

Interfaces

UNIDAD 5: HERENCIA, POLIMORFISMO E INTERFACES



Interfaces (Que es ?)

Las **interfaces** definen y estandarizan las formas en que pueden interactuar las cosas entre sí, como las personas y los sistemas.

La **interfaz** especifica las operaciones que pueden realizar los objetos, pero no **cómo** lo hacen.

Ej. Los controles en un radio sirven como una **interfaz** entre los usuarios del radio y sus componentes internos.



Radios con diferentes interfaces

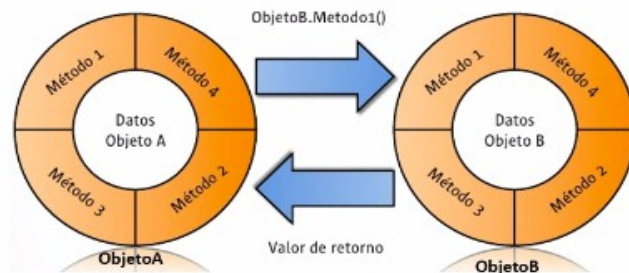
UNIDAD 5: HERENCIA, POLIMORFISMO E INTERFACES



Interfaces (Que es ?)

Los objetos de software también se comunican a través de interfaces.

Una **interfaz** describe un conjunto de métodos que pueden llamarse sobre un objeto



Una **interfaz** en Java es la especificación de un tipo (bajo la forma de un nombre de tipo y un conjunto de métodos) que no define ninguna implementación para los métodos.



Interfaces (Características?)

- En el encabezado de la declaración se usa la palabra clave **interface** en lugar de **class**.
- Todos los métodos de una interfaz son abstractos; no se permiten métodos con cuerpo. No es necesaria la palabra clave **abstract** en su definición.
- Las interfaces no contienen ningún constructor.
- Todas las firmas de los métodos de una interfaz tienen visibilidad pública. No es necesario declarar la visibilidad, es decir, no es necesario que cada método contenga la palabra clave **public**.
- En una interfaz, sólo se permiten como campos las constantes (campo público, estático y final). Pueden omitirse las palabras clave **public**, **static** y final pero todos los campos, igualmente, serán tratados como públicos, estáticos y finales.
- Una clase puede derivar de una interfaz de la misma manera en que deriva de una clase. Sin embargo, Java utiliza una palabra clave diferente, **implements**, para la herencia a partir de interfaces.



```
public interface IPersona {
    public abstract void operacion1();
    public abstract void operacion2();
}

«interface» IPersona
+Operation1()
+Operation2()

«interface» ITrabajador
+aporteSalud
+Operation3()
+Operation4()

«abstract» Persona

public class Persona implements IPersona {

    @Override
    public void operacion1() {
    }

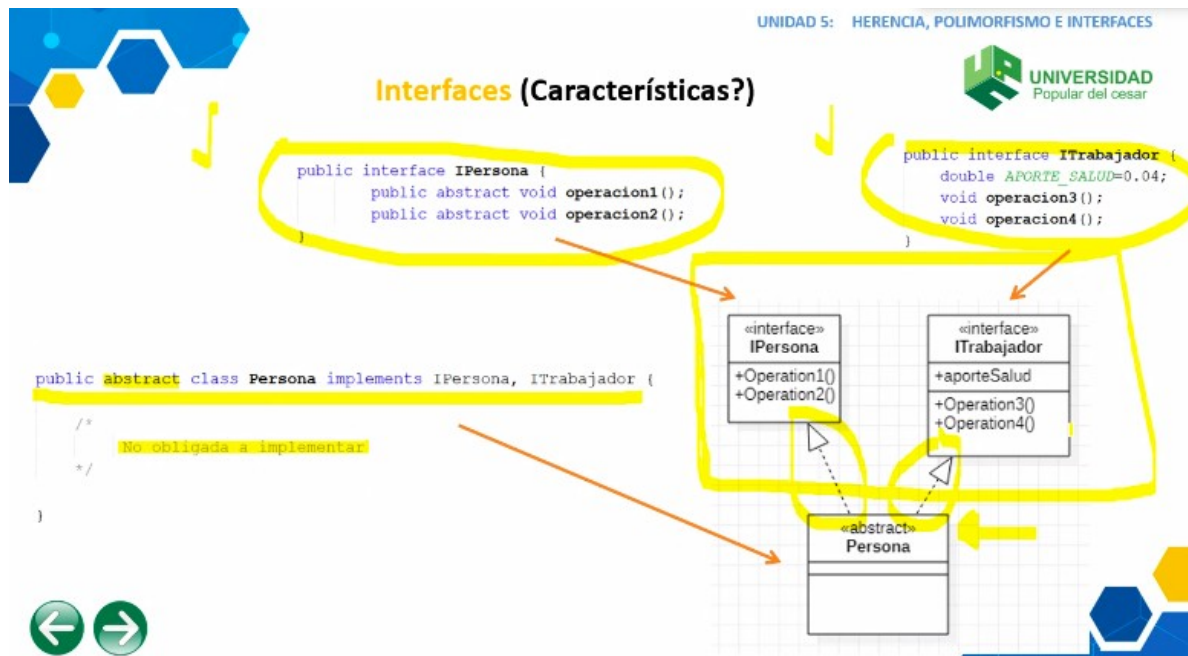
    @Override
    public void operacion2() {
    }
}
```

con las interfaces podemos aplicar herencia multiplica:

Coloocamos en el Implement todas las herencias que requerimos:

Implemente Personai, Trabajari por ejemplo.

Ejm con clase abstractas...



Interfaces (Herencia Múltiple con Interfaces)

La Herencia Múltiple presentará problemas cuando:

- Se herede varias veces de una misma clase base.
- Se herede métodos implementados de forma diferente que se llamen igual.

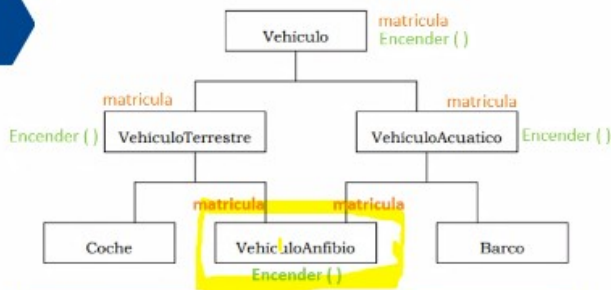
La flexibilidad de algunos lenguajes gracias a la H.M, deben pagarla en complejidad o ineficiencia.

El conflicto de la herencia múltiple ocurre porque una clase derivada puede recibir:

- Varias implementaciones para un mismo mensaje.
- Varias copias de un mismo atributo.

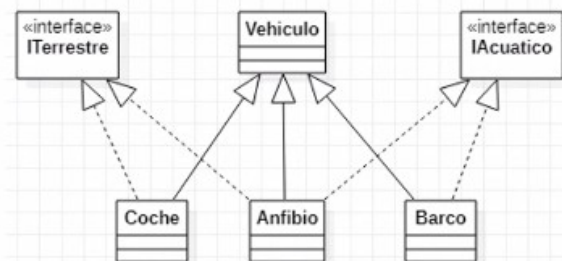


Interfaces (Herencia Múltiple con Interfaces)



¿Qué pasaría si la clase **Vehículo** tuviera un atributo llamado **matricula**? ¿Cuántas matrículas tendría la clase **VehículoAnfibio**?

¿qué pasaría si **Vehículo** tuviera un método que fuera implementado de forma diferente por **VehículoTerrestre** y por **VehículoAcuatico**.



Un **interface** se pueden indicar como “padres” de una jerarquía múltiple.

Es decir, con los interfaces se obtiene lo bueno de la herencia múltiple, eliminando lo malo. No se hereda codificación sólo definición de métodos.



Interfaces (Interfaces como tipos)

Una **interfaz** define un tipo tal como lo hace una clase. Esto quiere decir que **las variables pueden ser declaradas del tipo de la interfaz**, aun cuando no pueda existir ningún objeto de tal tipo (sólo de los subtipos).

```
IPersona ip = new Persona();
ip.Operacion1();
```

```
ITrabajador it = new Persona();
it.Operacion3();
```

```
public interface IPersona {
    public abstract void operacion1();
    public abstract void operacion2();
}
```

```
public interface ITrabajador {
    double APORTE_SALUD=0.04;
    void operacion3();
    void operacion4();
}
```



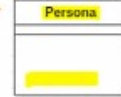
```
public class Persona implements IPersona, ITrabajador {
```

```
    @Override
    public void operacion1() { /*...*/ }
```

```
    @Override
    public void operacion2() { /*...*/ }
```

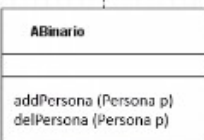
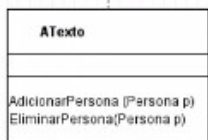
```
    @Override
    public void operacion3() { /*...*/ }
```

```
    @Override
    public void operacion4() { /*...*/ }
```



Cuando se crean upcasting o downcasting, es decir, se hace una generalización, solamente se puede acceder a los metodos que estén en el tipo de objeto que se creo. Por ejemplo: ip solo puede acceder a los metodos de su tipo.

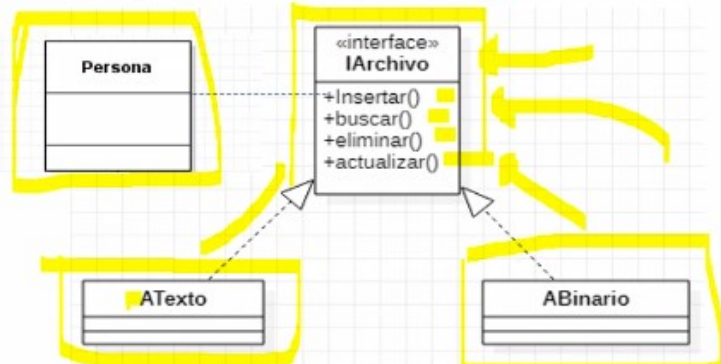
Interfaces (Interfaces como especificaciones)



```
Persona p = new Persona();
ATexto at = new ATexto();
at.AdicionarPersona(p);
```

```
Persona p = new Persona();
ABinario at = new ABinario();
at.addPersona(p);
```

Si en lugar de usar un **ATexto** o **ABinario**, como tipo de variable y tipo de parámetro usamos siempre **IArchivo**, nuestra aplicación funcionará independientemente del tipo específico de archivo que queramos implementar.

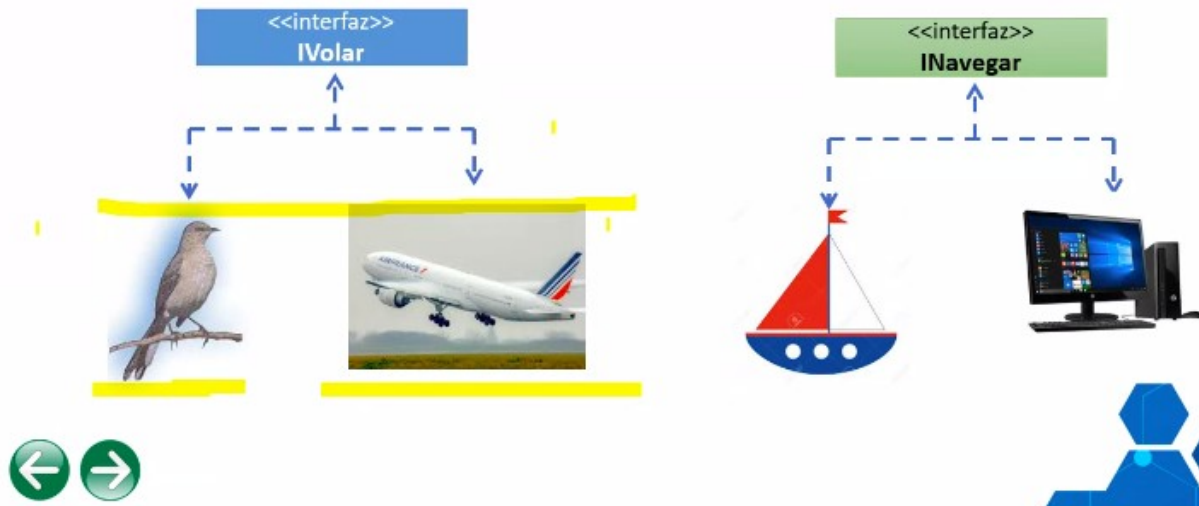


```
Persona p = new Persona();
IArchivo at = new ATexto();
at.AdicionarPersona(p);
```



Interfaces (Polimorfismo mediante interfaces)

Las **interfaces** permite implementar de forma polimórfica un conjunto de métodos comunes para clases no relacionadas .



Clases abstractas Vs Interfaces

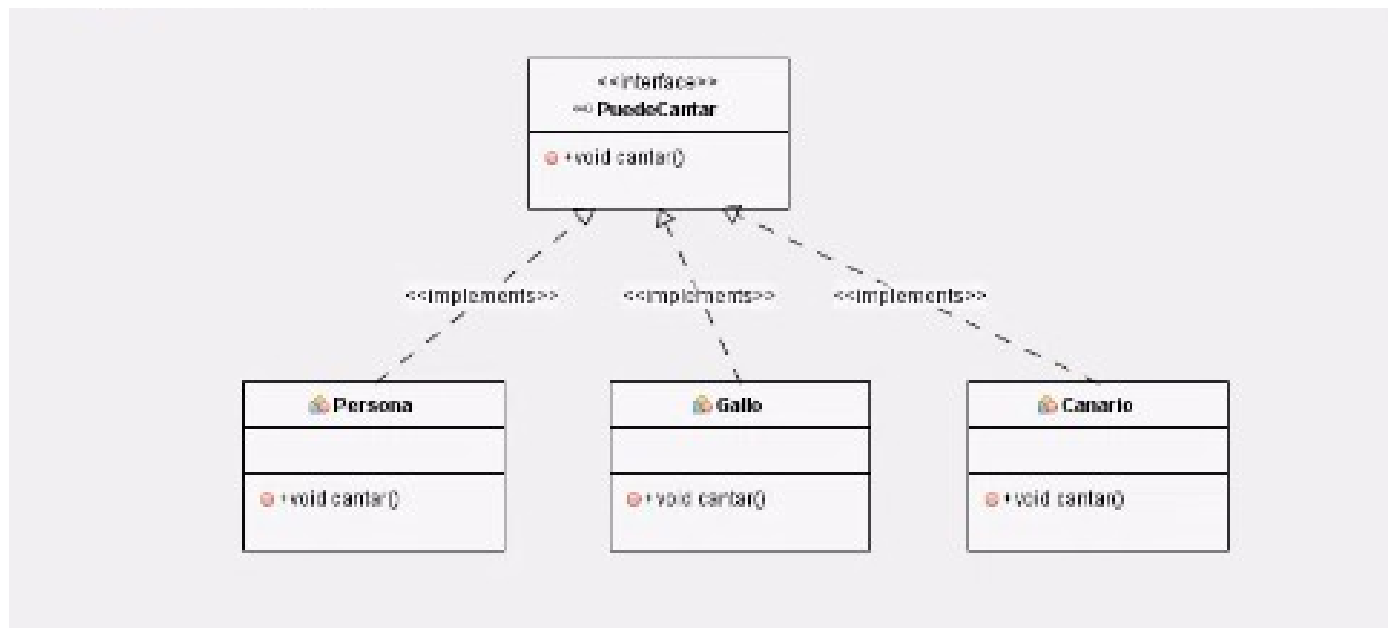
	Clases	Interfaces
Tipo	Tipo que al extenderlo mediante herencia se obtienen sus mensajes y sus implementaciones (métodos y atributos).	Tipo que al extenderlo mediante herencia se obtiene solamente mensajes.
Ventajas	Menor codificación al crear nuevos subtipos ya que los mensajes vienen con sus implementaciones.	Se pueden heredar varios interfaces sin conflictos.
Inconvenientes	Sólo se puede derivar de una de ellas.	Hay que codificar el método de cada mensaje en cada subtipo.

Interfaces útiles de java:
 Serializable, Cloneable,
 Enumeration, Comparable,
 List, Comparator

Quando se debe utilizar **interfaces** o cuando se debe utilizar **Clases abstractas** ??



Ejercicio1)



Ejercicio2)

