**CURSO EN PROGRAMACIÓN ORIENTADA A OBJETOS.**

<https://gemini.google.com/app/c36965a514440f17?hl=es-MX>

**Módulo 1: Fundamentos de la Programación Orientada a Objetos**

**Sesión Teórica 1: Introducción a la POO (¿Qué es, por qué es importante?)**

La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el código en torno a "objetos". Estos objetos son instancias de "clases", que actúan como plantillas para crear los objetos.

Imagina una clase como el plano de un coche y cada coche individual que se fabrica a partir de ese plano como un objeto.

**¿Por qué es importante la POO?**

* **Modularidad:** La POO permite dividir un problema complejo en partes más pequeñas (módulos) y manejables (los objetos), lo que facilita el desarrollo y la comprensión del código.
* **Reutilización:** Las clases pueden ser reutilizadas para crear múltiples objetos con características y comportamientos similares, lo que ahorra tiempo y esfuerzo en el desarrollo.
* **Mantenibilidad:** Al estar el código organizado en objetos independientes, los cambios en una parte del programa tienen menos probabilidades de afectar a otras, lo que facilita el mantenimiento y la corrección de errores.
* **Extensibilidad:** La POO facilita la adición de nuevas funcionalidades al programa mediante la creación de nuevas clases y objetos sin necesidad de modificar el código existente.
* **Modelado del mundo real:** La POO permite modelar los problemas del mundo real de una manera más intuitiva, ya que los objetos pueden representar entidades con sus propias propiedades (atributos) y acciones (métodos).

**Conceptos clave:**

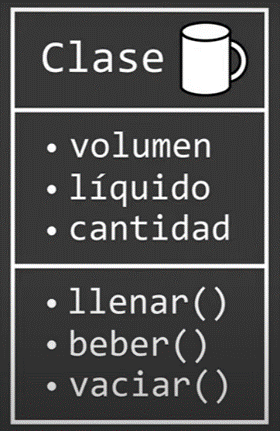
* **Clase:** Una plantilla o blueprint que define la estructura y el comportamiento de los objetos.
* **Objeto:** Una instancia específica de una clase. Contiene los datos (atributos) y puede realizar las acciones (métodos) definidos en su clase.
* **Atributo:** Una característica o propiedad de un objeto (ej. la marca de un coche, el color). Se declara como variables dentro de la clase.
* **Método:** Una acción o comportamiento que un objeto puede realizar (ej. acelerar un coche, frenar). Se define como funciones dentro de la clase.

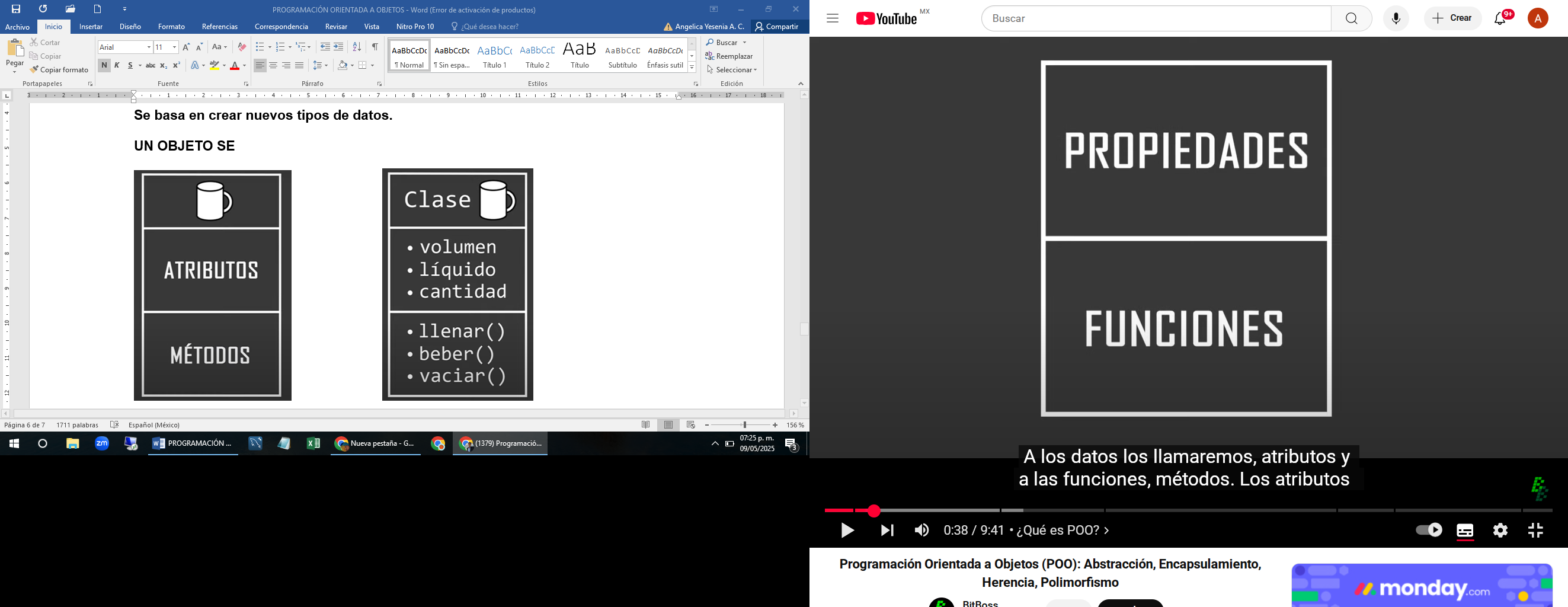
**Video Explicativo (Ejemplo):**

[**Programación Orientada a Objetos (POO): Abstracción, Encapsulamiento, Herencia, Polimorfismo**](https://www.youtube.com/watch?v=SI7O81GMG2A)

Para especificar que se quiere crear un tipo de dato nuevo, **se utiliza la clase**, la cual,

es la plantilla donde se definen los atributos y los métodos.

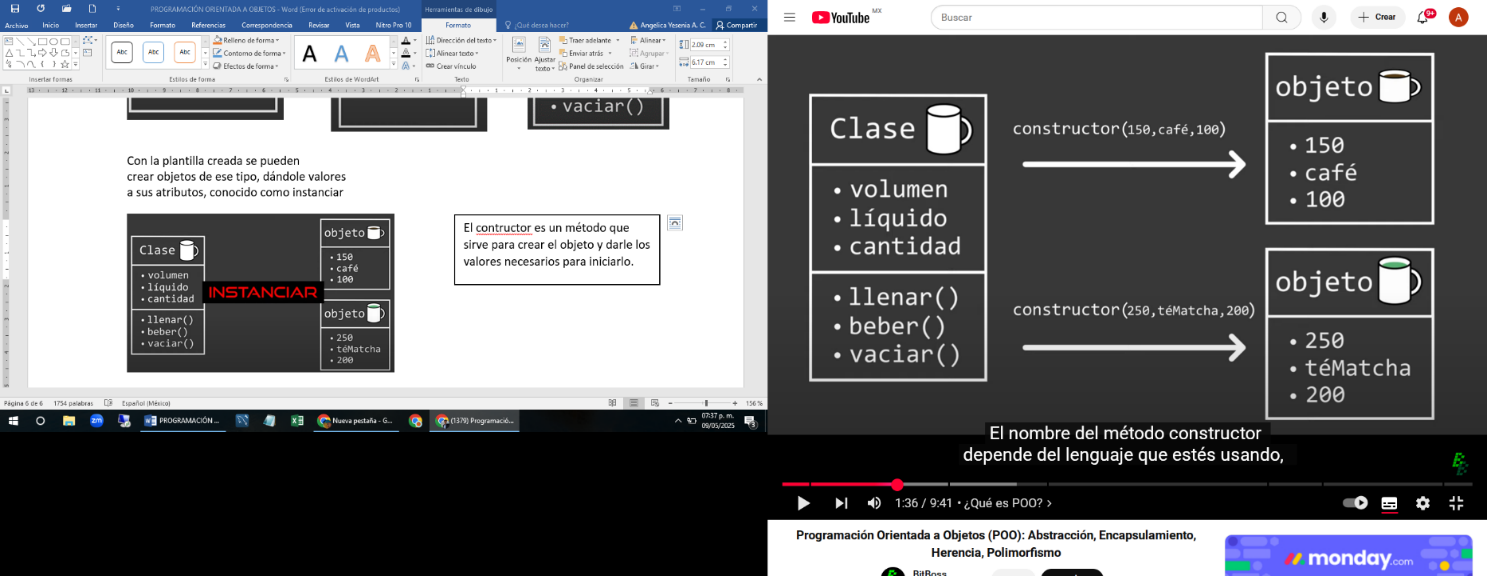
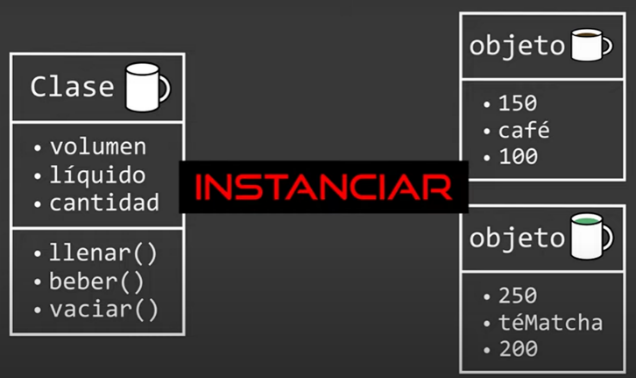




El contructor es un método que sirve para crear el objeto y darle los valores necesarios para iniciarlo. En **java,** el constructor debe tener el mismo nombre de la clase

Con la plantilla creada se pueden

crear objetos de ese tipo, dándole valores

**** sus atributos,conocido como instanciar

Una vez creado un objeto, se puede acceder a sus atributos con un punto (.), ya sea para consultarlos, modificarlos ó para utilizar sus métodos

miTaza1.cantidad 100 miTaza2.liquido leche consultar atributos

miTaza1.cantidad = 0 miTaza2.liquido= téMatcha modificar

miTaza1.llenar() miTaza2.beber() utilizar métodos

**Ahora, vamos a la práctica en IntelliJ IDEA:**

**Ejercicio Práctico 1 en IntelliJ IDEA:**

1. **Abre IntelliJ IDEA.** Si es la primera vez que lo usas, te pedirá que configures algunas opciones. Puedes usar la configuración por defecto para empezar.
2. **Crea un nuevo proyecto Java:**
   * Haz clic en "New Project".
   * Selecciona "Java" en el panel izquierdo.
   * Asegúrate de que el "Project SDK" esté configurado (si no lo está, tendrás que descargarlo).
   * Haz clic en "Next".
   * Desmarca la casilla "Create project from template" y haz clic en "Next".
   * Elige un nombre para tu proyecto (ej. CursoPOO) y una ubicación donde guardarlo.
   * Haz clic en "Finish".
3. **Crea una nueva clase:**
   * En la ventana del proyecto (normalmente a la izquierda), haz clic derecho en la carpeta src (source).
   * Selecciona "New" -> "Java Class".
   * Escribe el nombre de la clase (ej. Coche) y presiona "Enter". Se abrirá un archivo llamado Coche.java en el editor.
4. **Define los atributos de la clase Coche:** Dentro de la clase Coche, declara algunas variables de instancia (atributos):

Java

public class Coche {

String marca;

String modelo;

String color;

}

1. **Crea una clase Main para instanciar objetos Coche:**
   * Repite el paso 3 para crear otra clase llamada Main.
   * Dentro de la clase Main, agrega el método main:

<!-- end list -->

Java

public class Main {

public static void main(String[] args) {

// Aquí crearemos objetos Coche

}

}

1. **Instancia objetos Coche y asigna valores a sus atributos:** Dentro del método main de la clase Main, crea dos objetos de la clase Coche y asigna valores a sus atributos:

Java

public class Main {

public static void main(String[] args) {

Coche coche1 = new Coche(); // Crear un objeto Coche

coche1.marca = "Toyota";

coche1.modelo = "Corolla";

coche1.color = "Rojo";

Coche coche2 = new Coche(); // Crear otro objeto Coche

coche2.marca = "Ford";

coche2.modelo = "Focus";

coche2.color = "Azul";

}

}

1. **Imprime los atributos de los objetos en la consola:** Agrega líneas de código para mostrar los valores de los atributos de cada objeto:

Java

public class Main {

public static void main(String[] args) {

Coche coche1 = new Coche();

coche1.marca = "Toyota";

coche1.modelo = "Corolla";

coche1.color = "Rojo";

Coche coche2 = new Coche();

coche2.marca = "Ford";

coche2.modelo = "Focus";

coche2.color = "Azul";

System.out.println("Coche 1: Marca=" + coche1.marca + ", Modelo=" + coche1.modelo + ", Color=" + coche1.color);

System.out.println("Coche 2: Marca=" + coche2.marca + ", Modelo=" + coche2.modelo + ", Color=" + coche2.color);

}

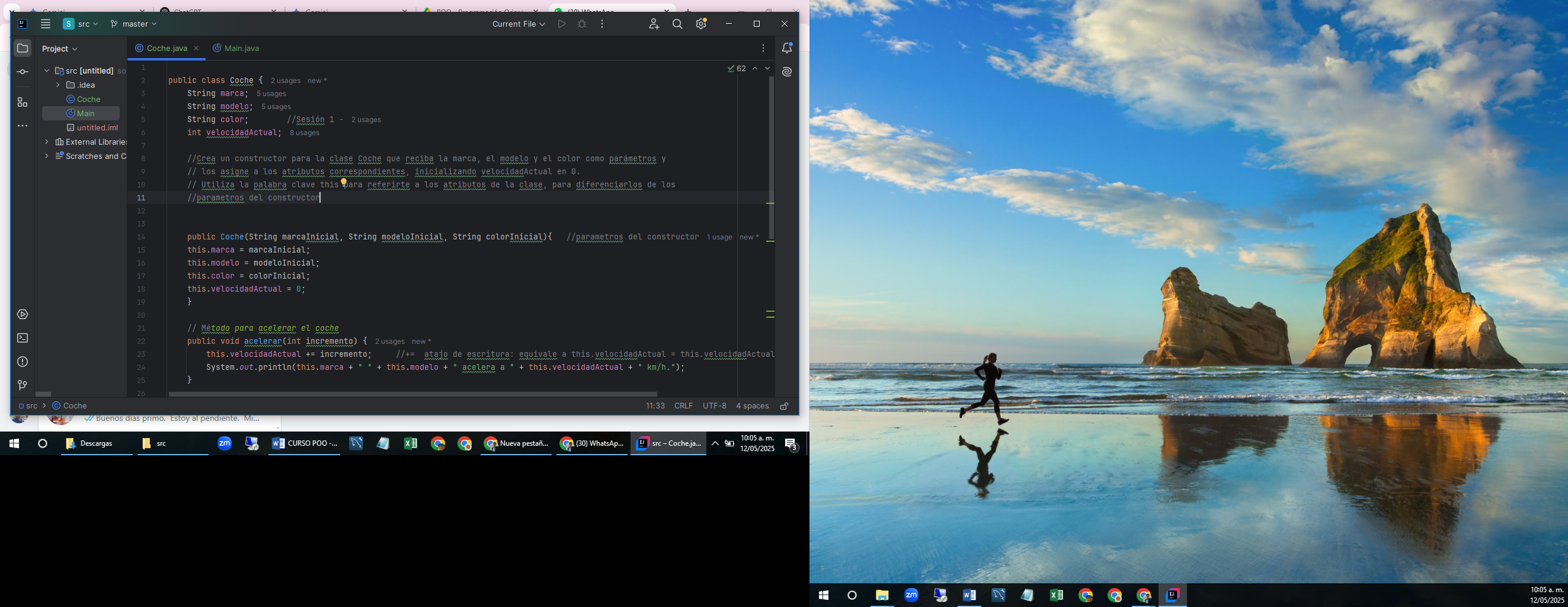
}

1. **Ejecuta el programa:**
   * Haz clic derecho en el archivo Main.java en el editor o en la ventana del proyecto.
   * Selecciona "Run 'Main.main()'".
   * Deberías ver la salida en la ventana de "Run" en la parte inferior de IntelliJ IDEA, mostrando los atributos de los dos objetos Coche.

¡Felicidades! Has creado tu primera clase y tus primeros objetos en Java utilizando IntelliJ IDEA. Hemos cubierto la introducción teórica y realizado un primer ejercicio práctico.

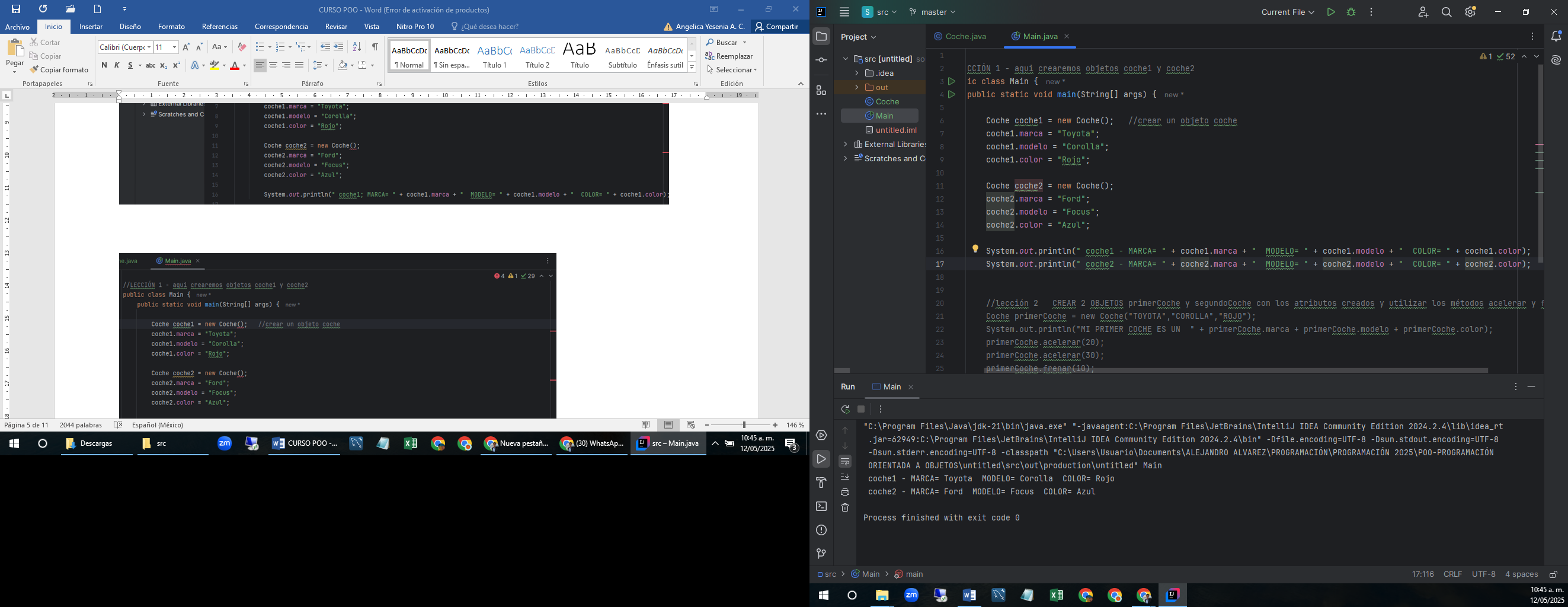
PRÁCTICA 1 EN INTELLIG IDEA

1.- SE CREA LA CLASE COCHE

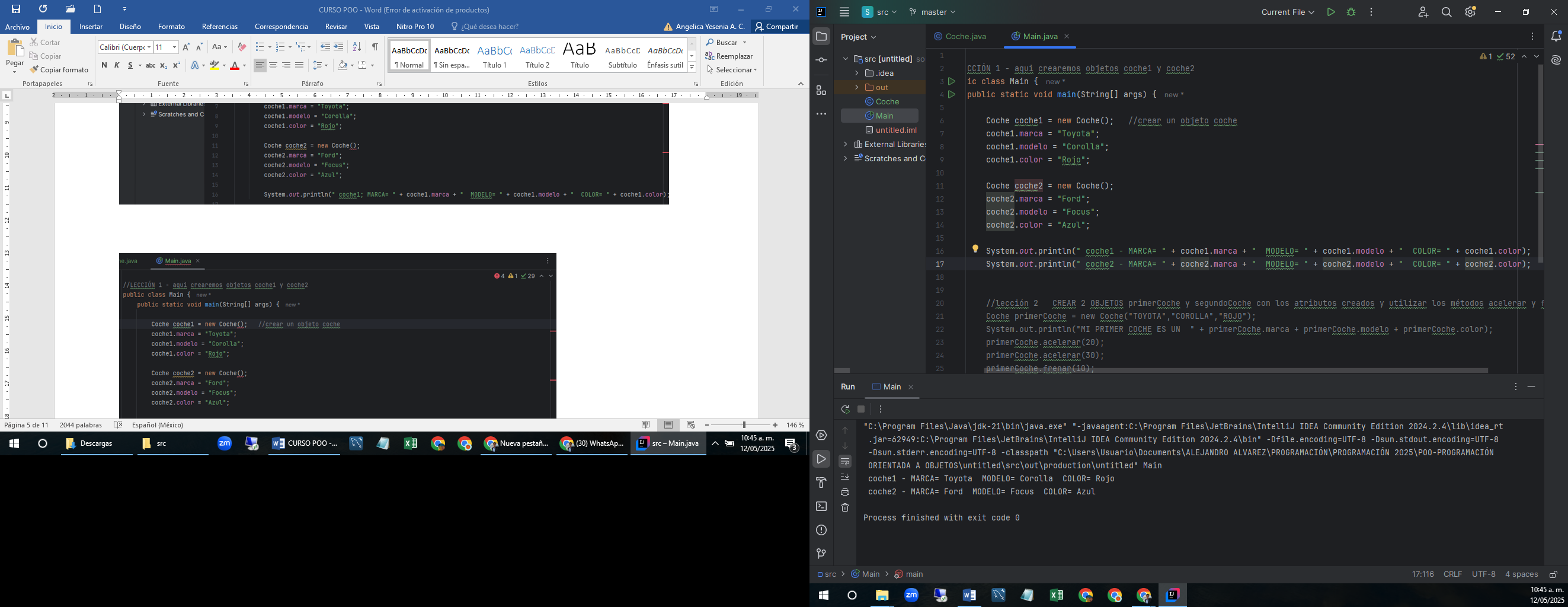


2.- Se crea la clase Main y los objetos coche 1 y coche 2. Aquí se utilizan las variables asignadas en la clase Coche

public class Main {  
 public static void main(String[] args) {  
  
 Coche coche1 = new Coche(); //crear un objeto coche  
 coche1.**marca** = "**Toyota**";  
 coche1.**modelo** = "**Corolla**";  
 coche1.**color** = "**Rojo**";  
  
 Coche coche2 = new Coche();  
 coche2.**marca** = "**Ford"**;  
 coche2**.modelo** = "**Focus**";  
 coche2**.color** = "**Azul**";  
  
 System.*out*.println(" **coche1 - MARCA=** " + coche1.**marca** + " **MODELO= "** + coche1.**modelo** + " **COLOR**= " + coche1.**color**);  
 System.*out*.println(" **coche2 - MARCA=** " + coche2.**marca** + **" MODELO= "** + coche2.**modelo** + " **COLOR=** " + coche2**.color**);



El resultado al ejecutar el programa. Se utilizan las variables de la clase Coche y se imprimen las características asignadas en la clase Main, de los 2 objetos (coche 1) (coche 2)



**SESIÓN Teórica 2: Clases y Objetos**

**(**Definición, instanciación, atributos, métodos).

**Definición y sintaxis de una clase en Java:**

Una clase es un bloque de construcción fundamental en la programación orientada a objetos. Define la estructura **(atributos o datos**) y el comportamiento (**métodos o funciones**) que tendrán los objetos creados a partir de ella.

En Java, la sintaxis básica para definir una clase es la siguiente:

Java

**public class NombreDeLaClase {**

**// Declaración de atributos (variables de instancia)**

**TipoDato nombreDelAtributo;**

**TipoDato otroAtributo;**

**// Definición de métodos**

**TipoRetorno nombreDelMetodo(TipoParametro parametro) {**

**// Cuerpo del método**

**return valorDeRetorno;**

**}**

**// Constructor (un tipo especial de método)**

**public NombreDeLaClase(TipoParametro parametro) {**

**// Inicialización de atributos**

**}**

**}**

**public class NombreDeLaClase**: Declara una clase pública con el nombre especificado. La palabra clave public indica que esta clase puede ser accedida desde cualquier otra parte del programa.

**Constructor**: Es un método especial que se llama automáticamente cuando se crea un nuevo objeto de la clase (usando la palabra clave new). Su propósito principal es inicializar los atributos del objeto. El constructor tiene el mismo nombre que la clase y no tiene un tipo de retorno. Puede haber múltiples constructores con diferentes parámetros (sobrecarga de constructores).

**Definición de métodos**: Los métodos definen el comportamiento de los objetos de la clase. Pueden realizar acciones, manipular los atributos del objeto y comunicarse con otros objetos.

**Declaración de atributos**: Dentro de la clase, se declaran las variables de instancia que representan las características de los objetos de esta clase. Cada objeto tendrá su propio conjunto de valores para estos atributos.

v

**Declaración de atributos (variables de instancia):**

Como vimos en el ejercicio anterior, los atributos son las variables que almacenan los datos asociados con un objeto. Cada objeto de una clase tendrá sus propias copias de estas variables. Se declaran dentro de la clase pero fuera de cualquier método.

**Definición de métodos (funciones dentro de una clase):**

Los métodos son bloques de código que contienen una secuencia de instrucciones para realizar una tarea específica. Dentro de un método, puedes acceder y modificar los atributos del objeto actual (usando la palabra clave this) y realizar otras operaciones.

**El constructor: Inicialización de objetos:**

El constructor es crucial para asegurar que los objetos se creen en un estado válido. Cuando usas new NombreDeLaClase(), se llama al constructor de esa clase. Si no defines explícitamente un constructor, Java proporciona un constructor por defecto sin argumentos que no realiza ninguna inicialización especial.

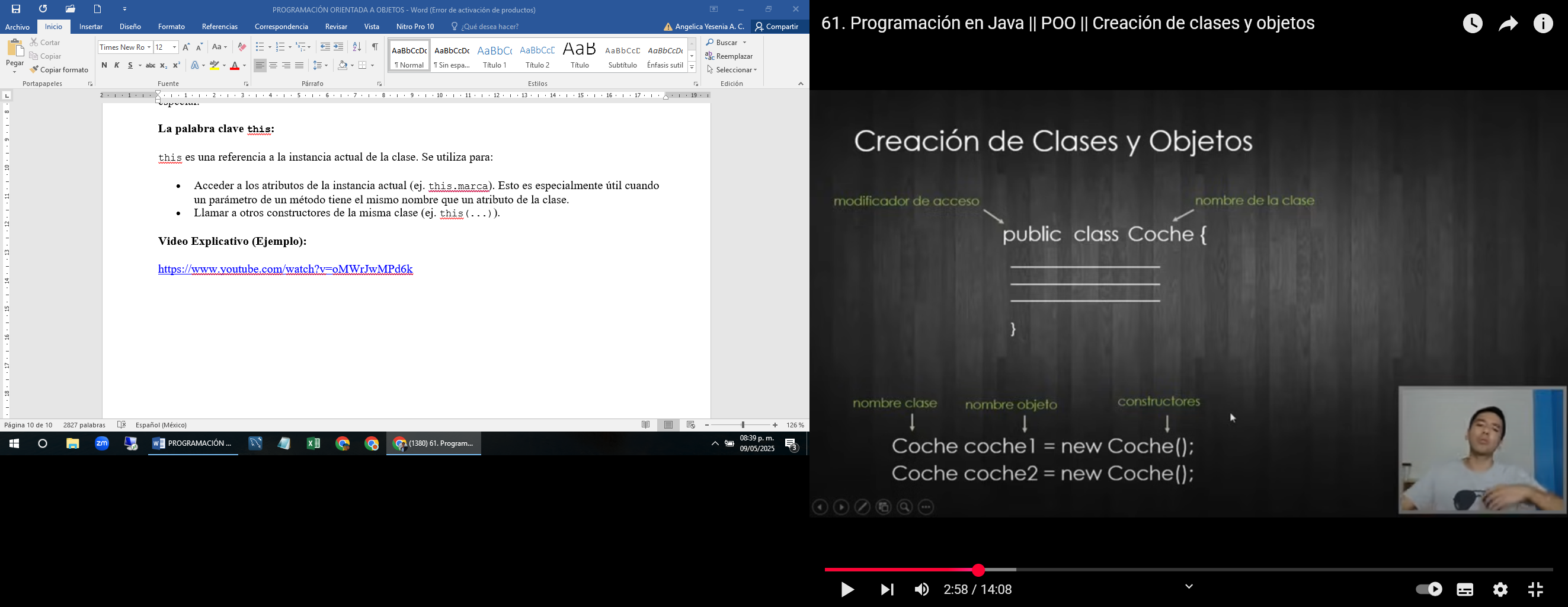
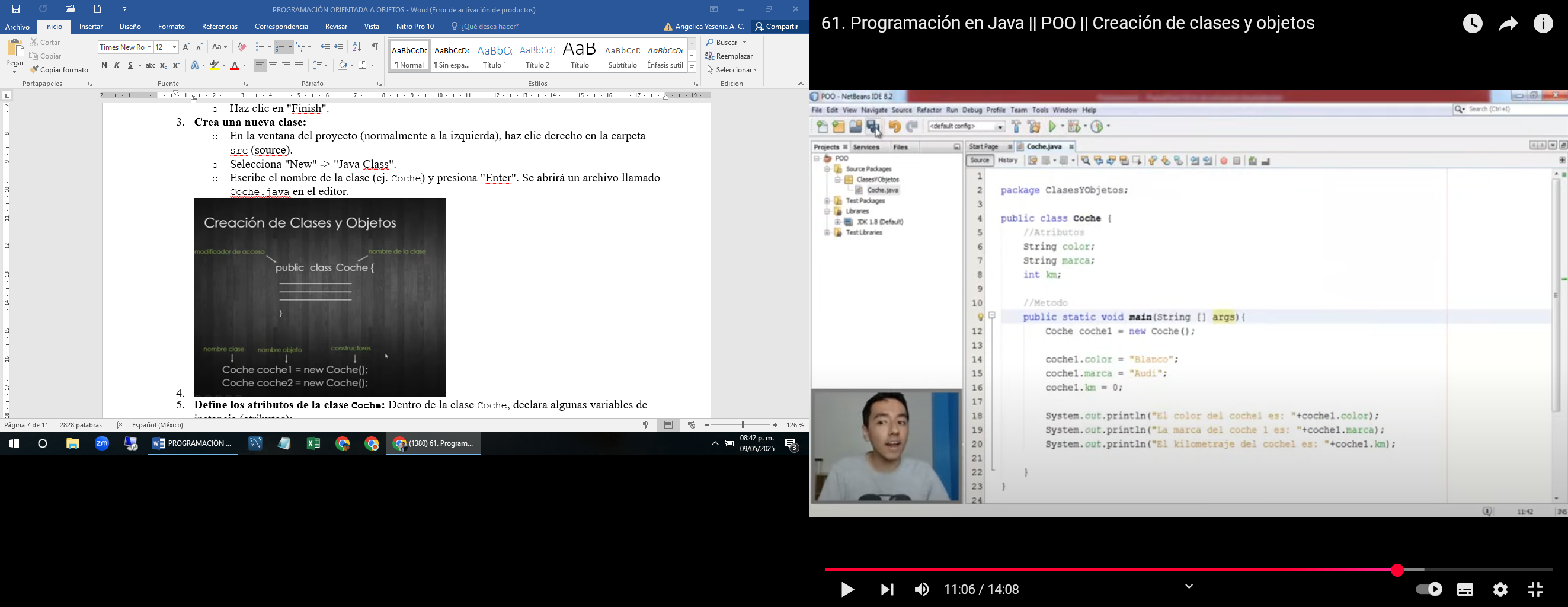
**La palabra clave this:**

this es una referencia a la instancia actual de la clase. Se utiliza para:

* Acceder a los atributos de la instancia actual (ej. this.marca). Esto es especialmente útil cuando un parámetro de un método tiene el mismo nombre que un atributo de la clase.
* Llamar a otros constructores de la misma clase (ej. this(...)).

**Video Explicativo (Ejemplo):**

<https://www.youtube.com/watch?v=oMWrJwMPd6k>

**Ejercicio Práctico 2 en IntelliJ IDEA:**

1. **Abre tu proyecto CursoPOO en IntelliJ IDEA.**
2. **Modifica la clase Coche.java para añadir métodos:**

public class Coche {  
 String marca;  
 String modelo;  
 String color; //Sesión 1 -  
 int velocidadActual;  
  
 //Crea un constructor para la clase Coche que reciba la marca, el modelo y el color como parámetros y  
 // los asigne a los atributos correspondientes, inicializando velocidadActual en 0.  
 // Utiliza la palabra clave this para referirte a los atributos de la clase, para diferenciarlos de los parametros del constructor  
  
 public Coche(String marcaInicial, String modeloInicial, String colorInicial){ //parametros del constructor  
 this.marca = marcaInicial;  
 this.modelo = modeloInicial;  
 this.color = colorInicial;  
 this.velocidadActual = 0;  
 }

//CREAR LOS MÉTODOS acelerar(), frenar(), obtenerVelocidad():  
  
 // Mé*todo para acelerar el coche* public void acelerar(int incremento) {  
 this.velocidadActual += incremento; //+= atajo de escritura: equivale a this.velocidadActual = this.velocidadActual + incremento;.  
 System.*out*.println(this.marca + " " + this.modelo + " acelera a " + this.velocidadActual + " km/h.");  
 }  
  
 // Mé*todo para frenar el coche* public void frenar(int decremento) {  
 if (this.velocidadActual >= decremento) {  
 this.velocidadActual -= decremento;  
 System.*out*.println(this.marca + " " + this.modelo + " frena a " + this.velocidadActual + " km/h.");  
 } else {  
 this.velocidadActual = 0;  
 System.*out*.println(this.marca + " " + this.modelo + " se ha detenido.");  
 }  
 }  
  
 // Mé*todo para obtener la velocidad actual* public int obtenerVelocidad() { //int: El mé*todo devuelve un valor de tipo entero (int), que en este caso es la velocidad.* return this.velocidadActual; //el metodo devuelve o retorna el valor al ser llamado.  
 }  
}

3.- abrir la clase main y agregar los métodos acelerar y frenar para implementarlos en los objetos **coche1** y **coche2**

//lección 2 CREAR 2 OBJETOS coche1 y Coche2 con los atributos creados en el constructor de la clase Coche  
// y utilizar los métodos acelerar y frenar  
Coche coche1 = new Coche("TOYOTA","COROLLA","ROJO");  
System.*out*.println("MI PRIMER COCHE ES UN " + coche1.marca + "- " + coche1.modelo +"- "+ coche1.color);  
coche1.acelerar(20);  
coche1.acelerar(30);  
coche1.frenar(20);  
System.*out*.println("VELOCIDAD ACTUAL ES : " + coche1.obtenerVelocidad() + " km/h.");  
coche1.frenar(40);  
System.*out*.println("LA VELOCIDAD ACTUAL ES" + coche1.obtenerVelocidad()+ " km/h");  
  
Coche coche2 = new Coche("FORD","FOCUS","AZUL");  
System.*out*.println("MI SEGUNDO COCHE ES UN "+ coche2.marca + "- "+ coche2.modelo + "- "+ coche2.color);  
coche2.acelerar(50);  
coche2.acelerar(70);  
coche2.frenar(100);  
System.*out*.println("VELOCIDAD ACTUAL ES : " + coche2.obtenerVelocidad() + " km/h.");  
coche1.frenar(80);  
System.*out*.println("LA VELOCIDAD ACTUAL ES" + coche2.obtenerVelocidad()+ " km/h");

// Método para acelerar

public void acelerar(int incremento) {

velocidadActual += incremento;

System.out.println("El coche " + marca + " acelera a " + velocidadActual + " km/h.");

}

// Método para frenar

public void frenar(int decremento) {

if (velocidadActual >= decremento) {

velocidadActual -= decremento;

System.out.println("El coche " + marca + " frena a " + velocidadActual + " km/h.");

} else {

velocidadActual = 0;

System.out.println("El coche " + marca + " se detiene.");

}

}

}

1. **Modifica la clase Main.java para crear objetos usando el constructor y llamar a los métodos:**

Java

public class Main {

public static void main(String[] args) {

// Crear objetos Coche utilizando el constructor

Coche coche1 = new Coche("Toyota", "Corolla", "Rojo");

Coche coche2 = new Coche("Ford", "Focus", "Azul");

// Llamar a los métodos de los objetos

coche1.acelerar(20);

coche1.acelerar(30);

coche1.frenar(10);

coche2.acelerar(50);

coche2.frenar(60); // Intentará frenar más de lo que va

}

}

1. **Ejecuta el programa Main.java.** Observa la salida en la consola.

En este ejercicio, hemos definido métodos (acelerar y frenar) dentro de la clase Coche que modifican el estado del objeto (el atributo velocidadActual) y muestran un comportamiento. También hemos utilizado un constructor para inicializar los atributos de los objetos al crearlos.

**ARCHIVO JAVA EN LA CLASE COCHE**

**Archivo: Coche.java**

En la ventana del editor de IntelliJ IDEA, el archivo Coche.java debería contener el siguiente código:

public class Coche {

String marca;

String modelo;

String color;

int velocidadActual = 0; // Nuevo atributo

// Constructor

public Coche(String marca, String modelo, String color) {

this.marca = marca;

this.modelo = modelo;

this.color = color;

}

// Método para acelerar

public void acelerar(int incremento) {

velocidadActual += incremento;

System.out.println("El coche " + marca + " acelera a " + velocidadActual + " km/h.");

}

// Método para frenar

public void frenar(int decremento) {

if (velocidadActual >= decremento) {

velocidadActual -= decremento;

System.out.println("El coche " + marca + " frena a " + velocidadActual + " km/h.");

} else {

velocidadActual = 0;

System.out.println("El coche " + marca + " se detiene.");

}

}

}

**Archivo: Main.java**

En otra pestaña del editor, o en un archivo diferente llamado Main.java dentro de la misma carpeta src, deberías tener este código:

public class Main {

public static void main(String[] args) {

// Crear objetos Coche utilizando el constructor

Coche coche1 = new Coche("Toyota", "Corolla", "Rojo");

Coche coche2 = new Coche("Ford", "Focus", "Azul");

// Llamar a los métodos de los objetos

coche1.acelerar(20);

coche1.acelerar(30);

coche1.frenar(10);

coche2.acelerar(50);

coche2.frenar(60); // Intentará frenar más de lo que va

///////////////////////EXPLICACIÓN DEL PROGRAMA////////////////////////

* Este es el **constructor** de la clase Coche. Es un método especial que se llama automáticamente cuando creamos un nuevo objeto de la clase Coche usando la palabra clave new.
  + public Coche(String marca, String modelo, String color):
    - public: Modificador de acceso, indica que el constructor puede ser llamado desde cualquier parte.
    - Coche: El nombre del constructor siempre es el mismo que el nombre de la clase.
    - (String marca, String modelo, String color): Estos son los **parámetros** del constructor. Cuando creas un objeto Coche, debes proporcionar valores para la marca, el modelo y el color.
  + Dentro del constructor, se **inicializan los atributos** del nuevo objeto Coche con los valores que se pasan como argumentos:
    - this.marca = marca;: Asigna el valor del parámetro marca al atributo marca del objeto actual (this).
    - this.modelo = modelo;: Asigna el valor del parámetro modelo al atributo modelo del objeto actual (this).
    - this.color = color;: Asigna el valor del parámetro color al atributo color del objeto actual (this).

Este es un **método** llamado acelerar. Los métodos definen el comportamiento de los objetos de la clase Coche.

* + // Método para acelerar: Comentario que describe la función del método.
  + public void acelerar(int incremento):
    - public: Modificador de acceso, indica que este método puede ser llamado desde cualquier parte.
    - void: Indica que este método no devuelve ningún valor.
    - acelerar: Es el nombre del método. Por convención, los nombres de los métodos en Java comienzan con una letra minúscula y usan "camelCase".
    - (int incremento): Es el **parámetro** del método. Cuando llamas al método acelerar, debes proporcionarle un valor entero que representa cuánto se incrementará la velocidad.
  + Dentro del método acelerar:
    - velocidadActual += incremento;: Incrementa el valor del atributo velocidadActual del objeto actual en la cantidad especificada por el parámetro incremento. Es una forma abreviada de escribir velocidadActual = velocidadActual + incremento;.
    - System.out.println("El coche " + marca + " acelera a " + velocidadActual + " km/h.");: Imprime un mensaje en la consola indicando la marca del coche y su nueva }

Este es otro **método** llamado frenar.

* + // Método para frenar: Comentario que describe la función del método.
  + public void frenar(int decremento):
    - public: Modificador de acceso.
    - void: Indica que el método no devuelve ningún valor.
    - frenar: Nombre del método.
    - (int decremento): Parámetro que indica cuánto se reducirá la velocidad.
    - Las llaves { y } delimitan el cuerpo del método.
  + Dentro del método frenar:
    - if (velocidadActual >= decremento): Se verifica si la velocidad actual es mayor o igual a la cantidad que se quiere decrementar.
      * Si es verdadero, se ejecuta el bloque if:
        + velocidadActual -= decremento;: Decrementa el valor del atributo velocidadActual en la cantidad especificada por decremento (equivalente a velocidadActual = velocidadActual - decremento;).
        + System.out.println("El coche " + marca + " frena a " + velocidadActual + " km/h.");: Imprime un mensaje con la nueva velocidad.
      * Si la condición del if es falsa (la velocidad actual es menor que la cantidad a decrementar), se ejecuta el bloque else:
        + velocidadActual = 0;: Establece la velocidad actual a cero (el coche se detiene).
        + System.out.println("El coche " + marca + " se detiene.");: Imprime un mensaje indicando que el coche se ha detenido.

**EJECUTAR EL PROGRAMA :**

IntelliJ IDEA compilará y ejecutará el programa Main. Deberías ver la siguiente salida en la ventana de la consola "Run" (normalmente ubicada en la parte inferior de la ventana de IntelliJ IDEA):

El coche Toyota acelera a 20 km/h.

El coche Toyota acelera a 50 km/h.

El coche Toyota frena a 40 km/h.

El coche Ford acelera a 50 km/h.

El coche Ford se detiene.

Esta salida muestra cómo los objetos coche1 y coche2 interactúan con los métodos acelerar y frenar, modificando su estado interno y mostrando mensajes en la consola.

<https://www.youtube.com/watch?v=hDPSRbXBi8U>

**Módulo 2: Los Pilares de la POO - Encapsulamiento**

**Sesión Teórica 3: Encapsulamiento (Modificadores de acceso, ocultación de datos)**

El **encapsulamiento** es uno de los principios clave de la POO. Consiste en **ocultar el estado interno (los atributos) de un objeto y exponer solo los mecanismos necesarios (los métodos públicos) para interactuar con él**. Piensa en un coche real: como conductor, interactúas con el volante, los pedales y la palanca de cambios (la interfaz pública), pero no necesitas saber cómo funciona internamente el motor, la transmisión o el sistema de frenos (el estado interno oculto).

**¿Por qué es importante el encapsulamiento?**

* **Control de acceso:** Permite controlar cómo se accede y se modifica el estado de un objeto desde fuera de la clase. Esto ayuda a prevenir modificaciones accidentales o no deseadas que podrían llevar a un estado inconsistente del objeto.
* **Integridad de los datos:** Al controlar el acceso a los atributos a través de métodos específicos, podemos implementar lógica de validación para asegurar que los datos siempre sean válidos. Por ejemplo, podríamos evitar que la velocidad de un coche sea negativa.
* **Ocultación de la complejidad:** El encapsulamiento oculta los detalles internos de implementación de una clase. Esto hace que el código sea más fácil de entender y usar, ya que los usuarios del objeto solo necesitan conocer su interfaz pública (los métodos).
* **Flexibilidad y mantenibilidad:** Si la implementación interna de una clase cambia (por ejemplo, se optimiza el funcionamiento de un método o se cambia la forma en que se almacenan los datos), mientras la interfaz pública (los nombres y parámetros de los métodos) se mantenga igual, el código que utiliza esa clase no se verá afectado. Esto facilita la evolución y el mantenimiento del software.

**Modificadores de acceso:**

Java proporciona modificadores de acceso para controlar la visibilidad de los atributos y métodos de una clase. Los más comunes son:

* **public:** Los atributos o métodos declarados como public son accesibles desde cualquier parte del programa.
* **private:** Los atributos o métodos declarados como private solo son accesibles desde dentro de la misma clase en la que se declaran. Esto es el núcleo de la ocultación de datos.
* **protected:** Los atributos o métodos declarados como protected son accesibles desde la misma clase, desde las clases hijas (subclases) y desde otras clases dentro del mismo paquete.
* **(Sin modificador):** Si no se especifica ningún modificador de acceso (también conocido como "package-private" o "friendly"), el acceso está limitado a las clases dentro del mismo paquete.

**Ocultación de datos: Hacer los atributos private:**

La práctica más común para lograr el encapsulamiento es declarar los atributos de una clase como private. Esto significa que ningún código fuera de la clase Coche (por ejemplo, desde la clase Main) puede acceder o modificar directamente los atributos marca, modelo, color o velocidadActual.

**Métodos getter y setter:**

Para permitir el acceso controlado a los atributos private, se utilizan los métodos getter (para obtener el valor de un atributo) y setter (para modificar el valor de un atributo).

* Un **getter** es un método público que devuelve el valor de un atributo privado. Por convención, su nombre suele ser getNombreDelAtributo (ej., getMarca()).
* Un **setter** es un método público que permite modificar el valor de un atributo privado. Por convención, su nombre suele ser setNombreDelAtributo(Tipo nuevoValor) (ej., setVelocidadActual(int nuevaVelocidad)). Dentro del setter, podemos incluir lógica de validación para asegurar que el nuevo valor sea válido.