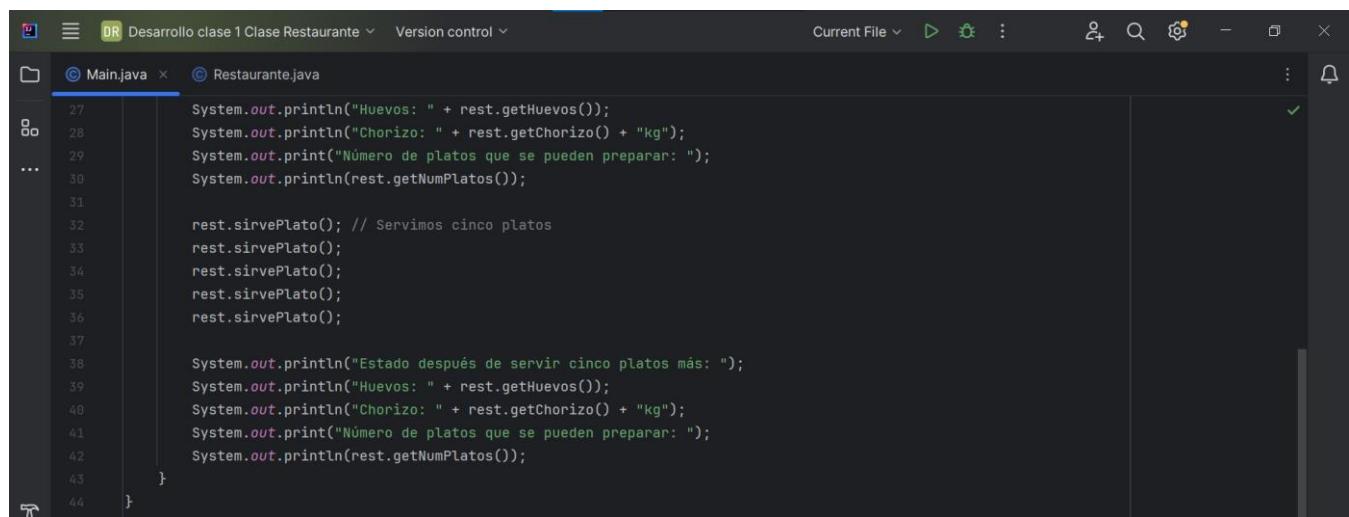
El constructor establece el valor de chorizoTotal según el parámetro numKilosChorizo que se pasa. Luego, durante la ejecución del método main(), se añaden 9 kilos adicionales de chorizo mediante el método addChorizos(9). Después de servir un plato, se consume 0.2 kg de chorizo. Por lo tanto, se reduciría en 0.2 kg. Constructor: chorizoTotal se establece en 10 (valor pasado como argumento). Método addChorizos(9): Se añaden 9 kilos adicionales, por lo que chorizoTotal se convierte en $10 + 9 = 19$.

Método sirvePlato(): Se consume 0.2 kg de chorizo, por lo que chorizoTotal se reduce en 0.2, quedando en $19 - 0.2 = 18.8$. Y así son las mismas operaciones para cuando se sirven los 5 platos restantes.

Clase principal Main con la prueba solicitada



```
1 import POO.Restaurante;
2 public class Main {
3     public static void main(String[] args) {
4         // Creamos un restaurante de prueba
5         // con 4 docenas de huevos y 10 kg de chorizo
6         Restaurante rest = new Restaurante(4,10);
7
8         System.out.println("\nPROBANDO LA CLASE RESTAURANTE");
9         System.out.println("Estado inicial: ");
10        System.out.println("Huevos: " + rest.getHuevos());
11        System.out.println("Chorizo: " + rest.getChorizo() + "kg");
12        System.out.print("Número de platos que se pueden preparar: ");
13        System.out.println(rest.getNumPlatos());
14
15        rest.addHuevos(2); // Añadimos dos docenas de huevos
16        rest.addChorizos(9); // Añadimos nueve kg de chorizo
17
18        System.out.println("Estado después de añadir huevos y chorizo: ");
19        System.out.println("Huevos = " + rest.getHuevos());
20        System.out.println("Chorizo = " + rest.getChorizo() + "kg");
21        System.out.print("Número de platos que se pueden preparar: ");
22        System.out.println(rest.getNumPlatos());
23
24        rest.sirvePlato(); // Servimos un plato
25
26        System.out.println("Estado después de servir un plato:");
27
28
29
30
31
32        rest.sirvePlato(); // Servimos cinco platos
33        rest.sirvePlato();
34        rest.sirvePlato();
35        rest.sirvePlato();
36        rest.sirvePlato();
37
38        System.out.println("Estado después de servir cinco platos más: ");
39        System.out.println("Huevos: " + rest.getHuevos());
40        System.out.println("Chorizo: " + rest.getChorizo() + "kg");
41        System.out.print("Número de platos que se pueden preparar: ");
42        System.out.println(rest.getNumPlatos());
43    }
44}
```



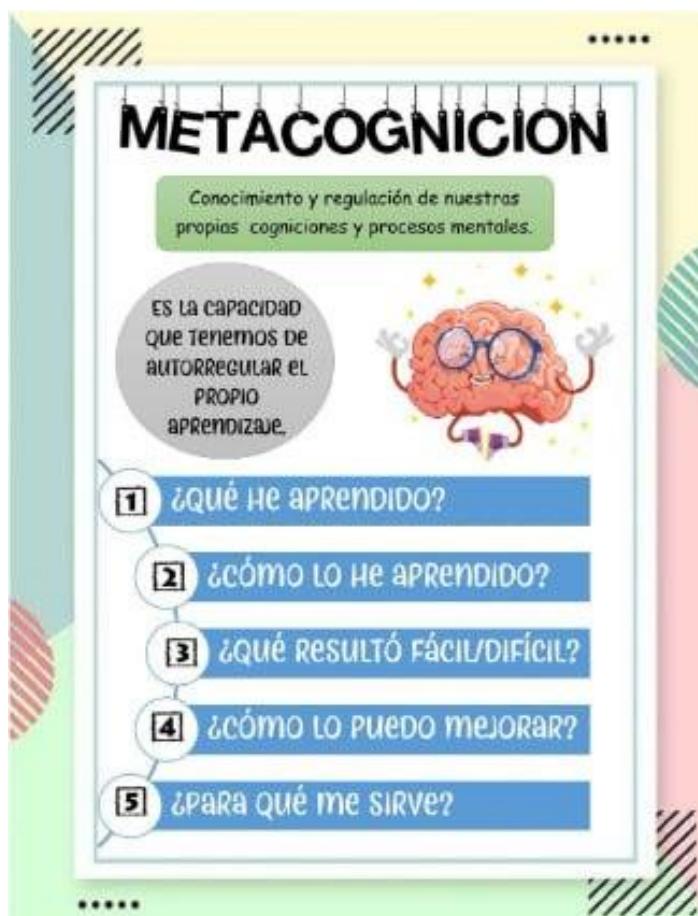
```
27
28
29
30
31
32        rest.sirvePlato(); // Servimos cinco platos
33        rest.sirvePlato();
34        rest.sirvePlato();
35        rest.sirvePlato();
36        rest.sirvePlato();
37
38        System.out.println("Estado después de servir cinco platos más: ");
39        System.out.println("Huevos: " + rest.getHuevos());
40        System.out.println("Chorizo: " + rest.getChorizo() + "kg");
41        System.out.print("Número de platos que se pueden preparar: ");
42        System.out.println(rest.getNumPlatos());
43    }
44}
```

Verificación de prueba

```
PROBANDO LA CLASE RESTAURANTE
Estado inicial:
Huevos: 48
Chorizo: 10.0kg
Número de platos que se pueden preparar: 24
Estado después de añadir huevos y chorizo:
Huevos = 72
Chorizo = 19.0kg
Número de platos que se pueden preparar: 36
Plato servido. Quedan 70 huevos y 18.8 kg de chorizo.
Estado después de servir un plato:
Huevos: 70
Chorizo: 18.8kg
Número de platos que se pueden preparar: 35
Plato servido. Quedan 68 huevos y 18.6 kg de chorizo.
Plato servido. Quedan 66 huevos y 18.400000000000002 kg de chorizo.
Plato servido. Quedan 64 huevos y 18.200000000000003 kg de chorizo.
Plato servido. Quedan 62 huevos y 18.000000000000004 kg de chorizo.
Plato servido. Quedan 60 huevos y 17.800000000000004 kg de chorizo.
Estado después de servir cinco platos más:
Huevos: 60
Chorizo: 17.800000000000004kg
Número de platos que se pueden preparar: 30
```

Desarrollo clase 1 Clase Restaurante > src > Main

AUTOEVALUACIÓN



- 1.- Lo que aprendí a lo largo de este tiempo en clase fue mejorar mi lógica matemática al momento de hacer conversiones de valores, también aprendí a mejorar mi distribución de variables y documentar mejor mis códigos.
- 2.- Lo aprendí en clase ya que el profesor es muy bueno y supo explicar cada ejemplo y a cómo hacerlo y ejecutarlo.
- 3.- Lo que más me resultó difícil fue las conversiones de valores ya que mi lógica no era muy buena.
- 4.- Puedo mejorar si dedico más tiempo a practicar programación en mis tiempos libres.
- 5.- Esto me va a servir para seguir motivándome a seguir programando y cumplir mis objetivos.

AUTOEVALUACIÓN

¿CÓMO LO HE APRENDIDO?

Lo aprendí teniendo un magnífico profesor que supo explicar bien las cosas

¿CÓMO LO PUEDO MEJORAR?

Practicando más en mis tiempos libres

¿QUÉ HE APRENDIDO?

Aprendí a mejorar mi lógica matemática para las conversaciones de valores, y a usar mejor mis valores y definir bien mis constructores.

¿QUÉ RESULTÓ FÁCIL/ DIFÍCIL?

Me resultó algo difícil la comprensión de los cambios de valores

¿PARA QUÉ SIRVE?

Todo esto me va a servir para motivarme más y seguir a delante