TEMA 2. Encapsulación

1. En Programación Orientada a Objetos (POO), ¿Qué buscan la **encapsulación** y **la ocultación** de información? Enumera brevemente algunas ventajas de la ocultación de información.

Respuesta

La encapsulación y la ocultación de información son principios fundamentales en la Programación Orientada a Objetos (POO) que buscan proteger los datos y garantizar la integridad de los objetos. La encapsulación consiste en agrupar datos y métodos que operan sobre esos datos dentro de una misma unidad, llamada clase. Por otro lado, la ocultación de información se refiere a restringir el acceso directo a los datos internos de un objeto, permitiendo que solo se interactúe con ellos a través de métodos específicos.

Entre las ventajas de la ocultación de información se encuentran las siguientes:

1. **Protección de los datos**: Al restringir el acceso directo a los atributos de una clase, se evita que estos sean modificados de manera indebida o accidental, lo que ayuda a mantener la consistencia y la validez de los datos.
2. **Facilidad de mantenimiento**: Al encapsular los datos y exponer solo lo necesario mediante métodos públicos, se reduce el riesgo de que cambios internos en la implementación afecten a otras partes del programa.
3. **Mayor control**: Los métodos que permiten acceder o modificar los datos pueden incluir validaciones o restricciones, lo que asegura que los datos siempre cumplan con ciertas condiciones.
4. **Modularidad**: La encapsulación facilita la división del código en módulos independientes, lo que mejora la organización y la reutilización del código.

2. ¿Qué se entiende por la **interfaz pública** de un objeto o clase en POO? Describe brevemente cómo se relaciona con la ocultación de información.

Respuesta

La interfaz pública de un objeto o clase en Programación Orientada a Objetos (POO) se refiere al conjunto de métodos y atributos que están disponibles para ser utilizados por otras clases o partes del programa. Es la forma en que un objeto expone su funcionalidad al mundo exterior, permitiendo que otras partes del programa interactúen con él sin necesidad de conocer los detalles internos de su implementación.

La relación entre la interfaz pública y la ocultación de información es fundamental. La ocultación de información implica que los detalles internos de una clase, como sus atributos o métodos privados, no son accesibles directamente desde fuera de la clase. En cambio, la interfaz pública actúa como un intermediario, proporcionando acceso controlado a las funcionalidades necesarias mientras protege los datos internos. Esto permite mantener la integridad de los datos y facilita la evolución del código, ya que los cambios internos en la implementación no afectan a las partes externas que interactúan con la clase.

3. Brevemente: ¿Por qué hay que ser conscientes y diseñar con cuidado la **interfaz pública** de una clase? ¿Es fácil cambiarla?

Respuesta

Diseñar con cuidado la interfaz pública de una clase es crucial porque esta define cómo otras partes del programa interactuarán con la clase. Una interfaz pública mal diseñada puede llevar a un uso incorrecto de la clase, dificultar su mantenimiento y limitar su reutilización. Además, una interfaz pública bien definida ayuda a que el código sea más comprensible y fácil de usar por otros desarrolladores.

Cambiar la interfaz pública de una clase no es una tarea sencilla, especialmente si la clase ya está siendo utilizada en múltiples partes del programa. Modificarla puede requerir cambios en todas las dependencias que interactúan con ella, lo que puede ser costoso y propenso a errores. Por esta razón, es importante planificar cuidadosamente la interfaz pública desde el principio, considerando tanto las necesidades actuales como las futuras del sistema.

4. ¿Qué son las **invariantes de clase** y por qué la ocultación de información nos ayuda?

Respuesta

Las invariantes de clase son condiciones o reglas que deben cumplirse en todo momento para garantizar que un objeto se encuentre en un estado válido. Estas condiciones suelen estar relacionadas con los valores de los atributos de la clase y las relaciones entre ellos. Por ejemplo, en una clase que representa un intervalo de números, una invariante podría ser que el valor del límite inferior siempre sea menor o igual al del límite superior.

La ocultación de información contribuye a mantener las invariantes de clase al restringir el acceso directo a los atributos internos. Al obligar a los usuarios de la clase a interactuar con ella a través de métodos públicos, se puede garantizar que cualquier modificación de los datos internos pase por validaciones que aseguren el cumplimiento de las invariantes. Esto reduce el riesgo de errores y mejora la robustez del programa.

5. Pon un ejemplo de una clase `Punto` en `Java`, con dos coordenadas, `x` e `y`, de tipo `double`, con un método `calcularDistanciaAOrigen`, y que haga uso de la ocultación de información. ¿Cuál es la interfaz

pública de la clase `Punto`? ¿Qué significa `public` y `private`?

Respuesta

```
public class Punto {
    private double x;
    private double y;

    public Punto(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public double calcularDistanciaAOrigen() {
        return Math.sqrt(x * x + y * y);
    }
}
```

La interfaz pública de la clase `Punto` está compuesta por los métodos `getX`, `getY`, `setX`, `setY` y `calcularDistanciaAOrigen`. Estos métodos permiten interactuar con los atributos `x` e `y` de manera controlada, sin exponerlos directamente.

El modificador `public` indica que un método o atributo es accesible desde cualquier otra clase. Por otro lado, el modificador `private` restringe el acceso, permitiendo que solo los métodos de la misma

clase puedan acceder o modificar los atributos privados. Esto asegura que los datos internos estén protegidos y que solo puedan ser manipulados de manera controlada.

6. En Java, ¿A quiénes se pueden aplicar los modificadores `public` o `private` ?

Respuesta

En Java, los modificadores `public` y `private` se pueden aplicar a:

1. **Clases**: Una clase puede ser declarada como `public` si se desea que sea accesible desde cualquier otro paquete. Si no se especifica ningún modificador, la clase tendrá visibilidad por defecto (package-private), lo que significa que solo será accesible desde otras clases dentro del mismo paquete.
2. **Atributos**: Los atributos de una clase pueden ser `public` para que sean accesibles desde cualquier lugar, o `private` para que solo sean accesibles dentro de la misma clase. Esto permite controlar el acceso a los datos y proteger la integridad de los mismos.
3. **Métodos**: Los métodos pueden ser `public` para que puedan ser invocados desde cualquier otra clase, o `private` para que solo puedan ser utilizados dentro de la clase en la que están definidos. Esto es útil para encapsular la lógica interna de la clase.
4. **Constructores**: Los constructores también pueden ser `public` o `private` . Un constructor `public` permite crear instancias de la clase desde cualquier lugar, mientras que un constructor `private` restringe la creación de instancias, lo que puede ser útil en patrones de diseño como el Singleton.

7. En POO, la visibilidad puede ser pública o privada, pero ¿existen más tipos de visibilidad? ¿Qué ocurre en Java? ¿Y en otros lenguajes?

Respuesta

Además de la visibilidad pública y privada, en Java existen otros dos niveles de visibilidad: `protected` y el nivel de visibilidad por defecto (también conocido como *package-private*).

1. **`protected`** : Los miembros con este modificador son accesibles desde la misma clase, desde las clases del mismo paquete y desde las subclases, incluso si estas se encuentran en un paquete

diferente. Este nivel de visibilidad es útil cuando se desea permitir que las subclases accedan a ciertos miembros, pero no otras clases externas.
2. **Visibilidad por defecto (package-private)**: Si no se especifica ningún modificador de acceso, los miembros de la clase son accesibles únicamente desde otras clases dentro del mismo paquete. Este nivel de visibilidad es útil para organizar el código en paquetes y limitar el acceso a los miembros solo a las clases relacionadas.

En otros lenguajes de programación orientados a objetos, como C++, existen conceptos similares, aunque con algunas diferencias. Por ejemplo, en C++ también se utiliza `public`, `private` y `protected`, pero además se puede especificar la visibilidad de los miembros en bloques dentro de la definición de la clase. Otros lenguajes, como Python, no tienen modificadores de acceso explícitos, pero utilizan convenciones como el uso de guiones bajos para indicar que un atributo o método es privado.

8. Responde: Los miembros de instancia privados de un objeto están ocultos para (a) otras clases o (b) otras instancias, aunque sean de la misma clase. Pon un ejemplo añadiendo un método `calcularDistanciaAPunto(Punto otro)` y explica la respuesta.

Respuesta

Los miembros de instancia privados de un objeto están ocultos para otras clases, pero no para otras instancias de la misma clase. Esto significa que, aunque un atributo privado no puede ser accedido directamente desde una clase externa, sí puede ser accedido por métodos de la misma clase, incluso si estos métodos operan sobre otra instancia de la misma clase.

Por ejemplo, en la clase `Punto`, se puede implementar un método `calcularDistanciaAPunto` que calcule la distancia entre el punto actual y otro punto pasado como parámetro. Aunque los atributos `x` e `y` son privados, el método puede acceder a ellos porque ambos puntos son instancias de la misma clase.

```
public class Punto {
    private double x;
    private double y;

    public Punto(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double calcularDistanciaAPunto(Punto otro) {
        double deltaX = this.x - otro.x;
        double deltaY = this.y - otro.y;
        return Math.sqrt(deltaX * deltaX + deltaY * deltaY);
    }
}
```

En este ejemplo, el método `calcularDistanciaAPunto` accede directamente a los atributos privados `x` e `y` del objeto `otro`, ya que ambos son instancias de la misma clase `Punto`. Esto demuestra que los miembros privados no están ocultos entre instancias de la misma clase, pero sí lo están para otras clases.

9. ¿Qué son los métodos "getter" y "setter" en los lenguajes orientados a objetos?

Respuesta

Los métodos "getter" y "setter" son funciones utilizadas en los lenguajes orientados a objetos para acceder y modificar los atributos privados de una clase. Estos métodos permiten implementar la encapsulación, ya que proporcionan un control sobre cómo se accede y se modifica el estado interno de un objeto.

Un método "getter" se utiliza para obtener el valor de un atributo privado. Por lo general, su nombre comienza con la palabra `get` seguida del nombre del atributo con la primera letra en mayúscula. Por ejemplo, si un atributo se llama `nombre`, el método getter correspondiente sería `getNombre`.

Por otro lado, un método "setter" se utiliza para modificar el valor de un atributo privado. Su nombre suele comenzar con la palabra `set` seguida del nombre del atributo con la primera letra en mayúscula. Por ejemplo, para el atributo `nombre`, el método setter sería `setNombre`. Este método

puede incluir validaciones para asegurarse de que el nuevo valor sea válido antes de asignarlo al atributo.

El uso de getters y setters es una práctica común en POO porque permite mantener el principio de ocultación de información, al mismo tiempo que proporciona una forma controlada y segura de interactuar con los datos internos de un objeto.

10. Cuando nos referimos a que la ocultación de información mejora la "seguridad" del programa, ¿nos referimos a que no pueda ser "hackeado"?

Respuesta

Cuando se dice que la ocultación de información mejora la "seguridad" del programa, no se refiere a la protección contra ataques externos o ciberseguridad, sino a la capacidad de proteger la integridad de los datos y el correcto funcionamiento del programa. La ocultación de información permite que los datos internos de una clase estén protegidos contra accesos o modificaciones indebidas, lo que reduce la posibilidad de errores y comportamientos inesperados.

Este concepto está relacionado con la idea de que los datos sensibles o críticos de un objeto no deben ser accesibles directamente desde fuera de la clase. En su lugar, se deben proporcionar métodos controlados (como getters y setters) que permitan acceder o modificar los datos de manera segura, aplicando validaciones o restricciones si es necesario. Esto asegura que el objeto siempre se mantenga en un estado válido y predecible.

Por lo tanto, aunque la ocultación de información no protege un programa contra ataques externos, sí contribuye a la robustez y confiabilidad del código, lo que puede considerarse una forma de "seguridad" en el contexto del diseño de software.

11. ¿Qué diferencia hay entre miembro de instancia y miembro de clase? ¿Los miembros de clase también se pueden ocultar?

Respuesta

12. Brevemente: ¿Tiene sentido que los constructores sean privados?

Respuesta

13. ¿Cómo se indican los miembros de clase en Java? Pon un ejemplo, en la clase `Punto` definida anteriormente, para que incluya miembros de clase que permitan saber cuáles son los valores `x` e `y` máximos que se han establecido en todos los puntos que se hayan creado hasta el momento.

Respuesta

14. Como sería un método factoría dentro de la clase `Punto` para construir un `Punto` a partir de dos coordenadas, pero que las redondee al entero más cercano. Escribe sólo el código del método, no toda la clase ¿Has usado `static` ?

Respuesta

15. Cambia la implementación de `Punto` . En vez de dos `double` , emplea un array interno de dos

posiciones, intentando no modificar la interfaz pública de la clase.

Respuesta

16. Si un atributo va a tener un método "getter" y "setter" públicos, ¿no es mejor declararlo público? ¿Cuál es la convención más habitual sobre los atributos, que sean públicos o privados? ¿Tiene esto algo que ver con las "invariantes de clase"?

Respuesta

17. ¿Qué significa que una clase sea inmutable? ¿qué es un método modificador? ¿Un método modificador es siempre un "setter"? ¿Tiene ventajas que una clase sea inmutable?

Respuesta

18. ¿Es recomendable incluir métodos "setter" siempre y como convención?

Respuesta

19. ¿La clase `String` en Java es mutable o inmutable? ¿Qué ocurre al concatenar dos cadenas? ¿Qué debemos hacer si vamos a hacer una operación que

implique concatenar muchas veces para construir paso a paso una cadena muy larga?

Respuesta

20. En POO ¿Cómo se comparan objetos de una misma clase? ¿Por su contenido o por su identidad? ¿Qué es el método equals en Java? ¿Qué hace por defecto? ¿Cómo se deben comparar dos cadenas en Java?

Respuesta

21. ¿Qué son las clases "wrapper" en un lenguaje de programación orientado a objetos? ¿Cómo se hace? ¿Es un proceso automático? ¿Qué ventajas tienen? ¿Todos los lenguajes orientados a objetos tienen tipos primitivos y necesitan wrappers?

Respuesta

22. ¿En POO qué es un tipo de dato enumerado? ¿En Java, un tipo de dato enumerado es una clase? ¿Qué ventajas tienen en términos de encapsulación los enumerados en Java?

Respuesta

23. Crea un tipo enumerado en Java que se llame `Mes`, con doce posibles instancias y que además

proporcione métodos para obtener cuántos días tiene ese mes, el ordinal de ese mes en el año (1-12), empleando atributos privados y constructores del tipo enumerado. Añade además cuatro métodos para devolver si ese mes tiene algunos días de invierno, primavera, verano u otoño, indicando con un booleano el hemisferio (norte o sur, parámetro `enHemisferioNorte`). Es decir:

`esDePrimavera(boolean esHemisferioNorte)`,
`esDeVerano(boolean esHemisferioNorte)`,
`esDeOtoño(boolean esHemisferioNorte)`,
`esDeInvierno(boolean esHemisferioNorte)`

Respuesta