UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

# Programacion Orientada a Objetos

## Sesion 8: Interfaces

# Recordando:

¿Que vimos la clase pasada?

# Logro de la sesión

- Comprender la definición y propósito de las interfaces.
- Saber cómo declarar e implementar interfaces en Java.
- Entender la importancia de las interfaces en el diseño de software.

## Agenda:

- Introducción a las interfaces
- ¿Qué son las interfaces en Java?
- Importancia y uso de las interfaces

Universidad Tecnológica del Perú

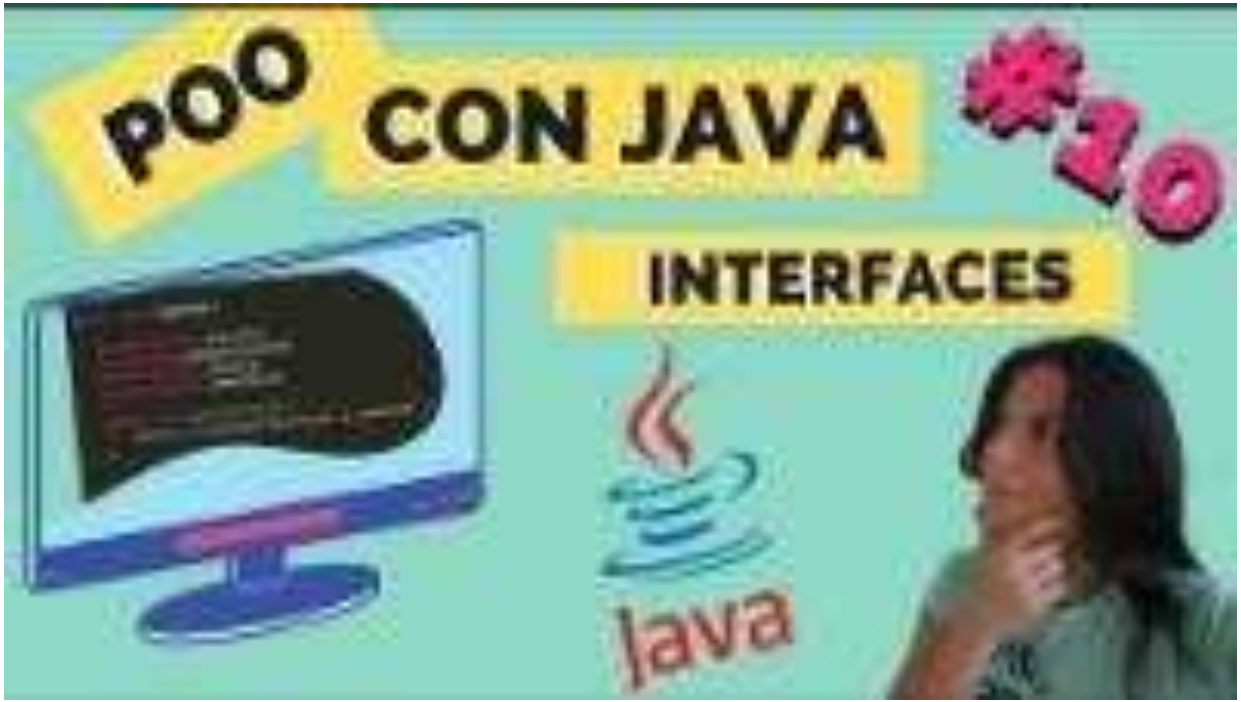
## Saberes previos:

- Definición de clases en Java
- Concepto de herencia y polimorfismo
- Diferencia entre clases abstractas e interfaces

# ¿Qué son las interfaces en Java?:

- Definición y propósito de las interfaces
- Sintaxis básica para declarar una interface
- Implementación de interfaces en clases

# Interfaces




https://www.youtube.com/watch?v=hfwtzjOhvKk

# Definición y propósito de las interfaces

**Definición:** Una interface en Java es un tipo de referencia similar a una clase que puede contener solo constantes, métodos abstractos, métodos predeterminados, métodos estáticos y métodos privados. No puede contener un constructor ya que no se puede instanciar.

**Propósito:** Las interfaces se utilizan para lograr la abstracción total; es decir, todos los métodos en una interface son abstractos por defecto. Además, sirven para establecer un contrato estándar que las clases pueden seguir. Esto es especialmente útil para definir comportamientos que pueden ser implementados por múltiples clases, sin preocuparse por los detalles de implementación.

# Sintaxis básica para declarar una interface

```
public interface NombreDeLaInterface {
    // Constantes (por defecto, public, static y final) int
    MI_CONSTANTE = 10;

    // Métodos abstractos (por defecto, public y abstract) void
    miMetodoAbstracto();

    // Métodos predeterminados (desde Java 8) default void
    miMetodoPredeterminado() {
        // Cuerpo del método
    }

    // Métodos estáticos (desde Java 8) static void
    miMetodoEstatico() {
        // Cuerpo del método
    }
}
```

# Implementación de interfaces en clases

- Una clase puede implementar una o varias interfaces utilizando la palabra clave implements.

- Al implementar una interface, la clase debe proporcionar implementaciones para todos los métodos
abstractos declarados en la interface.

```
public class MiClase implements
   NombreDeLaInterface { @Override
public void miMetodoAbstracto() {
  // Implementación del método
    }
}
```

Universidad Tecnológica del Perú

# Implementación de interfaces en clases

Si una clase no proporciona implementaciones para todos los métodos abstractos de una interface, entonces esa clase debe ser declarada como abstracta.

En resumen, las interfaces en Java proporcionan una forma de definir comportamientos (a través de métodos abstractos) que las clases pueden implementar. Esto permite una mayor flexibilidad y abstracción en el diseño de software, ya que diferentes clases pueden implementar la misma interface de diferentes maneras.

# Importancia y uso delas interfaces

- Promoción del polimorfismo
- Creación de contratos estandarizados para las clases
- Uso de interfaces para simular la herencia múltiple
- Casos prácticos y ejemplos

## Promoción del polimorfismo

**Polimorfismo:** Es la capacidad de un objeto de tomar muchas formas. En Java, el polimorfismo se logra principalmente a través de la herencia y las interfaces.

**Uso con interfaces:** Las interfaces permiten que diferentes clases sean tratadas como instancias de la misma interface. Esto significa que puedes referenciar un objeto de una clase que implementa una interface usando el tipo de la interface en lugar del tipo de la clase

```
interface Animal {
   void hablar();
}

class Perro implements Animal {
   public void hablar() {
      System.out.println("Guau");
   }
}

class Gato implements Animal {
   public void hablar() {
      System.out.println("Miau");
   }
}

public class Test {
   public static void main(String[] args) {
      Animal miPerro = new Perro();
      Animal miGato = new Gato();

      miPerro.hablar();  // Outputs: Guau
      miGato.hablar();  // Outputs: Miau
   }
}
```

# Creación de contratos estandarizados para las clases

Las interfaces actúan como contratos que las clases deben seguir. Cuando una clase implementa una interface, está garantizando que proporcionará una implementación específica para todos los métodos declarados en esa interface.

Esto es útil en situaciones donde diferentes clases deben adherirse a un conjunto estándar de operaciones pero pueden tener diferentes implementaciones internas.

**Uso de interfaces para simular la herencia múltiple**

Java no admite la herencia múltiple directamente a través de clases debido a problemas como la "ambigüedad del diamante". Sin embargo, las interfaces ofrecen una solución a esto.

Una clase puede implementar múltiples interfaces, lo que permite que una clase herede comportamientos (métodos) de múltiples fuentes.

```
interface A {
   void metodoA();
}

interface B {
   void metodoB();
}

class ClaseC implements A, B {
   public void metodoA() {
      // Implementación del método A
   }

   public void metodoB() {
      // Implementación del método B
   }
}
```

## Casos prácticos y ejemplos

**Callbacks:** Las interfaces se utilizan a menudo en Java para implementar callbacks, donde un objeto proporciona una referencia a un método de otro objeto y luego lo llama en respuesta a algún evento.

**APIs y bibliotecas:** Las interfaces son fundamentales en el diseño de APIs y bibliotecas en Java, ya que permiten que los desarrolladores sepan qué métodos están disponibles sin revelar la implementación interna.

**Adaptadores:** Las interfaces pueden ser utilizadas para crear patrones de diseño como el adaptador, donde una clase convierte la interfaz de una clase existente en otra interfaz esperada por los clientes.

- **En resumen,** las interfaces en Java son herramientas poderosas que ofrecen flexibilidad en el diseño de software. Permiten la creación de contratos estandarizados, promueven el polimorfismo y ofrecen una solución a la falta de herencia múltiple en Java. Además, son fundamentales en muchos patrones de diseño y en la creación de APIs robustas.

# Ejercicio sobre un estudio jurídico

```
// Interface para representar acciones legales generales
interface AccionLegal {
    void consultar();
    void presentarDocumentos();
}

// Interface para representar acciones específicas de un abogado penalista
interface AbogadoPenalista {
    void defenderEnJuicio();
}

// Interface para representar acciones específicas de un abogado civilista
interface AbogadoCivilista {
    void redactarContrato();
}
```

```
// Clase AbogadoPenal que implementa AccionLegal y AbogadoPenalista
class AbogadoPenal implements AccionLegal, AbogadoPenalista {
    @Override
    public void consultar() {
        System.out.println("Consultando sobre un caso penal...");
    }

    @Override
    public void presentarDocumentos() {
        System.out.println("Presentando documentos para el caso penal...");
    }

    @Override
    public void defenderEnJuicio() {
        System.out.println("Defendiendo al cliente en el juicio penal...");
    }
}
```

```
// Clase AbogadoCivil que implementa AccionLegal y AbogadoCivilista
class AbogadoCivil implements AccionLegal, AbogadoCivilista {
    @Override
    public void consultar() {
        System.out.println("Consultando sobre un caso civil...");
    }

    @Override
    public void presentarDocumentos() {
        System.out.println("Presentando documentos para el caso civil...");
    }

    @Override
    public void redactarContrato() {
        System.out.println("Redactando contrato civil...");
    }
}
```

```
public class EstudioJuridico {
    public static void main(String[] args) {
        AbogadoPenal abogadoPenal = new AbogadoPenal();
        AbogadoCivil abogadoCivil = new AbogadoCivil();

        System.out.println("Acciones del abogado penalista:");
        abogadoPenal.consultar();
        abogadoPenal.presentarDocumentos();
        abogadoPenal.defenderEnJuicio();

        System.out.println("\nAcciones del abogado civilista:");
        abogadoCivil.consultar();
        abogadoCivil.presentarDocumentos();
        abogadoCivil.redactarContrato();
    }
}
```

# Explicación

En este ejercicio, hemos definido una interface AccionLegal que representa acciones generales que cualquier abogado podría realizar. Luego, hemos definido dos interfaces adicionales, AbogadoPenalista y AbogadoCivilista, que representan acciones específicas para cada tipo de abogado.

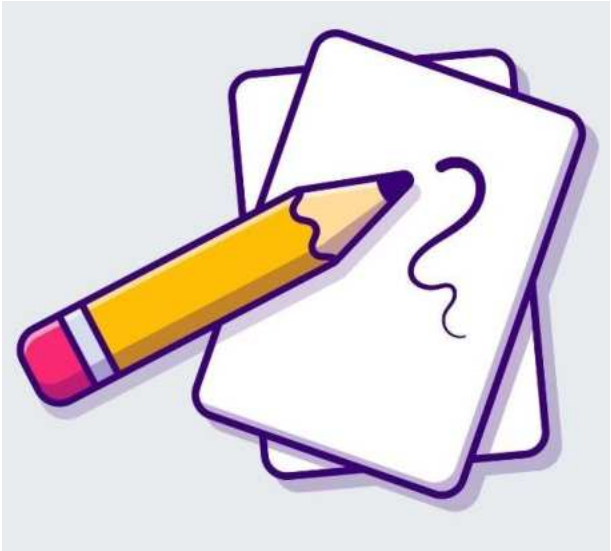
Las clases AbogadoPenal y AbogadoCivil implementan estas interfaces, proporcionando implementaciones específicas para cada método.

Finalmente, en la clase EstudioJuridico, creamos instancias de ambos tipos de abogados y llamamos a sus métodos para demostrar sus acciones.

# Practica:

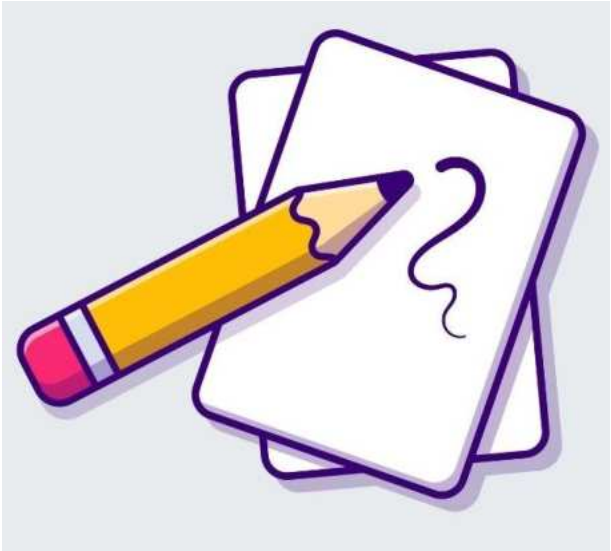
Desarrollar la Interface AbogadoFamiliaridad y agregar un método tenenciayCustodia y posteriormente crear la clase AbogadoFamiliar sobreescribir el método y crear el objeto en el Main.

# Conclusiones



- Las interfaces en Java son esenciales para lograr una abstracción completa, permitiendo que diferentes clases sean tratadas bajo un mismo tipo de referencia. Esto no solo promueve el polimorfismo, sino que también garantiza flexibilidad en el diseño de software.
- Al actuar como contratos estandarizados, las interfaces aseguran que las clases que las implementan sigan un conjunto específico de comportamientos, lo que facilita la interoperabilidad y cohesión en sistemas más grandes.

# Conclusiones



- Aunque Java no permite la herencia múltiple directamente a través de clases, las interfaces ofrecen una solución elegante a esta limitación.
- Una clase puede implementar múltiples interfaces, lo que le permite heredar comportamientos de diversas fuentes. Esta característica no solo amplía las capacidades de diseño en Java, sino que también evita problemas asociados con la herencia múltiple, como la ambigüedad del diamante.

# Cierre

¿Qué hemos aprendido hoy?

# 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