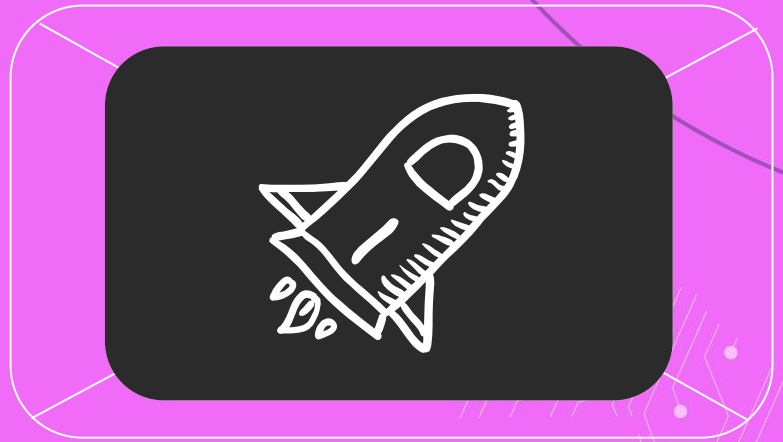
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

Semana 6
¡Bienvenidos!



Temas de hoy

1. POO: Repaso general
2. Ejercitación

1

POO - Repaso de lo que dimos

Desde ver la programación imperativa y pasar a la programación estructurada, luego ir por la programación funcional, llegamos a la programación orientada a objetos.

Todos estos paradigmas de programación te dan un conocimiento amplio, un mayor abanico de opciones y hacen que tu lógica como programador desate tu creatividad y te permita abordar los desafíos de desarrollo desde múltiples perspectivas. 🗄️ 👉

Programación Orientada a Objetos

¿Para qué nos sirve la POO?

Vamos a Imaginar que estás creando un videojuego. Con la POO, vamos a definir *clases* como *Personaje*, *Enemigo* y *Arma*. Cada *clase* tendrá sus propias características y acciones. Esto hace que el código sea más organizado y fácil de entender, incluso para otros programadores.

Entonces podemos decir que La Programación Orientada a Objetos (POO) es como una inversión a largo plazo, ya que al adoptar este paradigma, estaremos construyendo software más robusto, escalable y mantenible.

Nos facilitará la transición a lenguajes más modernos porque es un concepto fundamental en muchos lenguajes de programación actuales, por lo que aprender este paradigma de programación te abrirá muchas puertas en el futuro.

Al organizar el código en objetos con sus propias propiedades y comportamientos, conseguimos software más modular, fácil de mantener y reutilizable. Esto se traduce en una mayor productividad, menor número de errores y una mejor adaptación a los cambios.

En este paradigma tenemos 4 pilares:

- **Abstracción**: Definimos los atributos y métodos de una clase.
- **Herencia**: Las hijas heredan de las clases padre.
- **Polimorfismo**: Damos la misma orden a varios objetos para que respondan de manera diferente.
- **Encapsulamiento**: Protegemos la información de manipulaciones no autorizadas

Programación Orientada a Objetos: Ejercitación

Ejercicio 1

En clases anteriores vimos un ejemplo donde creamos un módulo personalizado para una biblioteca ¿Vamos a verlo desde la POO?

Para el ejemplo lo haremos un poco más grande y vamos a imaginar que vamos a modelar una librería.

Vamos a identificar las partes - Objetos y clases:

- **Libro**: Un *objeto* con *atributos* como **título**, **autor**, **ISBN**, **número de páginas**, **género** y *métodos* como **prestar**, **devolver**, **buscarPorTítulo**.
- **Autor**: Un *objeto* con *atributos* como **nombre**, **nacionalidad**, **fecha de nacimiento** y *métodos* como **publicarLibro**.
- **Lector**: Un objeto con *atributos* como **nombre**, **edad**, **dirección**, **número de socio** y *métodos* como **solicitarPréstamo**, **devolverLibro**.
- **Librería**: Un *objeto* que agrupa todos los **libros**, **autores** y **lectores**, y tiene *métodos* como **agregarLibro**, **buscarLibro**, **prestarLibro**, **devolverLibro**, **registrarLector**.

Ahora, vamos a ir a ver como entra la POO conceptualmente:

Abstracción: Nos enfocamos en las características esenciales de un libro (título, autor) y no en los detalles de su impresión.

Encapsulación: Los atributos de un libro (como el número de páginas) pueden ser privados, y solo se accede a ellos a través de los métodos (como prestar).

Programación Orientada a Objetos: Ejercitación

Herencia: Podríamos crear una subclase de *Libro* llamada ***LibroInfantil*** que herede los atributos y métodos de *Libro* y agregue otros específicos (*edad recomendada*, *ilustraciones*).

Polimorfismo: El método *prestar* podría tener un comportamiento diferente dependiendo del tipo de lector, por ejemplo, los socios pueden pedir prestados más libros.

- Este ejemplo va a cubrir:

→ **Creación de clases**: Libro, Lector, Librería, LibroElectronico, LibroAudiovisual

→ **Atributos y métodos**: Cada clase tiene sus propias propiedades y comportamientos.

→ **Relaciones entre objetos**: Un libro pertenece a una librería y puede ser prestado a un lector (Este punto, de relaciones, lo vas a comprender mejor la semana que viene cuando empecemos con Base de Datos).

→ **Encapsulación**: El estado de un libro (prestado o no) es privado.

→ **Polimorfismo**: Podríamos agregar más tipos de lectores (niños, adultos mayores) con comportamientos diferentes al prestar libros.

El código de este ejercicio se abordará en clase y en mentoría, por lo que tu profesor o mentor estará mostrándolo.

Programación Orientada a Objetos: Ejercitación

Ejercicio 2

Crearemos una *clase Persona* en la cual guardaremos la información general de cada persona, y una *clase Estudiante* que heredará los atributos y métodos de la clase anterior con información más precisa sobre la carrera que está estudiando.

→ En la *clase Persona()* indicaremos los atributos *nombre* y *apellido*.
Además, definimos un método el cual nos muestre ambos atributos en un mismo mensaje (nombre_completo()).

→ En la *clase Estudiante()* heredamos de *Persona()* sus atributos y métodos, por eso mismo, en el constructor requerimos los datos de nombre y apellido nuevamente, lo que nos permite pasar dichos valores a su clase padre mediante *super()* pero agregando el nuevo atributo *carrera*.
Por último, definimos un método que nos devuelva que carrera estudia actualmente el estudiante.

```python
class Persona():
    def __init__(self, nombre, apellido):
        self.nombre = nombre
        self.apellido = apellido

    def nombre_completo(self):
        print(f"El nombre completo es {self.nombre} {self.apellido}")

class Estudiante(Persona):
    def __init__(self, nombre, apellido, carrera):
        super().__init__(nombre, apellido)
        self.carrera = carrera

    def mostrar_carrera(self):
        print(f"El estudiante está estudiando la carrera {self.carrera}")
```

Programación Orientada a Objetos: Ejercitación

Ejercicio 3

Similar al ejemplo anterior, definiremos las clases *Universidad()* y *Carrera()* para posteriormente ser heredadas en la clase *Estudiante()* donde reuniremos el nombre de la universidad y la carrera que nuestro estudiante objeto estudia, además de requerir sus datos personales.

```python
class Universidad():
    def __init__(self, nombreUniversidad):
        self.nombreUniversidad = nombreUniversidad
```

```python
class Carrera():
    def __init__(self, especialidad):
        self.especialidad = especialidad
```

```python
class Estudiante(Universidad, Carrera):
    def __init__(self, nombre, apellido, nombreUniversidad, especialidad):
        Universidad.__init__(self, nombreUniversidad)
        Carrera.__init__(self, especialidad)
        self.nombre = nombre
        self.apellido = apellido

    def mostrar_datos(self):
        print(f"""
                El nombre del estudiante es {self.nombre} y tiene {self.edad} años.
                Estudia la carrera de {self.especialidad} en la universidad {self.nombreUniversidad}
                """)
```

Programación Orientada a Objetos: Ejercitación

Ejercicio 4

Crearemos un sistema bancario que nos permita crear una cuenta, depositar, retirar y consultar la misma. Como son datos sensibles que deseamos proteger su integridad por fuera de esta clase, es prioritario utilizar encapsulamiento en este ejemplo.

```python
class CuentaBancaria:
    def __init__(self, titular, saldo):
        self.__titular = titular
        self.__saldo = saldo
        print(f"""
              Cuenta creada exitósamente.
              Titular: {self.__titular}
              Saldo: ARS${self.__saldo}
              """)

    def depositar(self, monto):
        if type(monto) == int or type(monto) == float:
            if(monto > 0):
                self.__saldo += monto
                print(f"Se realizó el depósito de ARS${monto} correctamente. Saldo actual: ARS${self.__saldo}")
            else:
                print("El monto ingresado no es válido. Debe depositar un monto positivo")
        else:
            print("Error. El monto a depositar debe ser numérico.")


    def retirar(self, monto):
        if type(monto) == int or type(monto) == float:
            if monto > 0:
                if monto <= self.__saldo:
                    self.__saldo -= monto
                    print(f"Se realizó el retiro correctamente. Saldo actual: ARS${self.__saldo}")
                else:
                    print("Fondos insuficientes.")
            else:
                print("El monto a retirar debe ser mayor a cero")
        else:
            print("Monto incorrecto. El valor que desea retirar debe ser un número")

    def consultar_saldo(self):
        print(f"Su saldo actual es de ARS${self.__saldo}")

    def consultar_titular(self):
        print(f"El propietario de la cuenta es {self.__titular}")
```

Lo que vimos hoy 😉

En esta clase de ejercitación abordamos algunos ejercicios básicos y uno más complejo (pero sin pasarnos de mucha complejidad 😉).

Cuando lleguemos al proyecto final, algunas de las cosas dadas en estos ejemplo, te vendran a la mente, por lo que de a poco, te vamos dando ciertas herramientas para los fines de esta cursada. 😉

¡Nos vemos En la próxima clase!