ENCAPSULAMIENTO

En P.O.O. el elemento encapsulamiento es un tema en prot que consiste ocultar los detalles internos del funcionamiento de un objeto y restringir el acceso directo a algunos de sus componentes, exponiendo solo lo necesario atreves de interfaces públicas (métodos).

Definición: encapsulamiento es el mecanismo que permite agrupar los datos(atributos)y los métodos que operan sobre esos datos en una sola unidad (clase), y controlar el acceso a ellos para proteger el estado interno del objeto.

Características:

- **Ocultamiento de datos:** los atributos de una clase suelen declararse como privados o protegidos, lo que impide que se acceda directamente a ellos desde fuera de la clase.
- **Control mediante métodos:** Se utilizan métodos públicos (gether y setters) para leer o modificar los atributos, aplicando validaciones si es necesario.
- **Mayor seguridad y mantenimiento:** Al proteger los datos internos, se evita que otras partes del programa los modifique de forma incorrecta o inesperada.

Ventajas:

- ✓ Protege la integridad de los datos.
- ✓ Permite cambiar la implementación interna sin afectar el código externo.
- ✓ Favorece el diseño modular y la reutilización del código.

¿Qué es getter?

Un getter es un método que permite acceder al valor de un atributo privado o protegido de una clase.

- Su función es obtener el valor.
- Se une para leer un atributo sin modificarlo directamente.

¿Qué es setter?

Un setter es un método que permite modificar el valor de un atributo privado o protegido de una clase.

- Su función es establecer asignar un nuevo valor.
- Normalmente incluye validaciones para asegurar que el valor sea correcto.

Ejemplo N°1:

Clase:calculadoraSuma

Atributos:

Acción:

Objeto:

```
class CalculadoraSuma:
   def __init__(self):
      self.total = 0

   def sumarNumeros(self):
```

```
        print("Calcula la suma de números ingresados")
        print("Escribe números para sumar. Escribe 'fin' para terminar")
        entrada = ""
        while entrada.lower() != "fin":
            entrada = input("Ingrese un número: ")
            if entrada.isdigit():
                self.total += int(entrada)
            elif entrada.lower() != "fin":
                print("Entrada inválida: Escriba un número o 'fin'")
        print(f"La suma total es: {self.total}")

def main():
    calculadora = CalculadoraSuma()
    calculadora.sumarNumeros()

if __name__ == "__main__":
    main()
```

Ejemplo N°2:

Clase: fibonacci

Atributos: cantidad

Acción:

Objeto:

```
class Fibonacci:
    def __init__(self, cantidad):
        self.cantidad = cantidad
        self.a = 0
        self.b = 1
        self.contador = 0
    def generarSerie(self):
        print("Serie de Fibonacci")
        while self.contador < self.cantidad:
            print(self.a, end=" ")
            c = self.a + self.b
            self.a = self.b
            self.b = c
            self.contador+=1


def main():
    miFibonacci = Fibonacci(10)
    miFibonacci.generarSerie()
if __name__=="__main__":
    main()
```

Ejemplo N°3:

Clase: numeros

Atributos:cantidad

Acción:

Objeto:

```
class Numeros:
   def __init__(self, cantidad):
      self.cantidad = cantidad
      self.contador = 0

   def generarNumero(self):
      print("Imprime Numeros")
      while self.contador <= self.cantidad:
         print(self.contador)
         self.contador+=1

def main():
   miNumero = Numeros(10)
   miNumero.generarNumeros()
if __name__=="__main__":
   main()
```

Ejemplo N°4:

Clase:calculadoraSuma

Atributos:

Acción:

Objeto:

```
class CalculadoraSuma:
   def __init__(self):
      self.total = 0

   def sumarNumeros(self):
      print("Calcula la suma de números ingresados")
      print("Escribe números para sumar. Escribe 'fin' para terminar")
      entrada = ""
      while entrada.lower() != "fin":
         entrada = input("Ingrese un número: ")
         if entrada.isdigit():
            numero = int(entrada)
            self.total += numero

            if numero  == 0:
               print(f"{numero} es nulo")
            if numero % 2 == 0:
               print(f"{numero} es par")
            else:
               print(f"{numero} es impar")
```

```
        elif entrada.lower() != "fin":
            print("Entrada inválida: Escriba un número o 'fin'")
        print(f"La suma total es: {self.total}")

def main():
    calculadora = CalculadoraSuma()
    calculadora.sumarNumeros()

if __name__ == "__main__":
    main()
```

Ejemplo N°5:

Clase: Persona

Atributos: nombre,edad

Acción:

Objeto:

```
class Persona:
    def __init__(self,nombre,edad):
        self.__nombre = nombre #atributo privado
        self.__edad = edad #atributo privado

    def get_edad(self):
        return self.__edad

    def get_nombre(self):
        return self.__nombre

    def set_edad(self,nueva_edad):
        if nueva_edad > 0:
            self.__edad = nueva_edad
        else:
            print("Edad no valida")

    def set_nombre(self,nuevo_nombre):
        self.__nombre = nuevo_nombre
        return self.__nombre

persona = Persona("Ana",30)
print(persona.get_nombre())
print(persona.get_edad())
print(persona.get_nombre(),persona.get_edad())

persona.set_edad(35)
persona.set_nombre("Maria")

print(persona.get_edad())
print(persona.get_nombre())
```

Ejemplo N°6:

Clase: circulo

Atributos: radio

Acción:

Objeto:

```
import math
class Circulo:
    def __init__(self,radio):
        self.__radio = radio #atributo privado

    def get_radio(self):
        return self.__radio

    def set_radio(self,nueva_radio):
        if nueva_radio > 0:
            self.__radio = nueva_radio
        else:
            print("Radio no valida")

    def calcular_area(self):
        return math.pi*self.__radio**2

    def calcular_perimetro(self):
        return 2*math.pi*self.__radio

circulo = Circulo(3)
print("Area del Circulo",round (circulo.calcular_area(),2))
print("Perimetro del Circulo",round(circulo. calcular_perimetro(),2))
```