GUÍA #3 HERENCIA Y POLIMORFISMO

JEISON AFRICANO MARTINEZ

JOSEPH IMANOL REYES CHAPARRO

UNIVERSIDAD MANUELA BELTRAN

INGENIERIA DE SOFTWARE

PROGRAMACION ORIENTADA A OBJETOS

INGENIERA: DIANA MARCELA TOQUICA RODRÍGUEZ

BOGOTA D.C, COLOMBIA

10 DE SEPTIEMBRE DE 2023

INTRODUCCION

En los dos siguientes programas daremos a conocer los conceptos de herencia y polimorfismo en el lenguaje Python, en el primer código crearemos una estructura de clases para representar diferentes tipos de vehículos, en la segunda parte del programa se construirá un sistema de clases utilizando la herencia para interpretar distintos roles en una universidad, como estudiante docente y administrativo .

DESCRIPCIÓN DEL PROGRAMA

Programa 1:

En este programa se realiza la construcción de clases para representar diferentes tipos de vehículos

Clase Base : Comienza con una clase base llamada Vehiculo, que contiene atributos comunes a todos los vehículos, como la marca, el modelo y el año. También implementa un método llamado mostrar Información() que muestra la información básica del vehículo.

Clases Derivadas Automovil y Motocicleta: Luego, se crean dos clases derivadas, Automovil y Motocicleta, cada una con atributos adicionales relevantes para su tipo de vehículo (por ejemplo, número de puertas para automóviles y cilindrada para motocicletas.

Constructores: Se crean constructores para todas las clases (incluyendo la clase base y las clases derivadas) para permitir la inicialización de los atributo.

Método Principal: En el método principal (if __name__ == "__main__":), se crean instancias de las clases derivadas, se establecen sus atributos y se llama al método mostrarInformacion() para cada instancia. Esto muestra cómo se heredan los atributos y comportamientos de la clase base y cómo se pueden personalizar en las clases derivadas.

Polimorfismo : El polimorfismo se demuestra cuando el método mostrarInformacion() se llama en instancias de las clases derivadas, lo que permite que el comportamiento del método sea específico para cada tipo de vehículo.

Programa 2:

Este programa crea una jerarquía de clases para modelar diferentes roles en una universidad, como Estudiante, Docente y Administrativo.

Clase Base Persona: Comienza con una clase base llamada Persona que contiene atributos generales como nombre, apellido y edad, junto con un método para consultar la información personal de una persona.

Clases Derivadas Estudiante, Docente y Administrativo: Luego, se crean tres clases derivadas, cada una con atributos adicionales relevantes para su rol en la universidad.

Constructores: Se crean constructores para todas las clases (incluyendo la clase base y las clases derivadas) para permitir la inicialización de los atributos.

Método Principal: En el método principal, se crean instancias de las clases derivadas, se configuran sus atributos y se llama al método.

EXPLICACIÓN DEL PROCESO

 # Definición de la clase base Vehiculo

```
class Vehiculo:

    def _init_(self, marca, modelo, año):

        self.marca = marca

        self.modelo = modelo

        self.año = año



    def mostrarInfo(self):

        print("Marca:", self.marca)

        print("Modelo:", self.modelo)
```

```
        print("Año:", self.año)


# Definición de la clase derivada Coche, que hereda de Vehiculo
class Coche(Vehiculo):
    def _init_(self, marca, modelo, año, numeroPuertas):
        # Llamar al constructor de la clase base usando super()
        super().__init__(marca, modelo, año)
        self.numeroPuertas = numeroPuertas

    def mostrarInfo(self):
        # Llamar al método mostrarInfo de la clase base usando super()
        super().mostrarInfo()
        print("Número de Puertas:", self.numeroPuertas)

# Definición de la clase derivada Moto, que hereda de Vehiculo
class Moto(Vehiculo):
    def _init_(self, marca, modelo, año, cilindrada):
        # Llamar al constructor de la clase base usando super()
        super().__init__(marca, modelo, año)
        self.cilindrada = cilindrada

    def mostrarInfo(self):
        # Llamar al método mostrarInfo de la clase base usando super()
        super().mostrarInfo()
```

```
        print("Cilindrada:", self.cilindrada, "cc")
```

 # Bloque principal del programa

 # Crear instancias de Coche y Moto con nueva información

```
coche1 = Coche("Ford", "Focus", 2023, 5)  # Cambiar la información del coche 1
moto1 = Moto("Suzuki", "GSX-R750", 2022, 750)  # Cambiar la información de la moto 1
```

 # Llamar al método mostrarInfo() para cada instancia

```
print("Información del Coche 1:")
coche1.mostrarInfo()

print("\nInformación de la Moto 1:")
moto1.mostrarInfo()
```

segundo código

Definición de la clase abstracta Persona

```
class Persona:
    def _init_(self, nombre, apellidos, direccion, tipo_id, nro_id):
        self.nombre = nombre
        self.apellidos = apellidos
        self.direccion = direccion
        self.tipo_id = tipo_id
        self.nro_id = nro_id
```

```
    def consultar_info_personal(self):
        pass  # Este método es abstracto y se implementará en las clases hijas

# Definición de la clase Estudiante que hereda de Persona
class Estudiante(Persona):
    def _init_(self, nombre, apellidos, direccion, tipo_id, nro_id, codigo):
        super().__init__(nombre, apellidos, direccion, tipo_id, nro_id)
        self.codigo = codigo

    def consultar_info_personal(self):
        # Método para consultar información personal de un estudiante
        return f"Nombre: {self.nombre} {self.apellidos}, Código: {self.codigo}, Tipo de ID: {self.tipo_id}, Número de ID: {self.nro_id}"

# Definición de la clase Docente que hereda de Persona
class Docente(Persona):
    def _init_(self, nombre, apellidos, direccion, tipo_id, nro_id, escalafon):
        super().__init__(nombre, apellidos, direccion, tipo_id, nro_id)
        self.escalafon = escalafon

    def consultar_info_personal(self):
        # Método para consultar información personal de un docente
        return f"Nombre: {self.nombre} {self.apellidos}, Escalafón: {self.escalafon}, Tipo de ID: {self.tipo_id}, Número de ID: {self.nro_id}"
```

```
# Definición de la clase Administrativo que hereda de Persona
class Administrativo(Persona):
    def _init_(self, nombre, apellidos, direccion, tipo_id, nro_id, salario):
        super().__init__(nombre, apellidos, direccion, tipo_id, nro_id)
        self.salario = salario

    def consultar_info_personal(self):
        # Método para consultar información personal de un administrativo
        return f"Nombre: {self.nombre} {self.apellidos}, Salario: {self.salario}, Tipo de ID: {self.tipo_id}, Número de ID: {self.nro_id}"

# Función principal (main) para probar las clases

estudiantes = []
docentes = []
administrativos = []

while True:
    print("Bienvenido a la base de datos UMB")
    print("¿Qué deseas hacer?")
    print("1. Agregar un Estudiante")
    print("2. Agregar un Docente")
    print("3. Agregar un Administrativo")
    print("4. Ver la información de algún Estudiante, Docente, o Administrativo")
```

```
print("5. Salir")


menu = int(input("Selecciona una opción: "))


if menu == 1:
    print("Agregar un Estudiante")
    nombre = input("Nombre: ")
    apellidos = input("Apellidos: ")
    direccion = input("Dirección: ")
    tipo_id = input("Tipo de ID: ")
    nro_id = int(input("Número de ID: "))
    codigo = int(input("Código del Estudiante: "))
    estudiante = Estudiante(nombre, apellidos, direccion, tipo_id, nro_id, codigo)
    estudiantes.append(estudiante)
    print("Estudiante agregado correctamente.")


elif menu == 2:
    print("Agregar un Docente")
    nombre = input("Nombre: ")
    apellidos = input("Apellidos: ")
    direccion = input("Dirección: ")
    tipo_id = input("Tipo de ID: ")
    nro_id = int(input("Número de ID: "))
    escalafon = input("Escalafón del docente: ")
```

```
        docente = Docente(nombre, apellidos, direccion, tipo_id, nro_id, escalafon)
        docentes.append(docente)
        print("Docente agregado correctamente.")

    elif menu == 3:
        print("Agregar un Administrativo")
        nombre = input("Nombre: ")
        apellidos = input("Apellidos: ")
        direccion = input("Dirección: ")
        tipo_id = input("Tipo de ID: ")
        nro_id = int(input("Número de ID: "))
        salario = int(input("Salario del Administrativo: "))
        administrativo = Administrativo(nombre, apellidos, direccion, tipo_id, nro_id, salario)
        administrativos.append(administrativo)
        print("Administrativo agregado correctamente.")

    elif menu == 4:
        print("Ver la información de un Estudiante, Docente, o Administrativo")
        print("Selecciona el tipo de persona:")
        print("1. Estudiante")
        print("2. Docente")
        print("3. Administrativo")
        tipo_persona = int(input("Selecciona una opción: "))
        if tipo_persona == 1:
```

```
        codigo_estudiante = int(input("Ingrese el código del Estudiante: "))
        for estudiante in estudiantes:
            if estudiante.codigo == codigo_estudiante:
                print(estudiante.consultar_info_personal())
                break
        else:
            print("Estudiante no encontrado.")
    elif tipo_persona == 2:
        id_docente = int(input("Ingrese el número de ID del Docente: "))
        for docente in docentes:
            if docente.nro_id == id_docente:
                print(docente.consultar_info_personal())
                break
        else:
            print("Docente no encontrado.")
    elif tipo_persona == 3:
        id_administrativo = int(input("Ingrese el número de ID del Administrativo: "))
        for administrativo in administrativos:
            if administrativo.nro_id == id_administrativo:
                print(administrativo.consultar_info_personal())
                break
        else:
            print("Administrativo no encontrado.")
    else:
```

```
            print("Tipo de persona no válido.")


    elif menu == 5:

        break


    else:

        print("Opción no válida. Por favor, selecciona una opción válida.")
```

FLUJO DE DATOS

Inicio del Programa:

Ambos programas se inician.

Definición de la Clase Base Vehiculo en el Primer Programa:

Se define la clase base **Vehiculo** con atributos comunes: **marca**, **modelo**, y **año**.

Se define el método mostrarInformacion() en la clase base para mostrar información básica del vehículo.

Definición de las Clases Derivadas Automovil y Motocicleta en el Primer Programa:

Se definen las clases derivadas Automovil y Motocicleta.

Cada clase derivada tiene atributos adicionales específicos, como numeroPuertas y cilindrada, respectivamente.

Se sobrescribe el método mostrarInformacion() en cada clase derivada para mostrar información específica de cada tipo de vehículo.

Creación de Objetos en el Primer Programa:

Se crean objetos de las clases derivadas, como automovil1 y motocicleta1.

Se establecen los atributos de estos objetos.

Llamada al Método mostrarInformacion() en el Primer Programa:

Se llama al método mostrarInformacion() en los objetos creados en el primer programa, como automovil1.mostrarInformacion() y motocicleta1.mostrarInformacion().

El método mostrarInformacion() muestra la información específica de cada vehículo, así como la información común heredada de la clase base.

Definición de las Clases Base y Derivadas en el Segundo Programa:

En el segundo programa, se define la clase base Persona con atributos generales: nombre, apellido y edad.

Se definen las clases derivadas Estudiante, Docente y Administrativo con atributos específicos para su rol en la universidad.

Se sobrescribe el método consultarInfoPersonal() en cada clase derivada para mostrar información relevante para su función.

Creación de Objetos en el Segundo Programa:

Se crean objetos de las clases derivadas en el segundo programa, como estudiante1, docente1 y admin1.

Se establecen los atributos de estos objetos.

Llamada al Método consultarInfoPersonal() en el Segundo Programa:

Se llama al método consultarInfoPersonal() en los objetos creados en el segundo programa, como estudiante1.consultarInfoPersonal(), docente1.consultarInfoPersonal() y admin1.consultarInfoPersonal().

El método consultarInfoPersonal() muestra información específica de cada rol en la universidad, además de la información común heredada de la clase base.

CONCLUSIONES

En estos dos programas, hemos explorado cómo los conceptos de herencia y polimorfismo en la programación orientada a objetos permiten crear jerarquías de clases eficientes y flexibles para modelar objetos y roles en diferentes contextos.

En ambos programas, hemos utilizado la herencia para definir una clase base que contiene atributos y métodos comunes, lo que simplifica el diseño y la reutilización de código

La sobreescritura de métodos en las clases derivadas nos permitió personalizar el comportamiento de los métodos heredados, lo que demuestra el polimorfismo. El mismo método puede tener diferentes comportamientos según la clase derivada en la que se llame.

ANEXOS

```
class Ingrediente:

def _init_(self, nombre, cantidad, unidad):

self.nombre = nombre

self.cantidad = cantidad

self.unidad = unidad
```

class PasoPreparacion:

def _init_(self, descripcion):

self.descripcion = descripcion

class Receta:

def _init_(self, nombre, categoria):

self.nombre = nombre

self.categoria = categoria

self.ingredientes = []

self.pasos_preparacion = []

def agregar_ingrediente(self, nombre, cantidad, unidad):

ingrediente = Ingrediente(nombre, cantidad, unidad)

self.ingredientes.append(ingrediente)

def agregar_paso_preparacion(self, descripcion):

paso = PasoPreparacion(descripcion)

self.pasos_preparacion.append(paso)

def mostrar_receta(self):

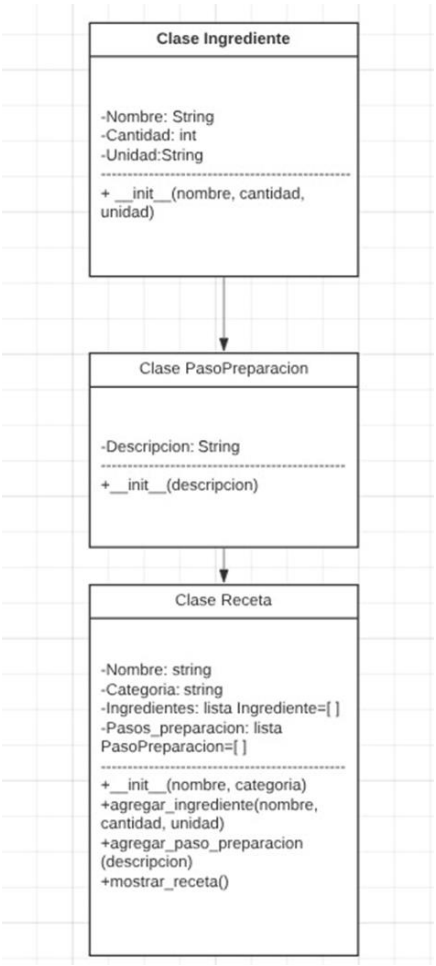
print(f"Receta: {self.nombre}")

print(f"Categoría: {self.categoria}")

print("\nIngredientes:")

for ingrediente in self.ingredientes:

```
print(f"- {ingrediente.cantidad} {ingrediente.unidad} de {ingrediente.nombre}")

print("\nPasos de Preparación:")

for i, paso in enumerate(self.pasos_preparacion, start=1):

print(f"{i}. {paso.descripcion}")



mi_receta = Receta("Tarta de Manzana", "Postre")



mi_receta.agregar_ingrediente("Manzanas", 4, "unidades")

mi_receta.agregar_ingrediente("Azúcar", 200, "gramos")

mi_receta.agregar_ingrediente("Harina", 150, "gramos")

mi_receta.agregar_ingrediente("Canela", 1, "cucharadita")



mi_receta.agregar_paso_preparacion("Pelar y cortar las manzanas en rodajas.")

mi_receta.agregar_paso_preparacion("Mezclar las manzanas con el azúcar y la canela.")

mi_receta.agregar_paso_preparacion("Forrar un molde con la masa de harina y verter la mezcla de manzanas.")

mi_receta.agregar_paso_preparacion("Hornear a 180°C durante 40 minutos.")



mi_receta.mostrar_receta()
```


Clase Ingrediente
-Nombre: String
-Cantidad: int
-Unidad:String
+ __init__(nombre, cantidad, unidad)
Clase PasoPreparacion
-Descripcion: String
+__init__(descripcion)
Clase Receta
-Nombre: string
-Categoria: string
-Ingredientes: lista Ingrediente=[ ]
-Pasos_preparacion: lista PasoPreparacion=[ ]
+__init__(nombre, categoria)
+agregar_ingrediente(nombre, cantidad, unidad)
+agregar_paso_preparacion (descripcion)
+mostrar_receta()