INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

INGENIERÍA MECATRÓNICA

EXCELENTE
INVESTIGACIÓN

30 %

3º SEMESTRE

MATERIA: PROGRAMACION AVANZADA

INVESTIGACIÓN: "PARADIGMA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS Y PROGRAMACIÓN VISUAL"

DOCENTE: ING. ROBERTO ESTEBAN GUERRERO HERNANDEZ

ALUMNO: MARTÍNEZ SOLIS ALESSANDRO
NUMERO DE CONTROL: 23IU0383

FECHA: 16 DE SEPTIEMBRE DE 2024

INTRODUCCIÓN

La programación ha evolucionado significativamente para abordar la creciente complejidad de los sistemas modernos, dando lugar a diferentes paradigmas que organizan y estructuran el código de manera eficiente. Entre estos, la Programación Orientada a Objetos (POO) se ha destacado como uno de los enfoques más utilizados en el desarrollo de software, permitiendo representar entidades del mundo real mediante objetos que interactúan entre sí. A lo largo de esta investigación, se explorarán los conceptos clave de la POO, como clases, objetos, herencia, polimorfismo, abstracción y encapsulamiento, analizando cómo estos principios facilitan la creación de aplicaciones modulares, escalables y mantenibles.

Asimismo, se profundizará en la Programación Visual, un enfoque que permite diseñar interfaces gráficas de usuario de manera intuitiva y eficiente, utilizando herramientas que facilitan el desarrollo visual y la interacción mediante eventos. Esta investigación busca ofrecer una visión clara de cómo ambos paradigmas se complementan en la creación de aplicaciones interactivas, proporcionando a los desarrolladores las bases para construir software robusto y accesible, tanto en proyectos de escritorio como en aplicaciones web y móviles.

La programación, es una herramienta clave que nos permite comunicarnos con las máquinas y resolver problemas complejos a través de la tecnología. Sin embargo, a medida que los sistemas se han vuelto más sofisticados, la forma en que programamos ha evolucionado considerablemente. Así es como la Programación Orientada a Objetos (POO) ha surgido como uno de los paradigmas más influyentes y utilizados en el desarrollo de software moderno, brindando una manera más natural y estructurada de organizar el código.

Para quienes se adentran por primera vez en este paradigma, puede resultar complicado asimilar los conceptos detrás de la POO y su relación con la programación visual, que permite la creación de interfaces de usuario amigables e interactivas. A menudo, la dificultad radica en cambiar el enfoque de una programación secuencial, lineal y basada en instrucciones simples, hacia un modelo más abstracto, donde las entidades del mundo real se representan mediante objetos con atributos y comportamientos propios.

1. PARADIGMA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

¿Qué es un paradigma de programación?

Un paradigma de programación es un estilo o enfoque que dicta cómo se estructura y organiza el código en un lenguaje de programación. Existen varios paradigmas, como la programación estructurada, la funcional, y la orientada a objetos. Cada uno ofrece una manera distinta de pensar y organizar el código.

¿Por qué surge la POO?

A medida que los proyectos de software se volvieron más complejos, surgió la necesidad de un enfoque que pudiera manejar esa complejidad. El paradigma secuencial o estructurado, en el que las instrucciones se ejecutan de arriba hacia abajo, no era suficiente para organizar programas grandes y manejarlos de manera eficiente. Aquí es donde entra en juego la POO, cuyo objetivo es hacer que los programas sean más fáciles de entender, mantener y expandir.

Conceptos Clave de la POO:

La Programación Orientada a Objetos propone un modelo donde el código se organiza en torno a objetos, que son entidades que representan conceptos del mundo real o de un problema específico. Estos objetos tienen atributos (datos) y métodos (funcionalidad).

1. Clases y Objetos:

- Clase: Es una plantilla o un plano que define cómo será un objeto. Una clase describe qué atributos (datos) y métodos (acciones) tendrá un objeto.

- Objeto: Es una instancia de una clase. Es decir, un objeto es un ejemplar concreto creado a partir de una clase. Piensa en la clase como un molde, y en el objeto como el resultado que se obtiene al utilizar ese molde.

Un ejemplo para comprender lo anterior es el siguiente: En una tienda en línea, los productos, usuarios, y carritos de compra son objetos. Cada producto tiene atributos como el precio, la marca y la talla, y métodos como "ser comprado" o "ser añadido al carrito".

2. Atributos y Métodos:

- Atributos: Son las propiedades o características de un objeto. Por ejemplo, en la clase Producto, los atributos pueden ser el nombre, el precio, y el tamaño.
- Métodos: Son las acciones que un objeto puede realizar. En la clase Usuario, los métodos podrían incluir "iniciar sesión", "comprar un producto", etc.

Los atributos son los datos y los métodos son la funcionalidad de un objeto.

3. Abstracción: La abstracción es el proceso de identificar los atributos y comportamientos esenciales de un objeto y ocultar los detalles innecesarios. Cuando creamos una clase, no necesitamos detallar todo sobre el objeto, solo aquello que es relevante para el programa.
Cuando se abstrae el concepto de un usuario, se eligen atributos importantes como el nombre y la contraseña, y métodos como iniciar sesión o comprar. No es necesario incluir detalles sobre cómo el usuario está almacenado en la base de datos o cómo funcionan internamente los sistemas de autenticación.

4. Encapsulamiento: El encapsulamiento es la práctica de ocultar los detalles internos de un objeto y exponer solo lo necesario para que otros objetos puedan interactuar con él. Esto asegura que los datos dentro de un objeto estén protegidos de accesos o modificaciones no autorizadas.
Ejemplo: Los atributos de un objeto pueden ser privados, pero accesibles mediante métodos públicos llamados getters y setters. Así, otros objetos pueden obtener o modificar ciertos atributos, pero de una manera controlada.

5. Herencia: La herencia permite crear nuevas clases basadas en clases existentes, reutilizando código. Por ejemplo, una clase Vehículo puede tener subclases como Coche o Motocicleta, heredando atributos como el color o la velocidad y métodos como arrancar o detenerse. Sin embargo, también pueden añadir nuevos atributos o métodos específicos.
Esto se puede explicar comparando a los hijos que heredan características de sus padres, pero también desarrollan sus propias cualidades. Así, una clase derivada hereda los atributos y métodos de su clase base, pero puede añadir o modificar su comportamiento.

LISTA DE COTEJO PARA REPORTE DE PRÁCTICA

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA	**ASIGNATURA:** PROGRAMACIÒN AVANZADA
NOMBRE DEL DOCENTE: MTI. ROBERTO E. GUERRERO HERNANDEZ	**FIRMA DEL DOCENTE**

DATOS GENERALES DEL PROCESO DE EVALUACIÓN

NOMBRE(S) DEL ALUMNO(S): MARTINEZ SOLIS ALESSANDRO	**MATRICULA:** 231U0383	**FIRMA DEL ALUMNO(S):**

PRODUCTO: REPORTE DE PRÁCTICA	**NOMBRE DE LA PRACTICA:** PROGRAMACION ORIENTADA A OBJETOS	**FECHA:** SEPTIEMBRE - 2024	**PERIODO ESCOLAR:** AGOSTO – DICIEMBRE 2024

INSTRUCCIONES

Revisar las actividades que se solicitan y marque con una “X” en los apartados “SI” cuando la evidencia se cumple; en caso contrario marque “NO”. En la columna “OBSERVACIONES” indicaciones que puedan ayudar al alumno a saber cuáles son las condiciones no cumplidas, si fuese necesario.

VALOR DEL REACTIVO	CARACTERÍSTICA PARA CUMPLIR (REACTIVO)	CUMPLE		OBSERVACIONES
		SI	NO	
4%	Presentación El trabajo cumple con los requisitos de: **a**. Buena presentación	X		
4%	**b**. No tiene faltas de ortografía	X		
4%	**c**. Mismo Formato (letra arial 12, títulos con negritas)	X		
4%	**d**. Maneja el lenguaje técnico apropiado	X		
5%	**Desarrollo**: Sigue una metodología y sustenta todos los pasos que se realizaron en el análisis de la conversión de datos a binarios al aplicar los conocimientos obtenidos, es analítico y bien ordenado.	X		
4%	**Responsabilidad**: Entregó la práctica en la fecha y hora señalada.	X		
30%	**CALIFICACIÓN**	25 %		

INSITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA
ING. MECATRONICA
PROGRAMACION AVANZADA
PRACTICA

GRUPO: _311-A_ ALUMNO: _ Alessandro Martinez Solis_____

Muestra código en Python para el cálculo del área de un rectángulo en programación estructurada.

Buen ejemplo

25%

```
def calcular_area(ancho, alto):
    return ancho * alto

def main():
    # Solicitar al usuario las dimensiones del rectángulo
    ancho = float(input("Introduce el ancho del rectángulo: "))
    alto = float(input("Introduce el alto del rectángulo: "))

    # Calcular el área
    area = calcular_area(ancho, alto)

    # Mostrar el resultado
    print(f"El área del rectángulo es: {area:.2f}")

# Ejecutar la función principal
if __name__ == "__main__":
    main()
```

INSITUTO TECNOLOGICO SUPERIOR DE SAN ANDRES TUXTLA
ING. MECATRONICA
PROGRAMACION AVANZADA
EXAMEN UNIDAD I

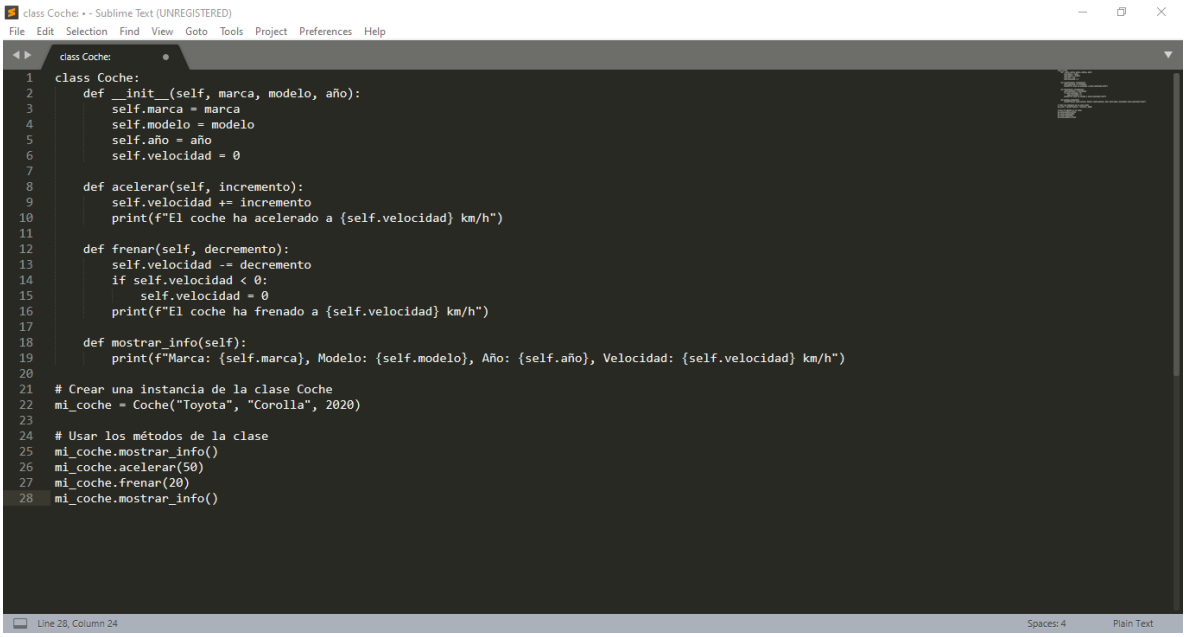
GRUPO: _311-A_ ALUMNO: _ Alessandro Martinez Solis_____

1.- Realiza código en Python en donde muestres código de Programación Orientada a Objetos de un coche.

FELICIDADES

40%

```
class Coche:
    def __init__(self, marca, modelo, año):
        self.marca = marca
        self.modelo = modelo
        self.año = año
        self.velocidad = 0

    def acelerar(self, incremento):
        self.velocidad += incremento
        print(f"El coche ha acelerado a {self.velocidad} km/h")

    def frenar(self, decremento):
        self.velocidad -= decremento
        if self.velocidad < 0:
            self.velocidad = 0
        print(f"El coche ha frenado a {self.velocidad} km/h")

    def mostrar_info(self):
        print(f"Marca: {self.marca}, Modelo: {self.modelo}, Año: {self.año}, Velocidad:
{self.velocidad} km/h")

# Crear una instancia de la clase Coche
mi_coche = Coche("Toyota", "Corolla", 2024)

# Usar los métodos de la clase
mi_coche.mostrar_info()
mi_coche.acelerar(50)
mi_coche.frenar(20)
mi_coche.mostrar_info()
```


class Coche: • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
class Coche:
class Coche:
def __init__(self, marca, modelo, año):
self.marca = marca
self.modelo = modelo
self.año = año
self.velocidad = 0
def acelerar(self, incremento):
self.velocidad += incremento
print(f"El coche ha acelerado a {self.velocidad} km/h")
def frenar(self, decremento):
self.velocidad -= decremento
if self.velocidad < 0:
self.velocidad = 0
print(f"El coche ha frenado a {self.velocidad} km/h")
def mostrar_info(self):
print(f"Marca: {self.marca}, Modelo: {self.modelo}, Año: {self.año}, Velocidad: {self.velocidad} km/h")
# Crear una instancia de la clase Coche
mi_coche = Coche("Toyota", "Corolla", 2020)
# Usar los métodos de la clase
mi_coche.mostrar_info()
mi_coche.acelerar(50)
mi_coche.frenar(20)
mi_coche.mostrar_info()
Line 28, Column 24
Spaces: 4
Plain Text