INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PARADIGMAS DE PROGRAMACIÓN

PRÁCTICA 8

**Fundamento de POO\_ Polimorfismo**

PRESENTA:

**Cruz Trejo Said Carbot**

**Torres Cárdenas Elian Alejandro**

GRUPO:

**3BV2**

PROFESOR: **ANDRES GARCIA FLORIANO**

**Introducción**

La práctica tiene como objetivo principal explorar el concepto de polimorfismo en programación orientada a objetos mediante el diseño de un sistema de gestión de vehículos. Este sistema está diseñado para modelar diferentes tipos de vehículos con comportamientos específicos, utilizando clases abstractas como base. La clase abstracta actúa como una plantilla que define los atributos y métodos esenciales que deben implementar las clases derivadas. De esta forma, se asegura una estructura común y se permite que cada tipo de vehículo adapte su comportamiento según sus particularidades.

El enfoque de la práctica se centra en la creación de una jerarquía de clases que aproveche el polimorfismo. Esto permite interactuar con objetos de diferentes tipos de vehículos de manera uniforme, independientemente de su clase específica. Al implementar métodos con el mismo nombre en las clases derivadas, el sistema garantiza que las operaciones principales, como encender, apagar y describir el vehículo, sean accesibles de forma consistente, promoviendo así un diseño más limpio y flexible.

Además, esta práctica resalta la importancia de la reutilización de código y la extensibilidad en sistemas complejos. Por ejemplo, la estructura diseñada facilita la adición de nuevos tipos de vehículos sin necesidad de modificar el código existente, cumpliendo con el principio de diseño abierto/cerrado. En este contexto, el polimorfismo permite gestionar una colección de objetos heterogéneos de manera eficiente, simplificando el desarrollo y mantenimiento del sistema. La actividad incluye una reflexión sobre los beneficios del polimorfismo y cómo este enfoque ayuda a manejar la diversidad de objetos en el software. Se analiza también cómo el uso de clases abstractas fomenta la modularidad y la coherencia, promoviendo un diseño de software robusto y escalable.

**Enlace de la carpeta del GitHub de Elian con la práctica 7:** [**Elian-pixel/ParadigmasProg**](https://github.com/Elian-pixel/ParadigmasProg)

**Enlace de la carpeta del GitHub de Said con la práctica 8:** [**https://github.com/SaidCarbot/paradigmas/tree/main/practica8**](https://github.com/SaidCarbot/paradigmas/tree/main/practica8)

**JAVA**

**PYTHON**

El polimorfismo es un concepto fundamental en la programación orientada a objetos que permite que diferentes clases puedan ser tratadas de manera uniforme, es decir, que objetos de diferentes tipos puedan ser manipulados a través de una interfaz común. En Python, el polimorfismo se logra principalmente mediante la herencia y la implementación de métodos con el mismo nombre en diferentes clases.

En términos generales, el polimorfismo permite que se puedan usar objetos de diferentes tipos de manera intercambiable, siempre que compartan una misma interfaz, sin necesidad de saber de antemano qué tipo exacto de objeto se está utilizando. Esto es particularmente útil cuando se trabaja con colecciones de objetos que pueden pertenecer a diferentes clases pero tienen comportamientos comunes.

En Python, el polimorfismo es posible debido a su modelo de tipado dinámico. A diferencia de otros lenguajes como Java o C++, Python no requiere declarar explícitamente los tipos de las variables, lo que permite que una función o un método acepte diferentes tipos de objetos, siempre que esos objetos implementen los métodos o atributos necesarios. Esto se conoce como "polimorfismo implícito".

En Python, se pueden tener varias clases que implementen métodos con el mismo nombre, y luego, cuando se interactúa con un objeto, se llama a ese método sin necesidad de conocer su tipo exacto.

**Polimorfismo y herencia:**

El polimorfismo se asocia frecuentemente con la herencia. En el caso de una clase base (o clase padre) que define un método, las clases derivadas pueden sobrescribir ese método para proporcionar una implementación específica. El método sobrescrito en la clase hija es llamado en tiempo de ejecución, lo que significa que el comportamiento depende del tipo de objeto.

**Polimorfismo en funciones:**

Otra forma en que el polimorfismo puede manifestarse en Python es mediante el uso de funciones que aceptan objetos de diferentes clases, siempre que estos objetos compartan ciertos métodos o atributos. Python no obliga a que las clases sigan una jerarquía rígida (como en otros lenguajes), lo que permite una forma más flexible de polimorfismo.

**Beneficios del polimorfismo:**

* **Facilita la reutilización de código**: Con polimorfismo, se pueden escribir funciones o métodos que trabajen con distintos tipos de objetos sin tener que preocuparse por el tipo específico de cada uno.
* **Mejora la extensibilidad**: El código es más fácil de extender. Se pueden agregar nuevas clases sin necesidad de modificar las funciones que utilizan polimorfismo, lo que promueve el principio de diseño de software de abierto/cerrado.
* **Mayor flexibilidad**: Las funciones que aceptan objetos de diferentes clases pueden funcionar con cualquiera de esos objetos sin tener que sobrecargar la función o usar una lógica compleja para diferenciar entre tipos [1].

**Descripción de la práctica**

**Sistema de Gestión de Vehículos**

**Objetivo:** Evaluar el concepto de **polimorfismo** y el uso de **clases abstractas** mediante el diseño y desarrollo de un sistema que gestione vehículos de diferentes tipos, cada uno con comportamientos específicos.

Instrucciones:

1. **Definir la Clase Abstracta Vehiculo:**
   * Crea una clase abstracta llamada Vehiculo que represente un vehículo genérico.
   * Atributos:
     + marca (tipo String)
     + modelo (tipo String)
     + año (tipo int)
   * Métodos:
     + encender(): método abstracto que simule el encendido del vehículo.
     + apagar(): método abstracto que simule el apagado del vehículo.
     + describir(): método abstracto que muestre una breve descripción del vehículo.
   * **Nota:** La clase Vehiculo no debe poder instanciarse directamente; solo sus clases derivadas podrán hacerlo.
2. **Crear Clases Derivadas de Vehiculo:**
   * Define las siguientes clases que hereden de Vehiculo e implementen los métodos abstractos:

**a. Auto:**

* + Representa un automóvil convencional.
  + Implementa encender() para simular el encendido de un auto.
  + Implementa apagar() para simular el apagado del auto.
  + Implementa describir() para mostrar información como "Este es un auto de marca X, modelo Y, del año Z".

**b. Motocicleta:**

* + Representa una motocicleta.
  + Implementa encender() para simular el encendido de una motocicleta.
  + Implementa apagar() para simular el apagado de la motocicleta.
  + Implementa describir() para mostrar información como "Esta es una motocicleta de marca X, modelo Y, del año Z".

**c. Camion:**

* + Representa un camión de carga.
  + Implementa encender() para simular el encendido de un camión.
  + Implementa apagar() para simular el apagado del camión.
  + Implementa describir() para mostrar información como "Este es un camión de marca X, modelo Y, del año Z".

1. **Implementar el Programa Principal:**
   * Crea un programa que realice las siguientes operaciones:
     + Define una lista de vehículos que incluya al menos un Auto, una Motocicleta y un Camion.
     + Para cada vehículo en la lista, llama a los métodos encender(), describir() y apagar() utilizando polimorfismo.
2. **Preguntas de reflexión:**
   * ¿Por qué crees que Vehiculo es una clase abstracta en lugar de una clase concreta?
   * ¿Qué ventaja ofrece el polimorfismo en este diseño cuando se trabaja con una lista de distintos tipos de vehículos?
   * ¿Qué pasaría si se quiere añadir otro tipo de vehículo (por ejemplo, BicicletaEléctrica)? ¿Cómo ayuda la estructura actual a facilitar la expansión del programa?

**Desarrollo en Java**

**Desarrollo en Python**

El código define una jerarquía de clases utilizando programación orientada a objetos con clases abstractas e implementación de polimorfismo. En primer lugar, se crea una clase abstracta llamada Vehiculo, la cual representa un vehículo genérico. Esta clase contiene tres métodos abstractos: encender(), apagar(), y describir(), los cuales deben ser implementados por cualquier clase que herede de Vehiculo. La clase Vehiculo también tiene un constructor que inicializa tres atributos: marca, modelo y año, los cuales son comunes para todos los vehículos.

Luego, se definen tres clases concretas: Auto, Motocicleta y Camion, que heredan de Vehiculo. Cada una de estas clases implementa los métodos abstractos encender(), apagar() y describir() con comportamientos específicos para cada tipo de vehículo. Por ejemplo, la clase Auto tiene un método encender() que simula el encendido de un automóvil, y un método describir() que proporciona detalles específicos sobre el auto, como su marca, modelo y año.

El programa principal crea una lista de objetos de diferentes tipos de vehículos, incluyendo un Auto, una Motocicleta y un Camion. A continuación, se itera sobre esta lista, llamando a los métodos encender(), describir() y apagar() para cada vehículo, lo que demuestra el uso del polimorfismo. El polimorfismo permite que, sin importar el tipo de vehículo, se puedan llamar los métodos de manera uniforme y ejecutar comportamientos específicos dependiendo del tipo de objeto que se encuentre en la lista.

Este diseño facilita la extensión del sistema. Si se desea añadir un nuevo tipo de vehículo, como una BicicletaEléctrica, solo es necesario crear una nueva clase que herede de Vehiculo e implemente los métodos requeridos. El código existente no necesita modificaciones, lo que sigue el principio abierto/cerrado en el diseño de software, permitiendo la expansión del sistema sin alterar la funcionalidad ya implementada.

En resumen, el uso de clases abstractas y polimorfismo permite gestionar una colección de vehículos de diferentes tipos de manera eficiente, mejorando la escalabilidad, reutilización y mantenimiento del código.

**Casos de uso**

**Casos de uso en Java**

Ilustración 1 Sección uno en JAVA,

Elaboración propia con datos del código.

**Casos de uso en Python**

Ilustración 2 Sección uno en Phyton, definir clase abstracta vehículo

****

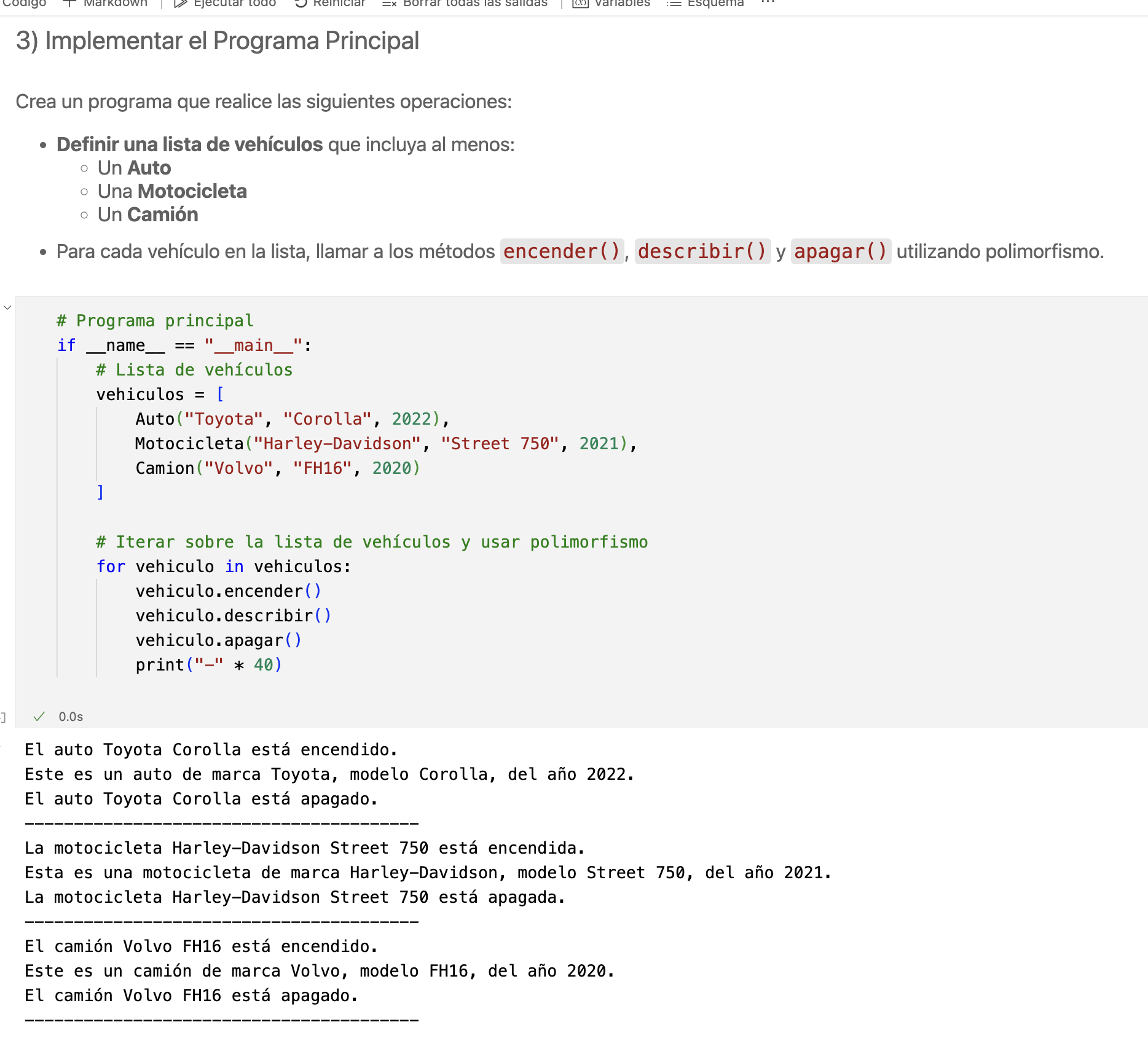
Elaboración propia con datos del código.

Ilustración 3 Sección dos en Phyton, crear clases derivadas de vehículo



Elaboración propia con datos del código.

Ilustración 4 Sección tres en Phyton, main



Elaboración propia con datos del código.

**Conclusiones**

La implementación de polimorfismo mediante clases abstractas en un sistema de gestión de vehículos demuestra ser una herramienta poderosa para la programación orientada a objetos. Este enfoque permite gestionar de manera uniforme objetos de diferentes tipos, promoviendo la reutilización de código y la extensibilidad del sistema.

El uso de una clase abstracta como base asegura que todas las clases derivadas compartan una estructura y comportamiento común, al tiempo que posibilita la personalización de las funciones específicas de cada tipo de vehículo. Esto simplifica el diseño, facilita el mantenimiento y refuerza la adherencia a principios de diseño de software como el abierto/cerrado.

Además, el polimorfismo potencia la flexibilidad del sistema al permitir que las operaciones se realicen sin necesidad de conocer de antemano el tipo exacto de cada objeto. Este enfoque reduce la complejidad del código y evita redundancias al trabajar con colecciones heterogéneas de objetos.

Por último, la estructura desarrollada asegura la escalabilidad del sistema, ya que permite integrar nuevos tipos de vehículos de manera sencilla y sin alterar la lógica existente. Esto no solo mejora la calidad del código, sino que también optimiza el tiempo de desarrollo y la adaptabilidad del software a futuras necesidades.

**Referencias**

|  |  |
| --- | --- |
| [1] | ORACLE, «Lesson: Interfaces and Inheritance,» s.f. s.f. 2024. [En línea]. Available: https://docs.oracle.com/javase/tutorial/java/IandI/index.html. [Último acceso: noviembre 2024]. |
| [2] | myCompiler, «myCompiler,» s.f. s.f. 2024. [En línea]. Available: https://www.mycompiler.io/es/new/java. [Último acceso: noviembre 2024]. |