GA4-220501095-AA2-EV01 - Taller de conceptos y principios de programación orientada a objetos

Integrantes:

Rodney Zapata Palacio

Presentado a la instructora:

Elizabeth Robayo Ramirez

Servicio Nacional de aprendizaje SENA

Centro de Comercio y Servicios (Regional Cauca)

Cauca - Popayán

Tecnólogo en Análisis y Desarrollo de Software

Ficha: 2675810

1. REVISIÓN HISTÓRICA

Fecha	Descripción	Autor	Versión
23/07/2023	**Taller de conceptos y principios de programación orientada a objetos**	Rodney Zapata Palacio	1.0

Tabla Contenido

Tabla de contenido

2. Glosario de terminología utilizada en la POO,

Realizar un glosario de terminología utilizada en la POO, cada término debe ser explicado con sus propias palabras. En su propio concepto explicar cuáles son las características y los principios o pilares básicos de la POO.

3. POO (Programación Orientada a Objetos).

La Programación Orientada a Objetos (POO) es un paradigma de programación que se centra en modelar el mundo real a través de objetos, que son entidades que contienen datos y comportamientos relacionados.

En este enfoque, un "objeto" puede ser cualquier cosa con atributos y acciones asociadas. Por ejemplo, si queremos representar un coche en un programa, podríamos crear un objeto "Coche" con atributos como color, modelo y año, y comportamientos como acelerar, frenar y girar.

La POO se basa en cuatro pilares fundamentales: Encapsulamiento, abstracción, Herencia y Polimorfismo.

La POO nos ayuda a organizar el código de manera modular, facilitar la resolución de problemas complejos y mejorar la mantenibilidad y extensibilidad del software.

4. Clases

En Programación Orientada a Objetos (POO), una "clase" es un plano, una plantilla o un molde que define la estructura y el comportamiento de los objetos que se crearán a partir de ella. Es decir, una clase es un conjunto de atributos y métodos que describen las características y acciones que tendrán los objetos que pertenecen a esa clase.

Supongamos que queremos representar el concepto de un coche en un programa. Podemos definir una clase llamada "Coche" que contenga los atributos como "marca", "modelo", "color" y "año", y también métodos como "acelerar", "frenar" y "girar".

Una vez que hemos definido la clase "Coche", podemos crear múltiples objetos a partir de ella, cada uno con sus propios valores específicos para los atributos. Por ejemplo, podemos crear dos objetos llamados "coche1" y "coche2", ambos pertenecientes a la clase "Coche", pero con diferentes valores para los atributos, como "coche1.marca = 'Toyota'", "coche1.modelo = 'Corolla'" y "coche2.marca = 'Honda'", "coche2.modelo = 'Civic'".

La clase actúa como una plantilla que define la estructura y el comportamiento compartido por todos los objetos que se crean a partir de ella. Los objetos, por otro lado, son las instancias concretas que representan entidades individuales y que pueden tener valores diferentes para los atributos, pero heredan el comportamiento definido por la clase.

5. Herencias:

La herencia permite crear una nueva clase (llamada "clase derivada" o "subclase") basada en una clase ya existente (llamada "clase base" o "superclase"), de modo que la nueva clase hereda todos los atributos y métodos de la clase base.

La herencia facilita la reutilización de código y promueve la organización jerárquica de las clases. Cuando una clase es heredada, la subclase automáticamente posee todas las características de la superclase, lo que significa que puede acceder y utilizar sus atributos y métodos directamente. Además, la subclase puede agregar nuevos atributos y métodos específicos o incluso modificar los comportamientos heredados de la superclase, adaptándolos a sus necesidades particulares.

Para ilustrar esto, consideremos el ejemplo del coche que mencionamos anteriormente. Si ya hemos definido una clase base llamada "Coche" con atributos como "marca", "modelo", "color" y métodos como "acelerar", "frenar" y "girar", podríamos crear una subclase llamada "CocheDeportivo" que herede de la clase "Coche".

La clase "CocheDeportivo" heredaría todos los atributos y métodos de la clase "Coche", lo que significa que tendría "marca", "modelo", "color" y los métodos de "acelerar", "frenar" y "girar". Sin embargo, también podríamos agregar atributos y métodos específicos para los coches deportivos, como "velocidadMaxima" y "activarModoDeportivo".

De esta manera, la herencia nos permite crear una jerarquía de clases, donde las clases base representan conceptos más generales y las subclases representan conceptos más específicos, y cada una puede extender y modificar el comportamiento de las clases que hereda.

6. Objetos:

En Programación Orientada a Objetos (POO), los "objetos" son las instancias concretas creadas a partir de una clase. Como mencioné anteriormente, una clase es una plantilla que define la estructura y el comportamiento común que tendrán los objetos pertenecientes a esa clase.

Tomando el ejemplo del coche que hemos utilizado previamente, si tenemos una clase llamada "Coche", podemos crear múltiples objetos a partir de esa clase, como "coche1", "coche2", "coche3", etc. Cada uno de estos objetos será una instancia independiente de la clase "Coche", con sus propios valores para los atributos y la capacidad de ejecutar los métodos definidos en la clase.

Por ejemplo, podríamos tener un objeto "coche1" con atributos como "marca = 'Toyota'", "modelo = 'Corolla'", "color = 'Rojo'", y la capacidad de acelerar, frenar y girar mediante los métodos proporcionados por la clase "Coche".

La POO se basa en el concepto de objetos para modelar el mundo real y organizar el código de una manera más modular y orientada a los problemas. Los objetos permiten encapsular datos y comportamientos relacionados, lo que facilita la manipulación y la interacción con ellos de manera más intuitiva y estructurada. Además, el uso de objetos favorece la reutilización de código, ya que las clases pueden ser instanciadas en diferentes partes del programa y heredadas para crear nuevas clases y jerarquías.

7. Métodos:

En Programación Orientada a Objetos (POO), los "métodos" son acciones o funciones específicas que pueden ser realizadas por los objetos creados a partir de una clase.

Los métodos operan sobre los datos (atributos) de un objeto y pueden modificar su estado interno o proporcionar funcionalidades relacionadas con el objeto en cuestión. Cada objeto creado a partir de una clase puede invocar y ejecutar los métodos definidos en esa clase.

Tomando el ejemplo del coche nuevamente, si tenemos una clase llamada "Coche", podríamos definir métodos para representar acciones que los objetos coche pueden realizar, como "acelerar", "frenar" y "girar". Estos métodos contendrían la lógica necesaria para llevar a cabo las acciones específicas que corresponden a cada comportamiento.

8. Eventos:

En Programación Orientada a Objetos (POO), los "eventos" son sucesos o acontecimientos que ocurren durante la ejecución de un programa y que pueden desencadenar la ejecución de acciones específicas. Los eventos están diseñados para permitir que los objetos de un programa respondan y reaccionen a situaciones o interacciones particulares que puedan ocurrir en el entorno en el que se encuentran.

Por ejemplo en una interfaz gráfica de usuario, cuando el usuario hace clic en un botón, se genera un evento que puede desencadenar una acción específica.

Cuando el usuario presiona una tecla en el teclado, se genera un evento de pulsación de tecla que puede ser capturado y utilizado para realizar acciones en respuesta.

Los eventos son esenciales para el desarrollo de aplicaciones interactivas, como aplicaciones con interfaz gráfica de usuario, videojuegos y aplicaciones web, ya que permiten que el programa reaccione a las acciones del usuario o a cambios en el entorno y realice acciones apropiadas en consecuencia. Además, los eventos ayudan a mantener el

código organizado y modular, ya que permiten separar las responsabilidades de manejo de eventos de otras partes del programa.

9. Atributos:

En Programación Orientada a Objetos (POO), los "atributos" son variables que representan las propiedades o características de un objeto. Cada objeto creado a partir de una clase puede tener sus propios valores específicos para estos atributos, lo que le da una identidad única.

Los atributos son también conocidos como "campos" o "propiedades". Son utilizados para representar datos asociados a un objeto y definir el estado interno del mismo. Por ejemplo, si tenemos una clase "Coche", algunos atributos típicos podrían ser "marca", "modelo", "color" y "año".

10. La abstracción:

En la POO, una clase actúa como una abstracción de un concepto o entidad del mundo real. Por ejemplo, si queremos representar el concepto de un coche en un programa, crearíamos una clase "Coche" que tendría atributos como "marca", "modelo", "color" y métodos como "acelerar", "frenar" y "girar". Esta clase encapsula los aspectos clave de un coche sin preocuparse por los detalles internos de cómo se implementan esos comportamientos.

La abstracción permite modelar entidades complejas de manera más simple y comprensible, lo que facilita el desarrollo de aplicaciones y el trabajo en equipo. Al abstraer los detalles complejos, los programadores pueden centrarse en cómo interactuar con los objetos sin preocuparse por su implementación interna.

Además, la abstracción promueve la reutilización de código, ya que una vez que se ha definido una clase abstracta que representa una entidad genérica, es posible crear múltiples objetos (instancias) de esa clase con diferentes valores para sus atributos específicos, pero todos compartiendo el mismo comportamiento general.

11. El encapsulamiento:

El encapsulamiento se refiere a la idea de ocultar los detalles internos de un objeto o clase, protegiendo sus atributos y comportamientos para que no puedan ser accedidos o modificados directamente desde fuera de la clase. En cambio, el acceso a estos datos y comportamientos se realiza a través de una interfaz controlada y definida por la clase, que se conoce como "interfaz pública".

El objetivo principal del encapsulamiento es lograr que los objetos mantengan su integridad interna y solo puedan ser manipulados de acuerdo con las reglas establecidas en la clase. Esto proporciona varias ventajas:

Protección de datos: Al ocultar los detalles internos del objeto, el encapsulamiento evita que los datos internos sean modificados o corrompidos de forma accidental o no autorizada.

Al encapsular el comportamiento y los datos, se facilita el mantenimiento del código, ya que los cambios pueden realizarse en un solo lugar (dentro de la clase) en lugar de propagarlos por todo el programa.

Para lograr el encapsulamiento, se utilizan modificadores de acceso en los lenguajes de programación orientados a objetos, como "public", "private" y "protected". Estos modificadores indican qué miembros (atributos y métodos) de la clase son accesibles desde fuera de la clase.

12. El polimorfismo:

El polimorfismo, viene de la palabra poli que significa varios, y formis que significa formas, osea varias formas Es decir, el polimorfismo permite que objetos de diferentes clases respondan a un mismo mensaje o llamada de manera específica para cada uno, proporcionando una interfaz común para interactuar con ellos.

Un ejemplo común de polimorfismo es el uso de un método con el mismo nombre en diferentes clases. Supongamos que tenemos una clase base llamada "Figura" con un método "calcularArea()". Luego, creamos dos clases derivadas, "Círculo" y "Rectángulo", que heredan de "Figura" y cada una de ellas reemplaza el método "calcularArea()" con su propia implementación adecuada.

Si tenemos un arreglo de objetos "Figura", podemos almacenar tanto objetos "Círculo" como "Rectángulo" en él. Cuando llamamos al método "calcularArea()" en cada uno de los objetos, se ejecutará la versión específica de ese método dependiendo del tipo de objeto en tiempo de ejecución.

El polimorfismo permite escribir un código más genérico y flexible, lo que facilita el desarrollo de aplicaciones y la reutilización de código. Además, mejora la mantenibilidad, ya que al agregar nuevas clases derivadas que cumplan con la interfaz común, no es necesario modificar el código existente que las utiliza.