UCA

Programación Orientada a Objetos I

JAVA – CLASES (IV)

Interfaces

- Las interfaces JAVA son expresiones puras de diseño.
- Son **conceptualizaciones no implementadas** que sirven de guía para **definir un determinado concepto y su comportamiento**, pero **sin desarrollar un mecanismo de solución**.
- El uso de interfaces proporciona las siguientes ventajas:
  - Obligar a que ciertas clases utilicen los mismos métodos (nombres y parámetros).
  - Es un mecanismo para abstraer métodos a un nivel superior.
  - Establecer relaciones entre clases que no estén relacionadas
  - Implementar herencia múltiple.
  - Organizar la programación.

Interfaces

- En JAVA **no está soportada la herencia múltiple**
- Para poder utilizar la potencia de la herencia múltiple, Java introduce este concepto de interface.
- Se trata de declarar métodos abstractos o virtuales y algunas constantes que puedan ser implementados de diferentes maneras según las necesidades de cada sistema en particular.
- Una interfaz es formalmente como una clase, con ciertas diferencias:
  - Se emplea la palabra reservada "*interface*" en lugar de "*class*".
  - Algunos de sus métodos pueden no estar definidos, solamente declarados (métodos abstractos).
  - Puede contener métodos implementados, llamados métodos default o métodos de extensión virtual (desde JAVA 8).
  - Sus atributos deben ser tratados como *public*, *static* y *final* (default).

Interface

- Declaración de una Interfaz en JAVA:

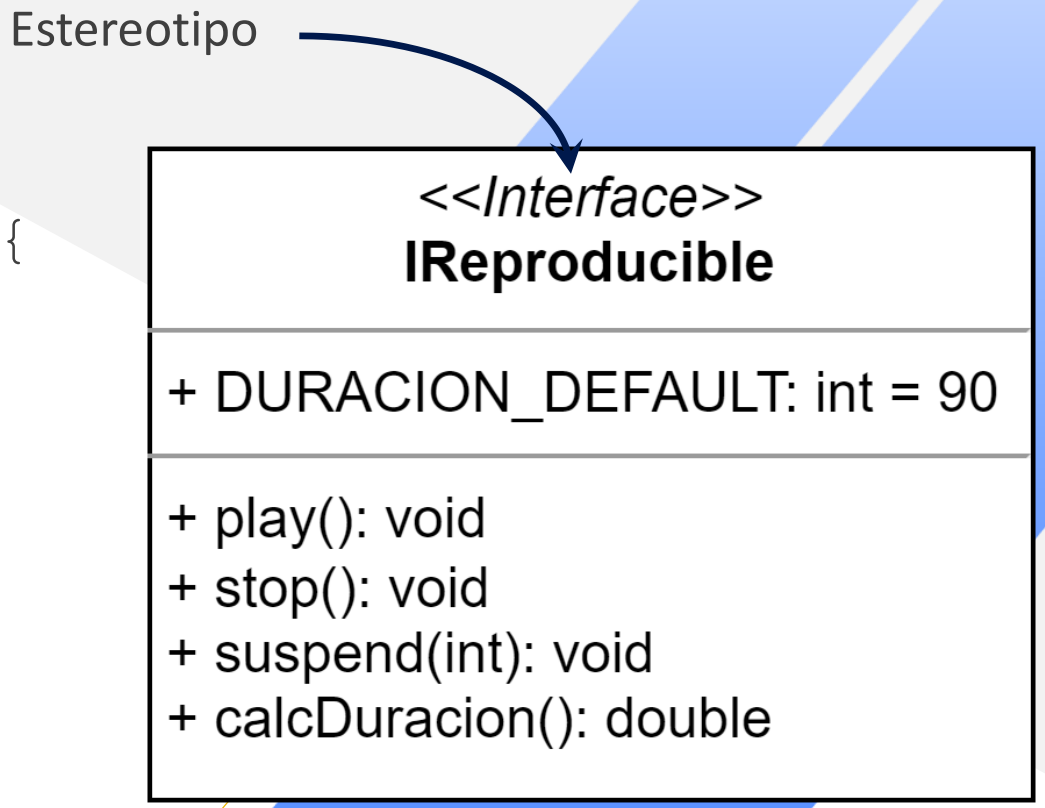
```
public interface INombreInterfaz
        { // declaración de la interfaz; }
```

- Por convención:
  - Se utilizan adjetivos (por ser acciones realizables)
  - Comienzan con la letra "I".
- Cada interfaz debe ser definida en un fichero .java con el mismo nombre de la interface (son *public* por defecto)
- Las interfaces sólo admiten los modificadores de acceso *public* y *package*.
- Los métodos son implícitamente *public* y *abstract*.
- Los atributos son implícitamente *public*, *final* y *static*.

Interface

Ejemplo:

```
public interface IReproducible {
    int DURACION_DEFAULT = 90;

    public void play();
    public void stop();
    public void suspend (int m);
    public double calcDuracion();
}
```



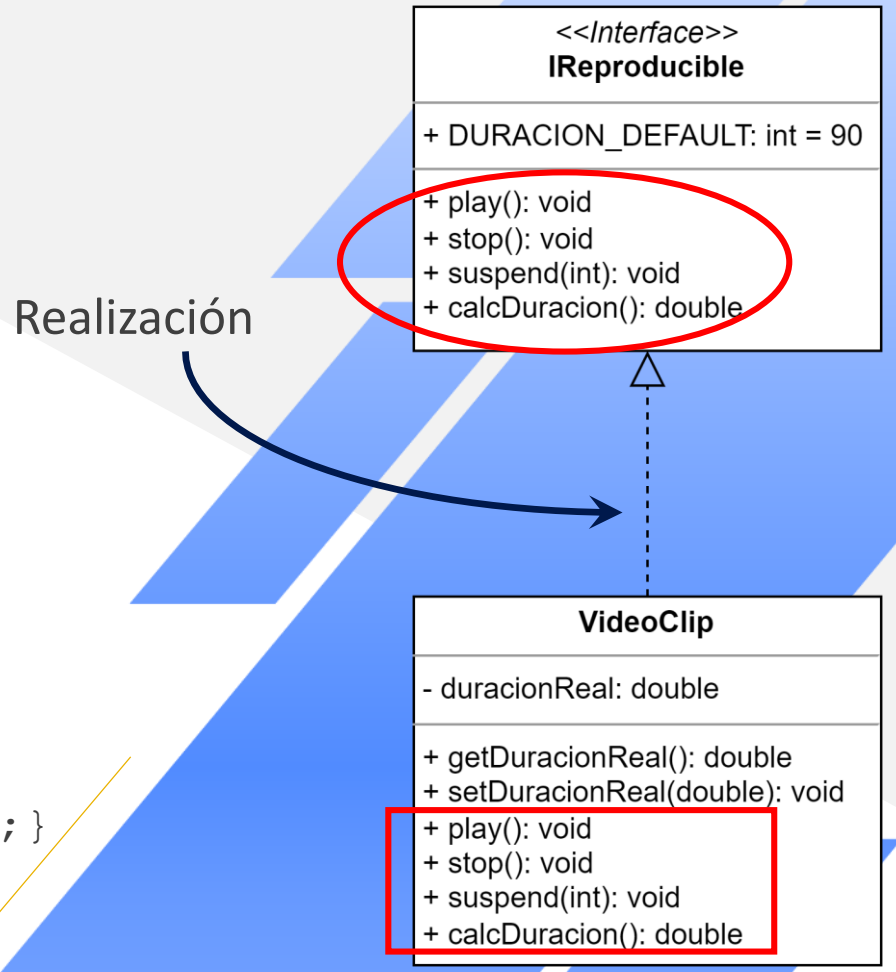
Interface

- Las interfaces carecen de funcionalidad dado que sus métodos, generalmente, no están definidos. Necesitan algún mecanismo que sí lo haga: las clases.
- Las interfaces deben ser implementadas a través de clases, las cuales son las encargadas de proporcionar la definición de sus métodos.
- Si una clase no sobrescribe alguno de sus métodos abstractos, automáticamente, esta clase se convierte en abstracta.
- Una interfaz define un tipo de conducta de la clase que la implementa: asume las constantes de la interfaz y codifica sus métodos.
- Para indicar que una clase implementa una interfaz, se agrega la palabra reservada "*implements*" en la definición de la clase.

```
public class NombreClase implements INombreInterfaz {
// definición de la clase
// debe sobre-escribir todos los métodos abstractos de la interfaz
}
```

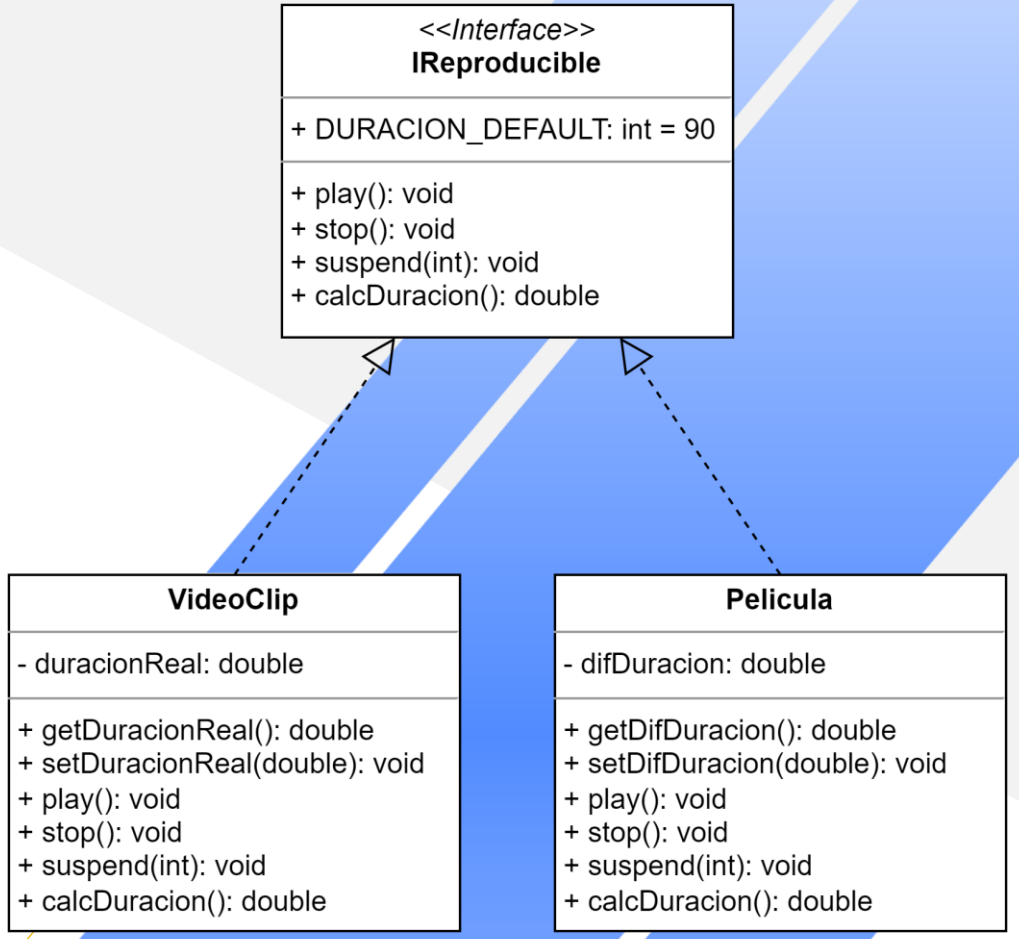
Interface

```
public class VideoClip implements IReproducible {
    private double duracionReal;

    public VideoClip(){}
    public VideoClip(double pDurac){
        duracionReal = pDurac;}
    public double getDuracionReal(){
        return duracionReal;}
    public void setDuracionReal(double durReal){
        duracionReal = durReal;}
    public void play(){
        System.out.print("Tecla PLAY");}
    public void stop(){
        System.out.print("Tecla STOP");}
    public void suspend(int min){
        System.out.print("Detenido por "+min+" min");}
    public double calcDuracion(){
        return duracionReal;}
}
```



Interface

```
public class Peliculas implements IReproducible {
    private double difDuracion;

    public Peliculas(){}
    public Peliculas(double difDuracion){
        this.difDuracion = difDuracion;}
    public double getDifDuracion(){
        return difDuracion;}
    public void setDifDuracion(double difDuracion){
        this.difDuracion = difDuracion;}
    public void play(){
        System.out.println("Otro PLAY");}
    public void stop(){
        System.out.println("Otro STOP");}
    public void suspend(int min){}
        // definirlo aunque no haga nada
    public double calcDuracion(){
        return DURACION_DEFAULT + difDuracion;}
}
```



Interface para agrupación de constantes

- Por definición, todos los datos miembros en una interface son static y final
- Eso significa que resulta adecuado para implementar constantes o grupos de constantes.
- Ejemplo:

```
public interface IMeses{
    int ENERO = 1, FEBRERO = 2, ...;
    String [] NOMBRES_MESES = {"","Enero","Febrero",...};
}
```

- Entonces, se podría usar así:

```
System.out.print(IMeses.NOMBRES_MESES[IMeses.ENERO]);
System.out.print(IMeses.NOMBRES_MESES
                [(fecha.get(Calendar.MONTH)+1)]);
                // fecha = objeto de la clase Calendar
```

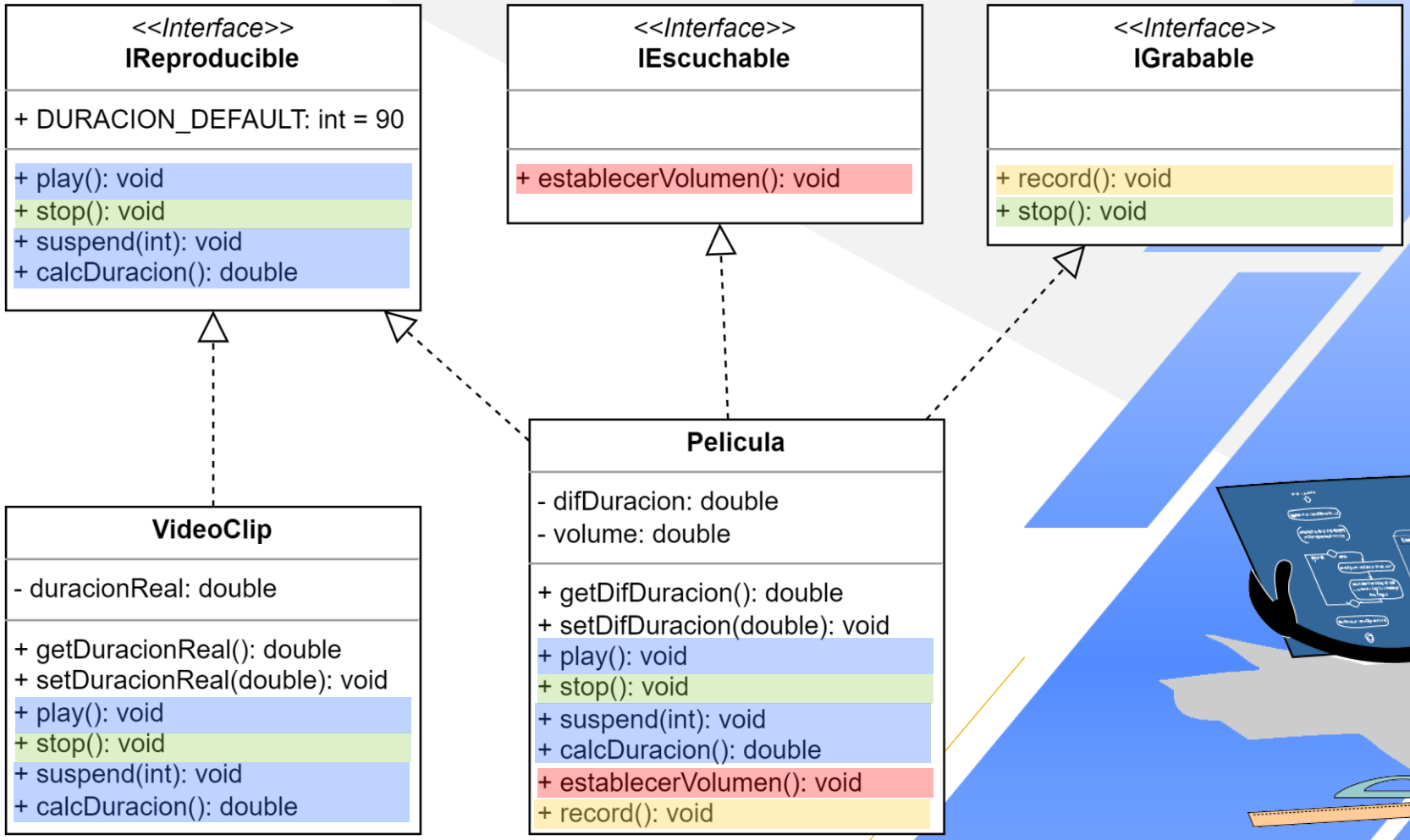
Herencia múltiple

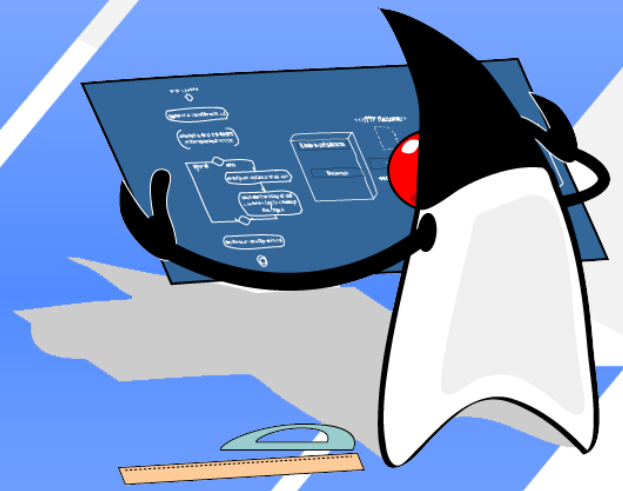
- Una clase puede implementar varias interfaces, con lo cual se obtiene un uso semejante al de herencia múltiple.
- La lista de las interfaces se indican luego de la palabra *implements* separadas por coma.
- Ejemplo:

```
public class ClaseA implements InterfaceA, InterfaceB{
    // … vendrá el código de la clase
}
```

- Donde *ClaseA* es la clase que **implementa** a las interfaces *InterfaceA* e *InterfaceB*

Herencia múltiple





Herencia múltiple

```
public interface IGrabable {
    public void record();
    public void stop();
}
public interface IEscuchable{
        public void establecerVolumen();
}
public class Peliculas implements IReproducible, IGrabable, IEscuchable {
    private double difDuracion, volumen;
    //... más los métodos requeridos por las nuevas interfaces:
    public void record(){
        System.out.println("Presione la tecla RECORD");
    }
    public void establecerVolumen(){
        System.out.println("Setee el volumen en" + volumen);
    }
}
```

Herencia múltiple

- Con la adicción de **métodos por defecto**, las clases pueden heredar **diferentes comportamientos de múltiples interfaces**.
- En caso de conflictos, el orden que se selecciona el método es el siguiente:
  - Implementaciones en clases concretas.
  - Implementaciones en subinterfaces.
- Se puede explícitamente indicar el método deseado, utilizando la referencia super entre el nombre de la interfaz y el método:

```
Interfaz.super.metodo()
```

Herencia múltiple

Ejemplo:

```
public interface ISaludoTarde {
	default void saludo(){
		System.out.println("Buenas tardes");}
}

public interface ISaludoNoche {
	default void saludo(){
		System.out.println("Buenas noches");}
}

public class Saludos implements ISaludoTarde, ISaludoNoche {
	public void saludo(){
		ISaludoTarde.super.saludo(); }    // para saber cuál invocar
 }
```

Herencia de interfaces

- Una clase no puede tener más que una clase antecesora (no hay herencia múltiple entre clases)
- Una clase puede implementar más de una interfaz

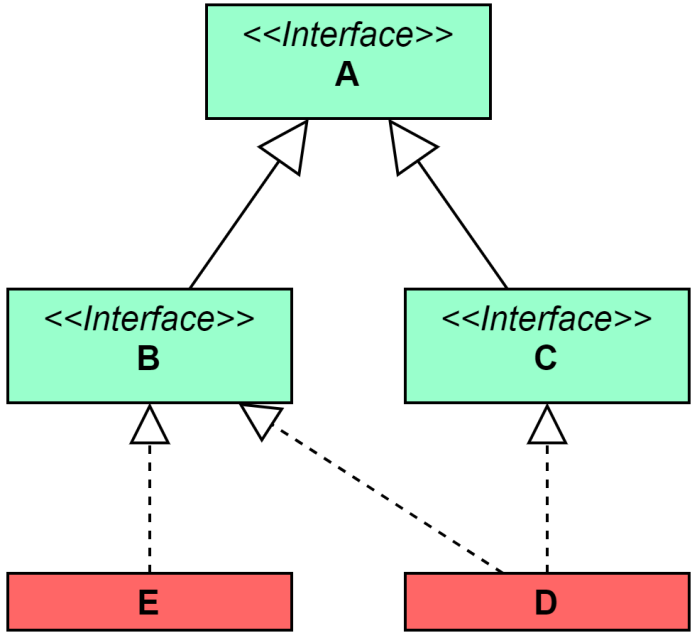
```
class MiClase extends SuPadre implements Interfaz1,Interfaz2
{// definición de la clase }
```

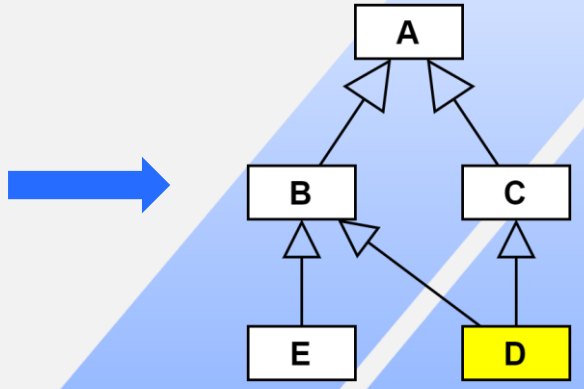
- Una interfaz **no puede implementar otra interfaz**, aunque sí extenderla (*extends*) ampliándola.
- Una interfaz **puede heredar de más de una interfaz antecesora**.
- Entre interfaces, está admitida tanto el uso de herencia simple como de herencia múltiple.
- La interface derivada incluye todas las constantes y declaraciones de métodos de la o las interfaces bases.

```
public interface InterfaceDerivada extends
    InterfaceBase1, InterfaceBase2, InterfaceBase3
    {// definición de la interfaz }
```

Herencia de interfaces

- Para implementar este esquema es necesario que las clases A, B y C de la figura sean interfaces, y que la clase D sea una clase que recibe la herencia múltiple:






```
public interface A{ }
public interface B extends A{ }
public interface C extends A{ }
public class D implements B,C{ }
public class E implements B{ }
```

Interfaces vs clases abstractas

- Una clase abstracta puede incluir métodos implementados y no implementados o abstractos y miembros constantes y otros no constantes.
  Las interfaces contienen una lista de métodos que pueden o no estar implementados e incluir la declaración de algunos miembros constantes.
- Una clase no puede heredar de dos clases abstractas, pero sí puede heredar de una clase abstracta e implementar una interface, o implementar dos o más interfaces.
  Las interfaces tienen una jerarquía propia, permitiéndose la herencia múltiple.
- Las interfaces proporcionan un mecanismo de encapsulación de los protocolos de los métodos sin forzar el uso de la herencia, asociando clases sin relación alguna.