CARACTERISTICAS POO:

ABSTRACCION:

Encapsulamiento: Esconde los detalles internos de un objeto y solo expone las operaciones relevantes a través de una interfaz pública.

Clases y Objetos: Las clases representan plantillas para crear objetos, definiendo atributos y métodos.

Herencia: Permite la creación de clases derivadas (subclases) a partir de clases base (superclases), facilitando la reutilización de código.

Polimorfismo: Permite que un mismo método pueda comportarse de diferentes maneras según el tipo de objeto que lo invoque.

ENCAPSULAMIENTO:

Ocultamiento de información: El encapsulamiento oculta los detalles internos de un objeto, permitiendo acceder a sus datos y métodos solo a través de una interfaz definida.

Protección de datos: Los datos dentro de un objeto están protegidos y solo pueden ser modificados mediante métodos específicos, lo que previene cambios no deseados y asegura la consistencia de los datos.

Abstracción de datos y comportamiento: Permite definir una interfaz clara para interactuar con un objeto, sin necesidad de conocer su implementación interna, promoviendo así la modularidad y facilitando el mantenimiento del código.

HERENCIA:

Reutilización de código: La herencia permite a una clase heredar atributos y métodos de otra clase, lo que facilita la reutilización del código existente.

Jerarquía de clases: Las clases se organizan en una jerarquía, donde una clase derivada (o subclase) hereda de una clase base (o superclase). Esto crea una estructura jerárquica que refleja relaciones de especialización y generalización.

Extensibilidad: Las subclases pueden agregar nuevos métodos y atributos a los que heredan de su superclase, lo que permite extender la funcionalidad de la clase base según las necesidades del programa.

Polimorfismo: Las subclases pueden proporcionar su propia implementación de métodos heredados, lo que permite que diferentes objetos respondan de manera diferente a la misma llamada de

método. Esto facilita el diseño de programas flexibles y modulares.

Encapsulamiento: La herencia no afecta el encapsulamiento de los objetos. Las subclases heredan los miembros públicos y protegidos de su superclase, pero no pueden acceder a los miembros privados directamente.

POLIMORFISMO:

Métodos con el mismo nombre: Permite que diferentes clases tengan métodos con el mismo nombre pero con implementaciones diferentes.

Sobrescritura de métodos: Las subclases pueden proporcionar una implementación específica para un método que ya está definido en su clase base.

Interfaces y clases abstractas: Se puede lograr polimorfismo a través de interfaces y clases abstractas, donde múltiples clases pueden implementar la misma interfaz o heredar de la misma clase abstracta y proporcionar implementaciones específicas para los métodos definidos en ellas.

Ligadura dinámica (Dynamic binding): La selección del método que se va a ejecutar se realiza en tiempo de ejecución en función del tipo de objeto al que se está haciendo referencia, lo que permite que el código sea más flexible y dinámico.

Código reutilizable y mantenible: El polimorfismo promueve la reutilización del código, ya que las clases pueden compartir métodos comunes a través de interfaces o clases base, lo que a su vez facilita el mantenimiento y la extensión del código.

CLASES Y OBJETOS:

Es importante señalar que la POO puede variar según el programador. Y esto sucede porque hay un cambio de concepto; no se trata tanto de una única escala sino, de una forma de concebir la programación.

Lo cierto es que este tipo de programación es mucho más abierta, aunque favorece una estructuración ordenada. Se requiere de una cierta formación previa, pero en la práctica hay varias ventajas por las que puede interesar esta metodología. La organización del código se realiza en distintas clases que, posteriormente, podrán concretarse en objetos.

El módulo fue la primera introducción de programación para reaprovechamiento, pero aquí se va un paso más allá. La POO busca, en definitiva, que las aplicaciones que se desarrollen sean cada vez más complejas sin que eso suponga desechar el código. Esta filosofía permitirá reutilizarlo, de manera que progresar no supondrá renunciar.

METOODS Y ATRIBUTOS

ATRIBUTOS :

Encapsulación: Los atributos pueden ser públicos, privados o protegidos, lo que controla su accesibilidad desde fuera de la clase.

Abstracción: Representan características o datos de un objeto, abstrayendo detalles internos de su implementación.

Estado: Los atributos definen el estado de un objeto, es decir, sus características en un momento dado.

Visibilidad: Los modificadores de acceso determinan quién puede acceder y modificar los atributos de una clase.

METODOS:

Encapsulación: Los métodos encapsulan el comportamiento de un objeto, definiendo qué puede hacer el objeto.

Abstracción: Los métodos proporcionan una interfaz para interactuar con los objetos, ocultando la complejidad de su implementación.

Comportamiento: Los métodos definen las acciones que un objeto puede realizar o los cálculos que puede realizar sobre sus datos.

Polimorfismo: Los métodos pueden tener el mismo nombre en diferentes clases, pero comportarse de manera diferente, dependiendo del objeto al que se apliquen.

MODULARIDAD:

Reutilización de código: Los módulos o clases pueden ser reutilizados en diferentes partes del programa o incluso en otros programas, lo que ahorra tiempo y esfuerzo de desarrollo.

Facilidad de mantenimiento: Los cambios en un módulo o clase específica tienen un impacto limitado en el resto del sistema. Esto facilita la detección y corrección de errores, así como la incorporación de nuevas funcionalidades.

Claridad y legibilidad: Al dividir el sistema en módulos bien definidos, el código se vuelve más legible y comprensible. Cada módulo se encarga de una parte específica de la funcionalidad, lo que facilita la comprensión del sistema en su conjunto.

Facilidad de colaboración: La modularidad facilita el trabajo en equipo, ya que diferentes desarrolladores pueden trabajar en módulos diferentes de forma independiente. Esto promueve la colaboración y la división del trabajo en proyectos grandes.

Acoplamiento débil: Los módulos están diseñados para tener dependencias mínimas entre sí. Esto significa que un cambio en un módulo no necesariamente requiere cambios en otros módulos, lo que reduce el riesgo de efectos secundarios no deseados.

Cohesión alta: Los módulos están diseñados para realizar una tarea específica y hacerlo bien. Esto se conoce como cohesión alta, lo que significa que cada módulo tiene una funcionalidad clara y única.

REUSABILIDAD

Encapsulación: Permite ocultar los detalles internos de un objeto y solo exponer lo que es necesario para su uso externo. Se logra mediante el uso de modificadores de acceso como public, private y protected en los miembros de la clase.

Abstracción: Permite modelar entidades del mundo real como objetos con características y comportamientos específicos. La abstracción se logra definiendo una interfaz clara para los objetos, ocultando los detalles de implementación innecesarios.

Herencia: Permite que una clase (subclase) herede atributos y métodos de otra clase (superclase). Esto promueve la reutilización del código y establece relaciones jerárquicas entre las clases.

Polimorfismo: Permite que los objetos de distintas clases respondan al mismo mensaje de diferentes maneras. Puede manifestarse en forma de sobrecarga de métodos (métodos con el mismo nombre pero diferentes parámetros) y sobreescritura de métodos (métodos con el mismo nombre y firma en clases relacionadas).