UNIVERSIDAD AMERICANA

****

Metodología y Programación Orientada a Objetos

**Caso de Estudio**

**Estudiantes:**

Nathaly Verónica Alvarado Rodríguez

Ana Sofia Fuentes Valladares

Kelly Xaviera Fuentes Areas

**Docente:**

Norman Jose Cash Arcia

# 

# 

# *29 de Septiembre 2025*

**1. Análisis del Código Base:**

* ***Herencia***  
  La clase Vehiculo es abstracta y contiene atributos comunes como marca, modelo y año. Tanto Automovil como Bicicleta heredan de esta clase, lo que les permite reutilizar dichos atributos y el método obtenerDetalles(). Además, cada una puede añadir características propias, como el nivel de combustible en el automóvil o el número de marchas en la bicicleta. Gracias a la herencia, se evita repetir código y se mantiene una estructura más ordenada.
* ***Polimorfismo***  
  Se observa cuando se utiliza una variable del tipo Vehiculo para almacenar objetos de las subclases. Por ejemplo, si se invoca el método mover() sobre una referencia de tipo Vehiculo, en tiempo de ejecución se ejecuta la versión correspondiente a la clase concreta (Automovil o Bicicleta). Lo mismo ocurre con obtenerDetalles(). Esto permite tratar a los distintos vehículos de forma uniforme, aunque cada uno tenga su propio comportamiento.
* ***Uso de super***  
  La palabra clave super se utiliza en los constructores de las subclases para llamar al constructor de la clase padre e inicializar los atributos heredados. También se usa en el método obtenerDetalles() de Automovil para reutilizar la información de la clase Vehiculo y luego añadir datos adicionales, como el nivel de combustible. De esta forma, se evita duplicar código y se aprovecha lo ya definido en la clase base.
* ***Interfaz Combustible***  
  La interfaz Combustible establece un contrato para las clases que necesiten manejar operaciones relacionadas con combustible, como recargar y consultar el nivel disponible. En este caso, únicamente Automovil la implementa, ya que en el ejemplo solo los automóviles requieren gestionar un tanque de combustible. La bicicleta no la implementa porque no necesita esas operaciones. En un futuro, otros vehículos como motocicletas podrían implementarla también.

**5. Reflexión Escrita**

El polimorfismo permite desarrollar sistemas más flexibles y fáciles de ampliar. Gracias a él, distintas clases pueden compartir una misma interfaz o clase base, pero comportarse de forma diferente al ejecutar métodos comunes. En una aplicación para gestionar vehículos, por ejemplo, se pueden añadir nuevas clases como Motocicleta, Camión o Autobús sin modificar la lógica que procesa la flota. Basta con que hereden de Vehiculo y redefinan métodos como mover() u obtenerDetalles() según sus características. Además, las interfaces como Combustible o Mantenimiento permiten asignar funciones específicas solo a las clases que realmente las necesitan. Esto mantiene el sistema organizado, claro y fácil de mantener. Los cambios se aplican de forma localizada, sin afectar el resto del código, y la expansión se logra añadiendo nuevas clases en lugar de modificar las existentes. Así, el diseño se vuelve más robusto, limpio y preparado para crecer sin complicaciones.