	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

GUÍA DE LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

Unidad didáctica 1: Diseño Orientado a Objetos

Eje Temático: Herencia y Polimorfismo

No. Guía	3			Resultados de Aprendizaje de la Unidad Didáctica	
				Escribe correctamente clases que utilicen herencia, implementando constructores en la clase base y en las clases heredadas.	
	2 sesiones			5 y 6 Semana	
	Horas de Trabajo			Realiza consultas bibliográficas de diferentes fuentes	
	Trabajo con Docente			Trabajo Autónomo	
6			12		
Tipo de trabajo			Comprende los fundamentos de Programación Orientada a Objetos.		
			Consulta sobre un lenguaje orientado a objetos diferente a los tradicionales.		
			Elabora un ejemplo de uso del lenguaje en cuanto a clases y objeto		
Laboratorio Requerido		Asistido por computador / Laboratorio de Informática			

Introducción

Sesión 1:

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
```

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

```

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 = []

```

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

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: ")

```

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

```

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: "))

```

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

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:

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

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

Subtemas:

Abstracción y Polimorfismo.
Definición y uso de la herencia.
Atributos y métodos protegidos.
Sintaxis para la herencia.
Aplicaciones de la herencia.
Constructores y herencia.

Preguntas Orientadoras

En este apartado se realiza el análisis de los datos obtenidos, estos pueden ser de forma cualitativa o cuantitativa según la naturaleza de la práctica.

¿Cuáles fueron los aprendizajes obtenidos al realizar esta guía?, liste como mínimo 3 aprendizajes y relaciónelos con su futuro que hacer profesional.

Herencia y Reutilización de Código: A través de la comprensión de la herencia y la sobreescritura de métodos, he aprendido cómo diseñar jerarquías de clases eficientes y reutilizables. Este conocimiento es esencial en mi futuro profesional, ya que me permitirá construir sistemas de software escalables y de fácil mantenimiento. Al reutilizar el código a través de la herencia, podré acelerar el desarrollo de aplicaciones y reducir errores.

Polimorfismo y Flexibilidad: El entendimiento del polimorfismo me ha mostrado cómo un mismo método puede comportarse de manera diferente según el contexto. Esto es fundamental para adaptar aplicaciones a requisitos cambiantes. En mi futuro profesional, seré capaz de diseñar sistemas que sean flexibles y capaces de acomodar nuevas funcionalidades sin tener que modificar el código existente, lo que ahorra tiempo y recursos.

Diseño Orientado a Objetos: Al trabajar con herencia y polimorfismo, he fortalecido mi comprensión del diseño orientado a objetos. Este enfoque de diseño es ampliamente utilizado en la industria del desarrollo de software. Con estos conocimientos, podré crear aplicaciones más estructuradas y modularizadas en mi futuro profesional, lo que facilitará la colaboración en equipos de desarrollo y mejorará la mantenibilidad de los sistemas.

En resumen, los conceptos de herencia, polimorfismo y diseño orientado a objetos son esenciales para mi futuro profesional en el campo de la programación y la ingeniería de software. Estos aprendizajes me permitirán ser un desarrollador más competente y versátil, capaz de abordar desafíos complejos y diseñar soluciones de software robustas y escalables.

¿Dónde presento mayor dificultad resolviendo la guía? y ¿cómo lo resolvieron? ¿cuáles fueron las estrategias de solución?


Haciendo el código, ya que fue lo que nos tomo mas tiempo dentro de la guía pero pues se resolvió consultando con mas compañeros dentro la clase

Presaberes Requeridos

Se requiere conocer los fundamentos de Programación e Ingeniería de Software, modelos y definiciones básicas relacionadas con el análisis, diseño, desarrollo e implementación de software. Identifica las clases, atributos y métodos envueltos en una situación práctica con el fin de plasmarlos en un programa.

Actividad de Comprobación del Trabajo Autónomo

Los estudiosos deben entregar un programa Java que cumpla con las especificaciones anteriores, en proyecto completo y comprimido en archivo independiente al trabajo de la introducción

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

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)

	Nombre del Proceso:	CODIGO: LA-FM-001
	GESTIÓN DE LABORATORIOS	
	Nombre del Documento:	VERSION: 7
	FORMATO PRACTICAS DE LABORATORIOS	FECHA: 15/junio/2022

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()

Palabras Clave

Control de cambios

Fecha de Actualización	Descripción	Participantes
07/07/2023	Actualización formato guía de laboratorio	Olga Lucía Roa
01/09/2023	Actualización ejercicios propuestos	Diana Marcela Toquica