CARACTERÍSTICAS POO

1. Abstracción: La abstracción en la Programación Orientada a Objetos (POO) es un pilar esencial que se centra en la representación y manipulación de ideas abstractas en el código. Este enfoque nos permite concentrarnos en las características clave de un objeto o sistema sin tener que lidiar con los detalles específicos de su implementación.

En POO, la abstracción se materializa a través de la creación de clases y objetos. Una clase actúa como una plantilla que define las propiedades y comportamientos compartidos por un grupo de objetos, mientras que un objeto es una instancia particular de esa clase. Estos se componen por atributos y elementos que afectan a las clases como objetos.

La abstracción en POO se da por el encapsulamiento, que oculta la complejidad interna de los objetos y muestra solo lo esencial al usuario; también se emplea la generalización a través de clases generales y herencia para facilitar la reutilización del código y la estructuración modular del programa y sirve sirve para simplificar, organizar y reutilizar el código de manera eficiente y modular.

2. Encapsulamiento:La encapsulación, esencial en la programación orientada a objetos (POO), agrupa datos y métodos dentro de una clase para ocultar la complejidad interna del objeto y presentar una interfaz pública para su uso.

Los beneficios que trae el Encapsulamiento es ocultar los datos internos de un objeto, protegiéndolos de cambios accidentales; facilita la abstracción al simplificar la interfaz de uso del objeto; fomenta la modularidad al dividir el código en unidades manejables; controla el acceso a datos y métodos, mejorando la seguridad del programa.

3. Herencia: La herencia en programación orientada a objetos (POO) significa que una clase puede heredar características (como atributos y

métodos) de otra clase, lo que facilita la reutilización de código y la organización en jerarquías de clases.

Sus características son que la herencia en programación es importante para reutilizar código al permitir que las clases heredan atributos y métodos de otras, evitando duplicaciones y promoviendo eficiencia. Además, crea una estructura jerárquica entre clases, facilita la adición de nuevas funcionalidades sin alterar la base, y permite que objetos derivados se comporten de manera coherente pero adaptada a sus necesidades específicas.

4. Polimorfismo: El polimorfismo es una característica fundamental de la Programación Orientada a Objetos (POO) que permite a los objetos de diferentes clases responder de manera distinta a la misma solicitud o mensaje. Esto se logra mediante la capacidad de un objeto de tomar múltiples formas y comportarse de manera adecuada según el contexto en el que se encuentra.

El polimorfismo en la POO permite a los objetos cambiar de forma y comportarse de manera única según el mensaje que reciben o el contexto. Esto brinda flexibilidad al adaptarse dinámicamente a diversas situaciones, lo que facilita la reutilización del código y contribuye a la creación de sistemas más escalables y fáciles de mantener.

5. Clases y objetos: Engloba la encapsulación, abstracción, herencia y polimorfismo.
6. Métodos y atributos: En la programación orientada a objetos (POO), los métodos y atributos son esenciales para definir el comportamiento y las características de las clases y objetos. Los métodos representan las acciones que los objetos pueden realizar, como cálculos, interacciones o modificaciones de estado. Por otro lado, los atributos son variables que almacenan información sobre el estado de un objeto, como números, textos o referencias a otros objetos. Ambos elementos son fundamentales para modelar objetos de manera efectiva y crear programas modulares y reutilizables.

En la programación orientada a objetos (POO), los métodos son las funciones que definen las capacidades y acciones de un objeto. Estas acciones pueden abarcar desde realizar cálculos complejos hasta interactuar con otros objetos en el sistema. Además, los métodos pueden tener diferentes niveles de acceso: públicos (accesibles desde fuera de la clase), privados (solo accesibles desde la propia clase) o protegidos (accesibles desde la clase y sus subclases), lo que permite controlar quién puede manipular la información del objeto y cómo lo hacen. Esta capacidad de restringir el acceso se conoce como encapsulamiento y es crucial para la seguridad y la integridad de los datos en el sistema. Además, la reutilización de métodos en diferentes partes del programa o clases promueve la modularidad y facilita el mantenimiento del código.

Por otro lado, los atributos en la POO representan las características o propiedades de un objeto, como su estado actual, almacenando datos como números, textos u objetos relacionados. Estos atributos también pueden tener diferentes niveles de visibilidad (públicos, privados o protegidos), determinando quién puede acceder y modificar sus valores. La inicialización de atributos con valores predeterminados o la posibilidad de modificarlos durante la ejecución del programa ofrece flexibilidad y adaptabilidad a las necesidades específicas de la aplicación. En conjunto, los métodos y atributos son pilares fundamentales de la POO que permiten modelar objetos de manera efectiva y estructurar programas de forma modular y mantenible.

7. Modularidad: La modularidad en la programación orientada a objetos (POO) se refiere a la capacidad de dividir un programa en módulos independientes, cada uno enfocado en una tarea específica o componente del sistema.

La modularidad en programación ofrece ventajas clave: divide el código en unidades comprensibles y funcionales, facilita la reutilización del código, promueve la abstracción al ocultar la complejidad interna, garantiza la independencia entre módulos para modificarlos sin afectar otros, agiliza la colaboración entre desarrolladores y mejora la mantenibilidad al permitir cambios sin impacto en otras partes del código.

8. Reusabilidad: La reusabilidad en la programación orientada a objetos (POO) se refiere a la capacidad de utilizar código existente en diferentes partes de un programa o en programas diferentes, con el objetivo de reducir la duplicación de código y mejorar la eficiencia del desarrollo.

se caracteriza por usar:

- Herencia: Permite a una clase heredar atributos y métodos de otra, facilitando crear nuevas clases con funcionalidades adicionales.
- Composición: Un objeto puede contener instancias de otros objetos, permitiendo combinar funcionalidades y modularidad.
- Interfaces: Definen métodos que clases deben implementar, fomentando la reusabilidad al permitir intercambio entre clases.
- Librerías/Módulos: Acceso a funcionalidades desarrolladas por otros, optimizando la reutilización de código.

- Patrones de diseño: Soluciones generales a problemas comunes, ayudando a estructurar el código para mayor entendimiento y reutilización.