UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programación Orientada a Objetos

Sesión 8:Principio de diseño Program-to-an-interface

Recordando:

¿Que vimos la clase pasada?

Logro de la sesión

- Entender la esencia y el propósito detrás del principio "Program-to-an- interface".
- Reconocer la importancia de programar para interfaces en lugar de implementaciones concretas.
- Ser capaz de aplicar este principio en escenarios de desarrollo reales para mejorar la calidad del código.

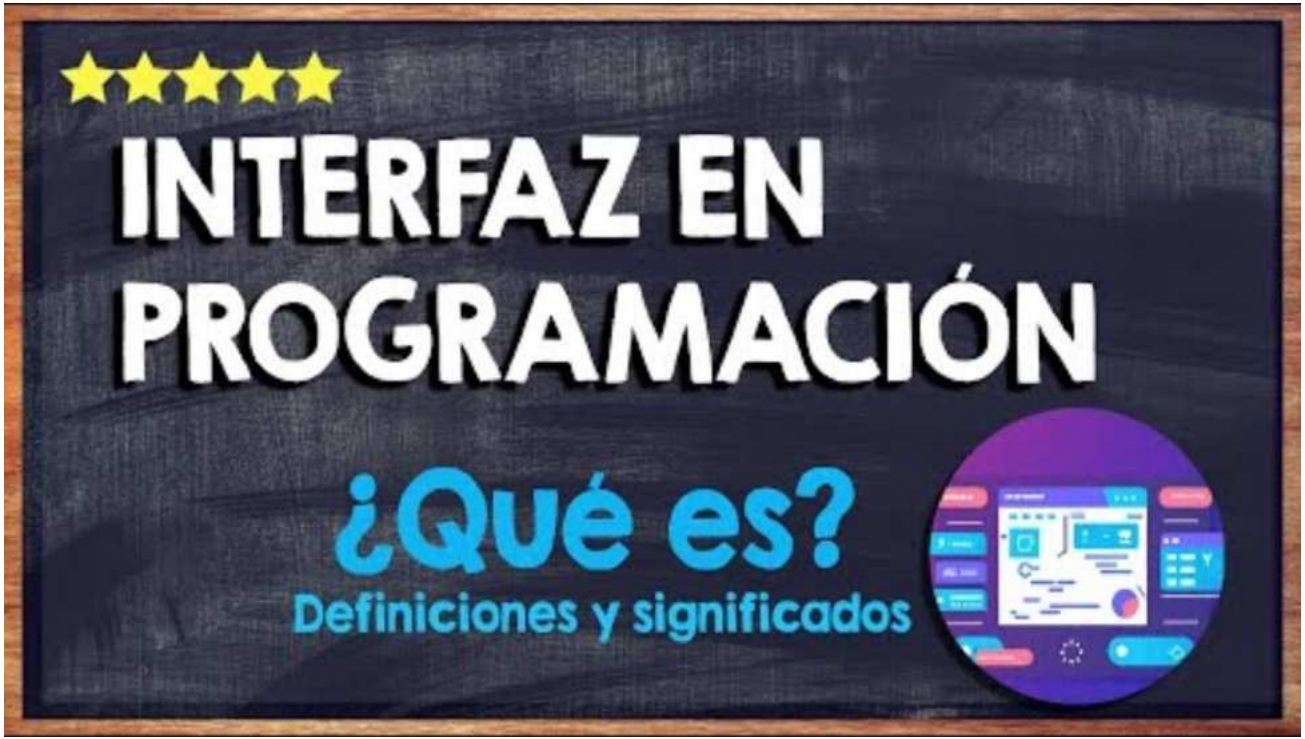
Saberes previos:

- Conceptos básicos de programación orientada a objetos (POO).
- Definición y uso de interfaces en lenguajes de programación.
- Diferencia entre clases e interfaces

Agenda:

- Introducción al principio "Program-to-an-interface
- Detalles y fundamentos del principio
- Importancia y beneficios de "Program-to-an-interface"

Program-to-an-interface



https://www.youtube.com/watch?v=-HyV42VmJeQ

Universidad Tecnológica del Perú

Program-to-an-interface: Detalles y fundamentos del principio

Definición

Program-to-an-interface: Este principio sugiere que los desarrolladores deben programar para una interface en lugar de una implementación concreta. En otras palabras, en lugar de codificar para clases específicas, deberíamos codificar para interfaces abstractas.

Razonamiento detrás del principio

Desacoplamiento: Al programar para una interface, desacoplamos nuestro código de implementaciones específicas. Esto significa que si una implementación cambia en el futuro, no afectará al resto del sistema siempre que la interface se mantenga constante.

Abstracción: Las interfaces proporcionan una capa de abstracción que oculta los detalles de implementación. Esto permite que los desarrolladores se concentren en "qué" hace algo en lugar de "cómo" lo hace.

Ejemplos prácticos

Sistema de pagos:

- **Interface:** ProcesadorDePago
- **Método:** procesarPago(monto)
- **Implementaciones:** PagoConTarjeta, PagoConPayPal, PagoConBitCoin
- Al programar para la interface ProcesadorDePago, podemos cambiar fácilmente entre diferentes métodos de pago sin alterar el código que utiliza esta interface.

Sistema de notificaciones:

- **Interface:** Notificador
- **Método:** enviarNotificacion(mensaje)
- **Implementaciones:** NotificacionPorEmail, NotificacionPorSMS, NotificacionPorApp
- Al usar la interface Notificador, el sistema puede enviar notificaciones a través de diferentes medios sin preocuparse por los detalles específicos de cada medio.

Explicación

•En ambos ejemplos, vemos que las interfaces actúan como contratos que las implementaciones deben seguir. Esto permite que el código que utiliza estas interfaces sea flexible y extensible. Si en el futuro se introduce un nuevo método de pago o un nuevo medio de notificación, simplemente necesitamos crear una nueva clase que implemente la interface correspondiente sin tener que modificar el código existente.

•En resumen, el principio "Program-to-an-interface" promueve la creación de sistemas modulares y desacoplados, donde las implementaciones concretas pueden cambiar y evolucionar sin afectar al resto del sistema. Es una práctica esencial para desarrollar software escalable y mantenible.

Importancia y beneficios de "Program-to-an-interface”

1. Desacoplamiento

Concepto:

El desacoplamiento se refiere a la reducción de dependencias directas entre diferentes partes de un sistema.

Beneficio en Program-to-an-interface

Al programar para una interface en lugar de una implementación especificas , se minimizan las dependencias directas.

•Esto significa que cambios en una implementación no afectarán a las partes del sistema que dependen de la interface, siempre que la interface no cambie.

Importancia y beneficios de "Program-to-an-interface”

2. Flexibilidad

Concepto:

La flexibilidad se refiere a la capacidad de un sistema para adaptarse a cambios sin requerir grandes modificaciones.

Beneficio en "Program-to-an-interface":

Las interfaces actúan como puntos de extensión en un sistema. Si se necesita una nueva implementación o comportamiento, simplemente se puede crear una nueva clase que implemente la interface existente.

Esto permite adaptar y expandir el sistema sin alterar el código existente.

Importancia y beneficios de "Program-to-an-interface”

3. Reutilización

Concepto:

La reutilización de código es la práctica de usar el mismo código en múltiples lugares, reduciendo la duplicación y mejorando la mantenibilidad.

Beneficio en "Program-to-an-interface":

Las interfaces definen comportamientos que pueden ser implementados por múltiples clases. Esto significa que el código que opera en estas interfaces puede reutilizarse para cualquier clase que implemente la interface, promoviendo la reutilización de código y reduciendo la duplicación.

Importancia y beneficios de "Program-to-an-interface”

4. Testabilidad

Concepto:

La testabilidad se refiere a la facilidad con la que se pueden escribir y ejecutar pruebas para un sistema o componente.

Beneficio en "Program-to-an-interface":

Al programar para interfaces, es más fácil crear "mocks" o "stubs" para pruebas unitarias. Estos son objetos que simulan el comportamiento de implementaciones reales. Al desacoplar el código de implementaciones específicas, se puede probar el código en aislamiento, asegurando que funcione correctamente independientemente de las implementaciones concretas.

En resumen…

En resumen, el principio "Program-to-an-interface" es fundamental para crear sistemas robustos, escalables y mantenibles. Al reducir las dependencias directas, proporcionar puntos de extensión claros y promover la reutilización de código, este principio ayuda a los desarrolladores a construir software que puede adaptarse y evolucionar con el tiempo. Además, facilita la tarea de probar el software, asegurando que cumpla con los requisitos y funcione correctamente en diferentes escenarios.

Ejercicio sobre una clínica utilizando el principio "Program- to-an-interface"

```
// Interface para representar servicios generales de un profesional de la salud
interface ProfesionalSalud {
    void consultar();
    void diagnosticar();
}

// Interface para representar acciones específicas de un médico
interface Medico {
    void recetarMedicamento();
}

// Interface para representar acciones específicas de un psicólogo
interface Psicologo {
    void realizarTerapia();
}
```

```
// Clase MedicoGeneral que implementa ProfesionalSalud y Medic
class MedicoGeneral implements ProfesionalSalud, Medico {
    @Override
    public void consultar() {
        System.out.println("Consultando al paciente sobre sus síntomas...");
    }

    @Override
    public void diagnosticar() {
        System.out.println("Diagnosticando la enfermedad del paciente...");
    }

    @Override
    public void recetarMedicamento() {
        System.out.println("Recetando medicamento al paciente...");
    }
}
```

Ejercicio sobre una clínica utilizando el principio "Program- to-an-interface"

```
//Clase PsicologoClinico que implementa ProfesionalSalud y
Psicologo
class PsicologoClinico implements ProfesionalSalud, Psicologo {
    @Override
    public void consultar() {
        System.out.println("Consultando al paciente sobre sus
emociones y pensamientos...");
    }

    @Override
    public void diagnosticar() {
        System.out.println("Diagnosticando el estado emocional del
paciente...");
    }

    @Override
    public void realizarTerapia() {
        System.out.println("Realizando sesión de terapia con el
paciente...");
    }
}
```

```
public class Clinica {
    public static void main(String[] args) {
        ProfesionalSalud doctor = new MedicoGeneral();
        ProfesionalSalud psicologo = new PsicologoClinico();

        System.out.println("Acciones del médico general:");
        doctor.consultar();
        doctor.diagnosticar();
        ((Medico) doctor).recetarMedicamento();

        System.out.println("\nAcciones del psicólogo clínico:");
        psicologo.consultar();
        psicologo.diagnosticar();
        ((Psicologo) psicologo).realizarTerapia();
    }
}
```

Explicación

En este ejercicio, hemos definido una interface ProfesionalSalud que representa acciones generales que cualquier profesional de la salud podría realizar. Luego, hemos definido dos interfaces adicionales, Medico y Psicologo, que representan acciones específicas para cada tipo de profesional.

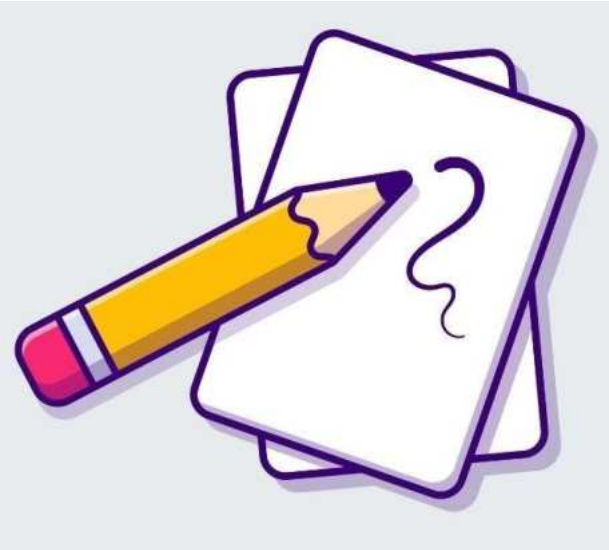
Las clases MedicoGeneral y PsicologoClinico implementan estas interfaces, proporcionando implementaciones específicas para cada método.

Finalmente, en la clase Clinica, creamos instancias de ambos tipos de profesionales y llamamos a sus métodos para demostrar sus acciones. Al programar para la interface ProfesionalSalud, el código se vuelve más flexible y desacoplado, siguiendo el principio "Program-to-an-interface".

Practica:

Implementar la **interface Terapeuta** y agregar un método según tu criterio. Agregar al programa la **clase TerapeutaClinico** y sobreescribir el método agregado en la interface. Crear el objeto en el Main y mostrar las ejecuciones de los métodos.

Conclusiones

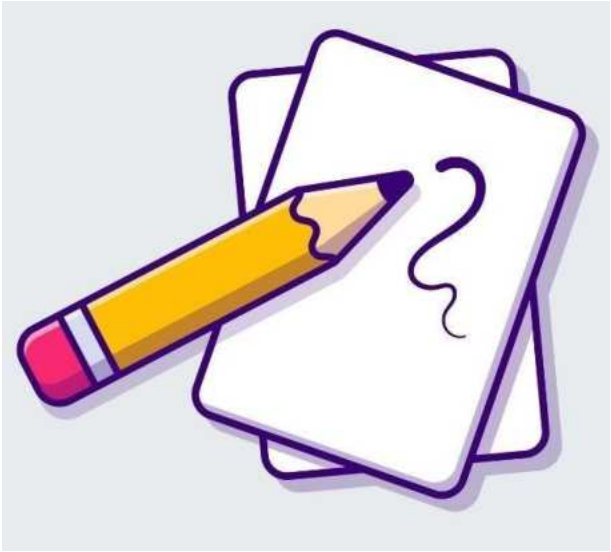


Promoción de Sistemas Robustos y Escalables:

- El principio "Program-to-an-interface" es esencial para el desarrollo de software moderno, ya que promueve la creación de sistemas desacoplados y modulares.
- Al reducir las dependencias directas y programar para interfaces en lugar de implementaciones concretas, los sistemas se vuelven más robustos ante cambios y escalables ante nuevas necesidades.

Universidad Tecnológica del Perú

Conclusiones



Facilitación de la Mantenibilidad y Testabilidad:

- Al adherirse al principio "Program-to-an-interface", se mejora la mantenibilidad del software. Las interfaces actúan como contratos claros que las implementaciones deben seguir, lo que reduce la ambigüedad y mejora la coherencia en el código.
- Además, este enfoque facilita la testabilidad, ya que permite la creación de "mocks" y "stubs" para pruebas unitarias, asegurando que el código se comporte como se espera en diversos escenarios y contextos.

Cierre

¿Qué hemos aprendido hoy?

Universidad Tecnológica del Perú

Bibliografía

- MORENO PÉREZ, J. Programación orientada a objetos.RA-MA Editorial. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933
- Vélez Serrano, José. Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java. Dykinson. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368

Universidad Tecnológica del Perú