DuocUC® INFORMÁTICA Y TELECOMUNICACIONES

EXPERIENCIA DE APRENDIZAJE 2
HERENCIA Y COLECCIONES

Sobrecarga, Polimorfismo y Herencia

DSY1102-Desarrollo Orientado a Objetos

CONTENIDO

DuocUC®

01

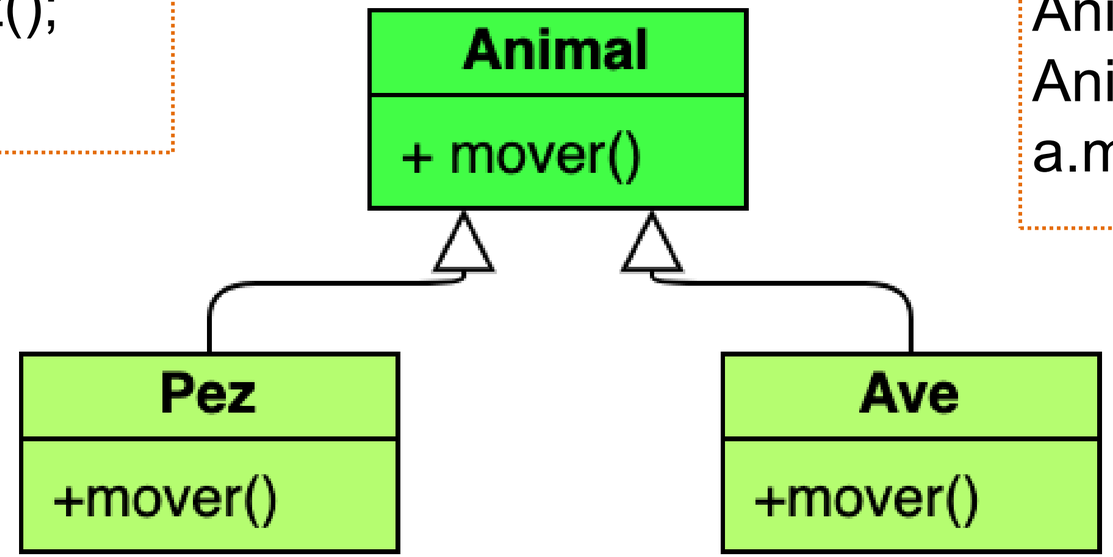
POLIMORFISMO

DuocUC®

Polimorfismo

Observa la siguiente imagen…

```
Pez p = new Pez();
p.mover();
```




```
Animal a = new Animal();
a.mover();
```

```
Ave av = new Ave();
av.mover();
```

¿Se mueven todos los animales de la misma forma?

¿Qué crees que es polimorfismo?

DuocUC®

Polimorfismo

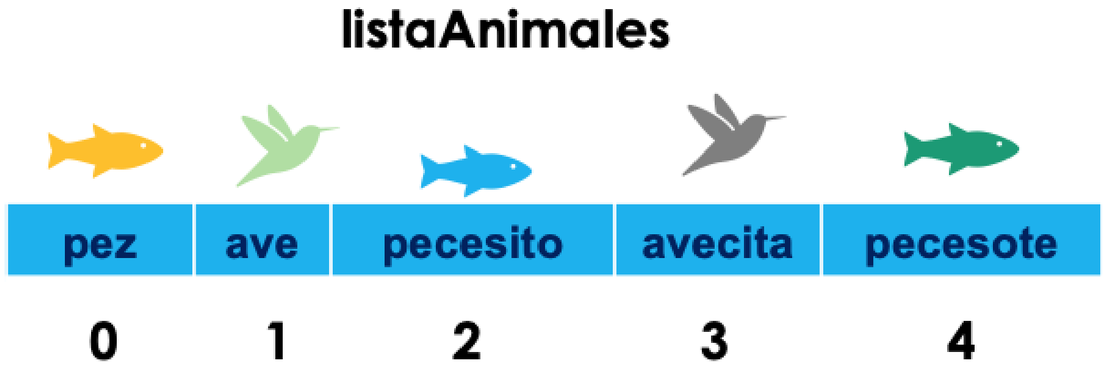
El **polimorfismo** nos permite “programar en forma general”, en vez de “programar en forma específica”. Nos permite escribir programas que procesen objetos que compartan la misma superclase en una jerarquía de clases, como si todos fueran objetos de la superclase

listaAnimales

pez	ave	pecesito	avecita	pecesote
0	1	2	3	4

Cada tipo especiífico de Animal responde a un mensaje mover de manera única; un Pez podría nadar 30 cm y un Ave podría volar 10 km. El programa invoca al mismo método mover de cada objeto Animal en forma genérica, pero cada objeto sabe cómo moverse. Confiar en que cada objeto sepa cómo “hacer lo correcto” en respuesta a la llamada al mismo método, es el concepto clave del **polimorfismo**. El método mover tiene **“muchas formas”**, de ahí nace el polimorfismo.

Polimorfismo



pez, pecesito y pecesote son objetos de la superclase Pez, por lo que al moverse ejecutarán el método definido en la clase Pez

ave y avecita son objetos de la superclase Ave, por lo que al moverse ejecutarán el método definido en la clase Ave

¿Por qué? → Polimorfismo

DuocUC®

CLASES Y MÉTDOS ABSTRACTOS

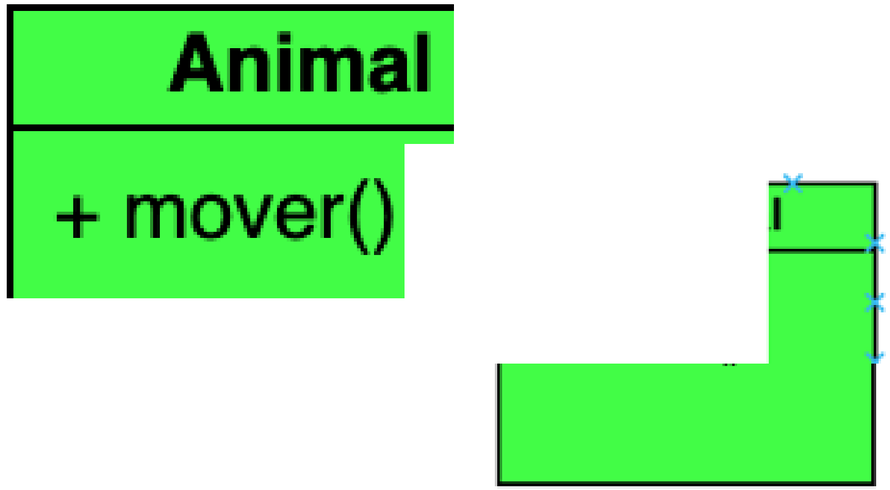
02

DuocUC®

Clases y métodos abstractos

Suponemos que al crear una clase, será para crear objetos de ella.
No obstante, en algunos casos es conveniente declarar clases para las cuales **nunca se crearán objetos de ella**. Estas son las **clases abstractas**. ¿Por qué?

Las clases abstractas están incompletas. Las subclases deben declarar las “piezas faltantes”.



DuocUC®

Clases y métodos abstractos

El propósito de una clase abstracta

Es proporcionar una superclase apropiada, a partir de la cual puedan heredar otras clases y, por ende, **compartir un diseño común**. Las subclases proporcionan los detalles específicos que hacen razonable la creación de instancias de objetos.

Clases y métodos abstractos

¿Cómo se declara una **clase abstracta**?
Se declara con la palabra clave **abstract**.

```
public abstract class Animal{
}
```

Esta clase, no se puede instanciar:

```
Animal a = new Animal();
```

Una clase abstracta declara los atributos y comportamientos comunes de las diversas clases en una jerarquía de clases. Creamos objetos de sus subclases que tendrán todas los atributos necesarios para ese objeto.

DuocUC®

Clases y métodos abstractos

Por lo general, una clase abstracta contiene uno o más métodos abstractos. Un **método abstracto** tiene la palabra clave **abstract** en su declaración y termina en **;**

```
public abstract void mover();
```

Una clase que contiene métodos abstractos debe declararse como clase abstracta

Los métodos abstractos **no tienen cuerpo**. Cada **subclase** de una superclase abstracta debe **proporcionar implementaciones a los métodos abstractos de la superclase**.

```
public abstract void mover();
```

superclase

```
public abstract void mover{
    //implementar cuerpo al método
}
```

subclase

DuocUC®

Clases y métodos abstractos

Tratar de instanciar un objeto de una clase abstracta es un error de compilación

Si no se implementan los métodos abstractos de la superclase en una subclase, se produce un error de compilación

DuocUC®

03

CLASES Y MÉTODOS FINAL

DuocUC®

Clases y métodos final

Las **variables** pueden declararse como **final** para indicar que **no pueden modificarse una vez que se inicializan**, es decir, representan valores **constantes**.

```
final IVA = 0.19;
```

Un **método** que se declara como **final** en una superclase **no puede sobrescribirse en una subclase**. Este método no puede cambiar, por lo cual todas las subclases utilizan la misma implementación del método.

```
public final void calcular(){
    //implementar cuerpo al método
}
```

superclase

```
public void calcular{
}
```

subclase

DuocUC®

Clases y métodos final

Una **clase** que se declara como **final no puede ser una superclase**, es decir, **una clase no puede extender a una clase final**. Todos los métodos en una clase final son implícitamente final.

```
public final class Animal{
}
```

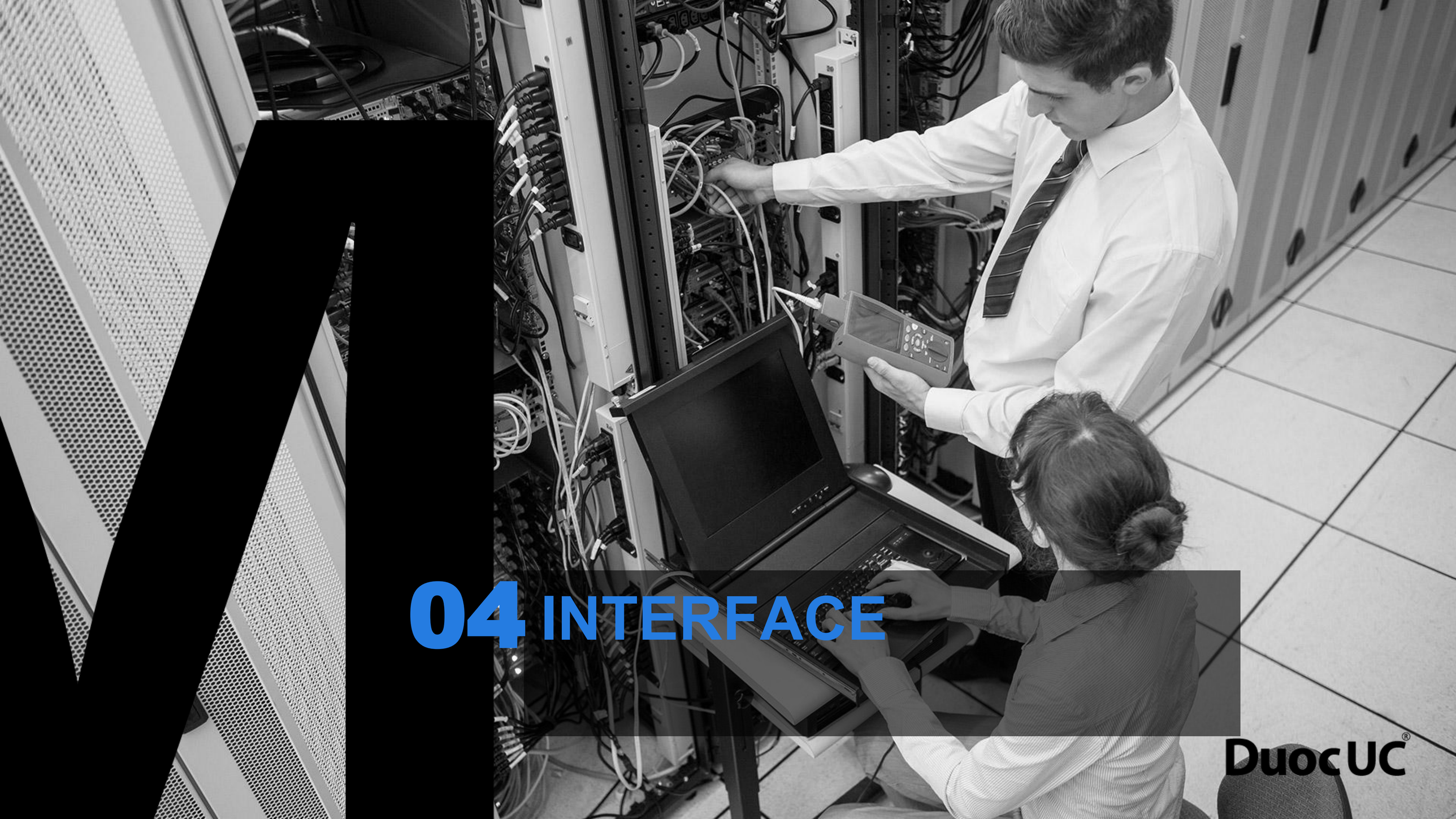
```
public class Pez extends Animal{
}
```

Clases y métodos abstractos

Tratar de declarar una subclase de una clase final es un error de compilación

Tratar de sobreescribir un método final es un error de compilación

En Java, la mayoría de clases no se declara como final. Esto permite la herencia y el polimorfismo: las características fundamentales de la programación orientada a objetos. Sin embargo, en algunos casos es importante declarar las clases como final, por razones de seguridad

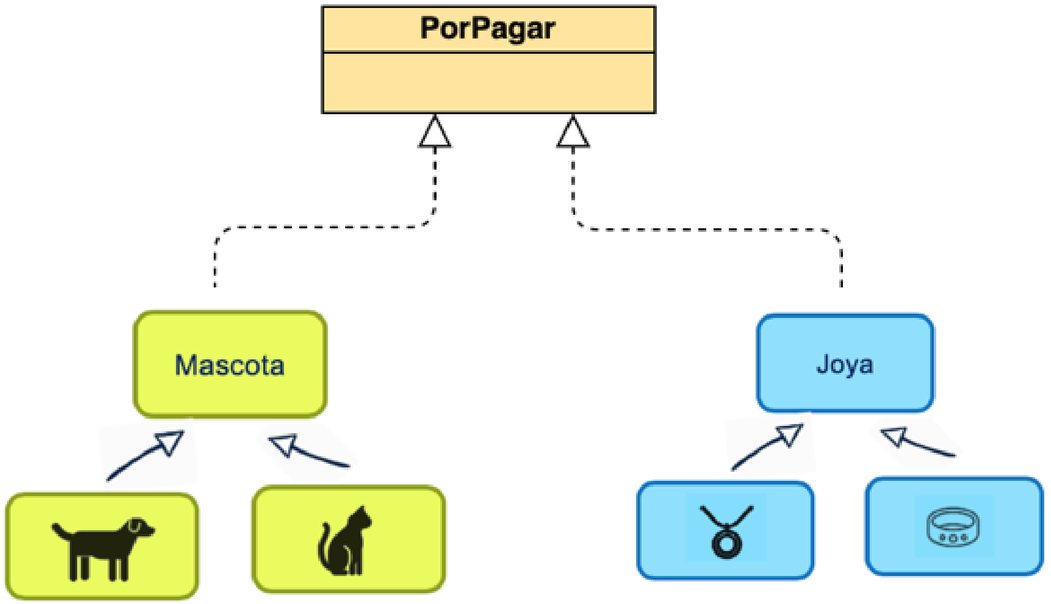


04 INTERFACE

DuocUC®

Interface

Las **interfaces** ofrecen una herramienta donde **clases no relacionadas implementen un conjunto de métodos comunes**.



La interfaz **especifica qué operaciones hay que implementar, pero no especifica cómo** deben realizarse las operaciones.

DuocUC®

Interface

La **declaración** de una **interfaz** empieza con la palabra clave **interface** y sólo **puede contener constantes y métodos abstract**.

```
public interface PorPagar{
    // constantes
    // métodos abstractos
}
```

Todos los métodos que se declaran en una interfaz son public abstract de manera implícita, y todos los campos son implícitamente public, static y final:

Interface

.

```
public interface PorPagar{
      IVA = 0.19
      void calcularPago();
}
```

is the same

```
public interface PorPagar{
      public static final IVA = 0.19
      public abstract void
calcularPago();
}
```

DuocUC®

Interface

Para **utilizar una interfaz**, **una clase debe especificar que implementa** (**implements**) **a esa interfaz y debe declarar cada uno de sus métodos** con la firma especificada (encabezado) en la declaración de la interfaz.

```
public class Animal implements PorPagar{
}
```

Implementar una interfaz es como **firmar un contrato** con el compilador que diga, “Declararé todos los métodos especificados por la interfaz”.

DuocUC®

Interface

Si no declaramos ningún miembro de una interfaz en una clase que implemente (implements) a esa interfaz, se produce un error de compilación

Una interfaz, se utiliza cuando clases no relacionadas, necesitan compartir métodos y constantes comunes. Esto permite que los objetos de clases no relacionadas se procesen en forma polimórfica; los objetos de clases que implementan la misma interfaz pueden responder a las mismas llamadas a métodos

La herencia y las interfaces son similares en cuanto a su implementación de la relación “es un”. Un objeto de una clase que implementa a una interfaz puede considerarse como un objeto del tipo de esa interfaz

Al declarar un método en una interfaz, seleccione un nombre para el método que describa su propósito en forma general, ya que el método podría implementarse por muchas clases no relacionadas

¿Qué hemos aprendido?

✓Utilizar polimorfismo.
✓Distinguir entre clases abstractas y concretas.
✓Implementar métodos abstractos y finales .
✓Implementar interfaces.
.

¿Qué te resultó difícil entender?

DuocUC®