POLIMORFISMO

Estructura de la clase

1. Introducción

- **Definición de Polimorfismo**:

  Explica qué es el polimorfismo en términos simples: "es la capacidad que tienen los objetos de diferentes clases de responder al mismo mensaje (o método) de diferentes maneras".

  Muestra cómo el polimorfismo es uno de los pilares fundamentales de la Programación Orientada a Objetos (POO).

- **Objetivo de la clase**: Que los estudiantes entiendan qué es el polimorfismo, por qué es importante en el diseño de software y cómo se aplica en la programación.

Subtemas a cubrir:

- ¿Por qué necesitamos polimorfismo?
- Diferencia entre **sobrecarga** y **sobreescritura**.
- Tipos de polimorfismo:

  Polimorfismo en tiempo de compilación (sobrecarga de métodos, operadores).

  Polimorfismo en tiempo de ejecución (herencia y sobreescritura de métodos).

2. Conceptos clave

- **Herencia**: Recordar a los estudiantes que el polimorfismo depende de la **herencia**. Reafirma la relación entre las clases base y derivadas. Explica cómo una clase derivada puede redefinir métodos de la clase base.
- **Sobreescritura de métodos**: Este es el mecanismo que permite el polimorfismo en tiempo de ejecución. Se debe mostrar cómo una subclase puede redefinir un método de la clase base.

Ejemplo:

```
Clase Animal:
  - método: hacer_sonido()

Clase Perro (hereda de Animal):
  - método sobrescrito: hacer_sonido() => "El perro ladra"

Clase Gato (hereda de Animal):
  - método sobrescrito: hacer_sonido() => "El gato maúlla"
```

3. Teoría: Polimorfismo en Tiempo de Compilación vs. Tiempo de Ejecución

- **Polimorfismo en tiempo de compilación**: Explica que se refiere a **sobrecarga de métodos**, que permite definir varios métodos con el mismo nombre pero diferentes firmas (distintos tipos o número de parámetros).
- **Polimorfismo en tiempo de ejecución**:
  Detalla cómo ocurre a través de la **sobreescritura de métodos**. Explica que en este caso, la decisión sobre qué método se invoca ocurre durante la ejecución del programa, dependiendo del tipo real del objeto.

4. Ejemplos en Código

- **Código base**: Presenta un ejemplo de una clase base con un método genérico, y varias clases derivadas que sobrescriben ese método, demostrando polimorfismo en tiempo de ejecución.

Ejemplo en Python:

```python
# Clase base
class Entidad:
    def atacar(self):
        pass  # Método base que será sobrescrito por las subclases

# Subclase Zombi
class Zombi(Entidad):
    def atacar(self):
        return "El zombi ataca mordiendo."

# Subclase Esqueleto
class Esqueleto(Entidad):
    def atacar(self):
        return "El esqueleto ataca disparando flechas."

# Subclase Humano
class Humano(Entidad):
    def __init__(self, tiene_arma):
        self.tiene_arma = tiene_arma  # True si tiene un arma, False si no

    def atacar(self):
        if self.tiene_arma:
            return "El humano ataca con un arma."
        else:
            return "El humano ataca con las manos."

# Función que demuestra el polimorfismo
def entidad_ataca(entidad):
    print(entidad.atacar())

# Crear instancias de Zombi, Esqueleto y Humano
zombi = Zombi()
esqueleto = Esqueleto()
humano_con_arma = Humano(tiene_arma=True)  # El humano tiene un arma
humano_sin_arma = Humano(tiene_arma=False) # El humano no tiene un arma

# Llamar la función polimórfica
entidad_ataca(zombi)               # El zombi ataca mordiendo.
entidad_ataca(esqueleto)           # El esqueleto ataca disparando flechas.
entidad_ataca(humano_con_arma)     # El humano ataca con un arma.
entidad_ataca(humano_sin_arma)     # El humano ataca con las manos.
```

```
El zombi ataca mordiendo.
El esqueleto ataca disparando flechas.
El humano ataca con un arma.
El humano ataca con las manos.
```

5. Ventajas del Polimorfismo

- **Flexibilidad**: Permite que el código sea más flexible y adaptable a futuros cambios. Si se agregan más personajes en el futuro, solo necesitas crear nuevas clases derivadas sin modificar el código existente.
- **Reutilización del código**: Los métodos genéricos como realizar acción(personaje) pueden operar sobre cualquier subclase de Personaje, sin necesidad de duplicar código.
- **Escalabilidad**: El código polimórfico es más fácil de escalar, ya que agregar nuevos comportamientos no requiere modificar los métodos generales, solo agregar nuevas subclases.

6. Actividades Prácticas

- **Ejercicio 1**: Pide a los estudiantes que creen una clase “base” Vehículo con un método mover(), y que luego creen subclases “Coche, Bicicleta, y Avión” que sobrescriban el método mover() con comportamientos específicos.

7. Comparación con Otros Conceptos

- **Herencia vs. Polimorfismo**: Resalta las diferencias y similitudes. Aunque el polimorfismo depende de la herencia, no es lo mismo. Herencia permite que una subclase herede propiedades y métodos de una clase base, mientras que el polimorfismo permite que las subclases reinterpreten esos métodos de maneras diferentes.

8. Polimorfismo en Otros Lenguajes

- Muestra ejemplos del uso de polimorfismo en otros lenguajes como Java o C++ para que los estudiantes entiendan cómo se implementa de manera general.

9. Conclusión

- Resume los conceptos clave aprendidos sobre polimorfismo.
- Refuerza la importancia de este concepto en el desarrollo de software orientado a objetos.
- Motiva a los estudiantes a identificar oportunidades para aplicar polimorfismo en sus propios proyectos.