Herencia Múltiple
Interfaces JAVA

Contenido

- Motivación
- Ventajas
- Problema del diamante
- Interfaces

Motivación

En el mundo real podemos encontrar muchas entidades que exhiben características y comportamientos heredados de más de un ancestro.

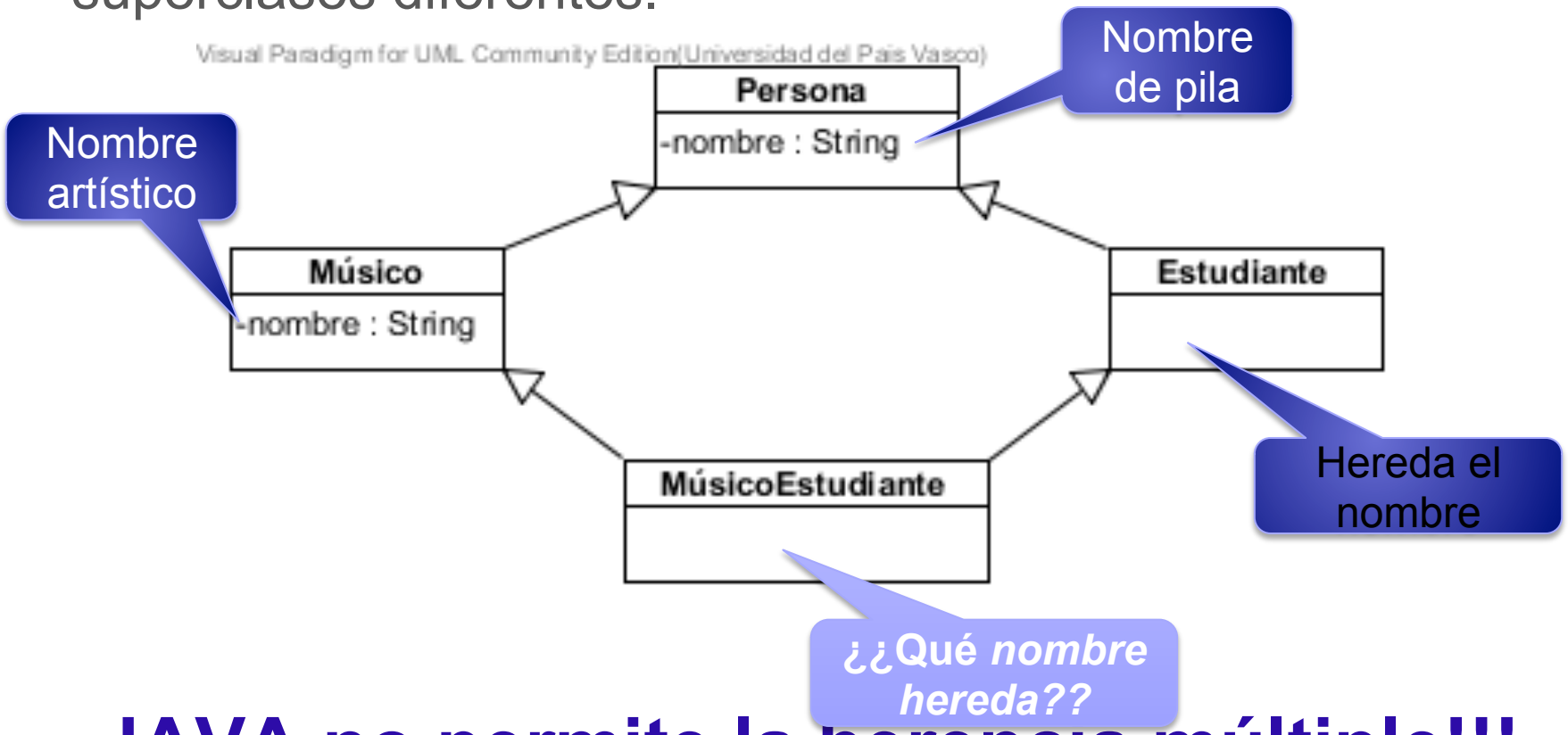
La **herencia múltiple** es un mecanismo de los lenguajes de programación orientada a objetos que permite que una clase puede heredar comportamientos y características de más de una superclase.

Ventajas

- Además de ser realista, incrementa las posibilidades de reutilización aportadas por la herencia simple.

Problema del Diamante

Surge cuando una clase hereda una característica o funcionalidad con el mismo nombre desde varias superclases diferentes.



JAVA no permite la herencia múltiple!!!

Interfaces

- Una interfaz JAVA es una clase abstracta pura (todos sus métodos son abstractos).
- Puede contener atributos, pero éstos son siempre *static* y *final*.
- Las interfaces JAVA se declaran igual que las clases pero utilizando la palabra reservada "`interfaz`" en vez de "`class`".

  public **`interfaz`** ***I****NombreInterfaz* { … }
- Los métodos declarados en una interfaz son siempre públicos y se implementarán en otras clases.

  public class nombreClase **`implements`** *INombreInterfaz* { … }
- Las interfaces pueden extender otras interfaces
- Una clase puede implementar más de una interfaz

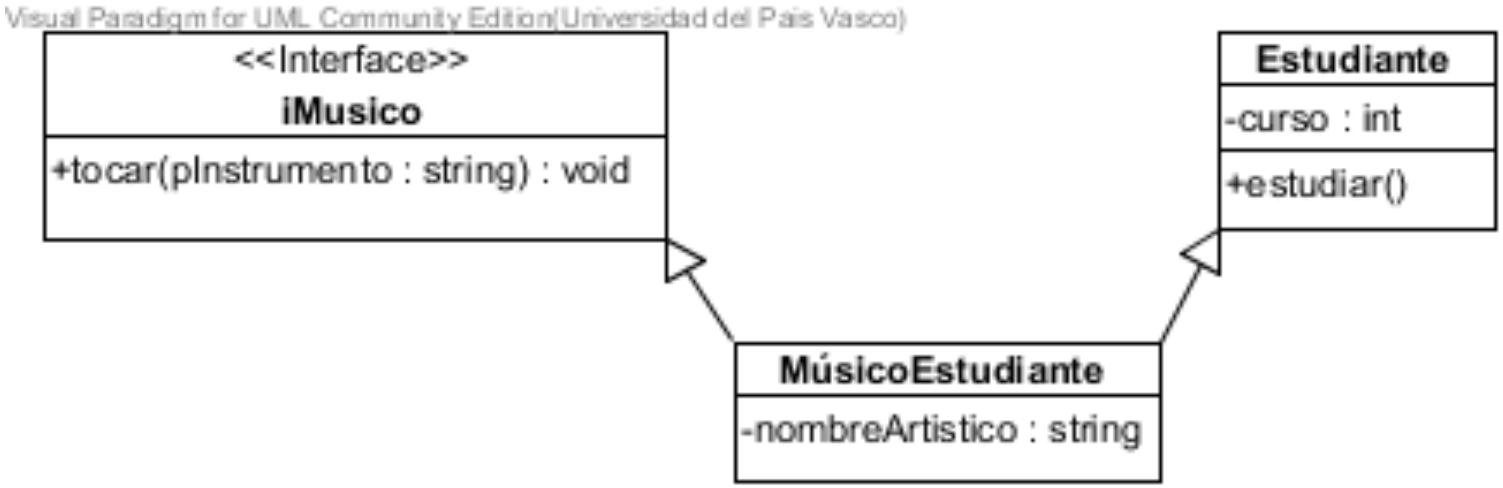
Interfaces

¿Para qué sirven las interfaces?

1. Para suplir la carencia de herencia múltiple en JAVA.
2. Para definir un comportamiento común a clases "*independientes*".
3. Para definir atributos semánticos de clases.

Interfaces

Suplen la carencia de herencia múltiple en JAVA




```
public class MusicoEstudiante extends Estudiante
                                    implements IMusico {
  string nombreArtistico;
  public void tocar() {
    …
      }
}
```

Interfaces

Permiten definir propiedades semánticas de clases

Es posible definir una interfaz vacía, SIN MÉTODOS NI ATRIBUTOS, para indicar propiedades semánticas de una clase.

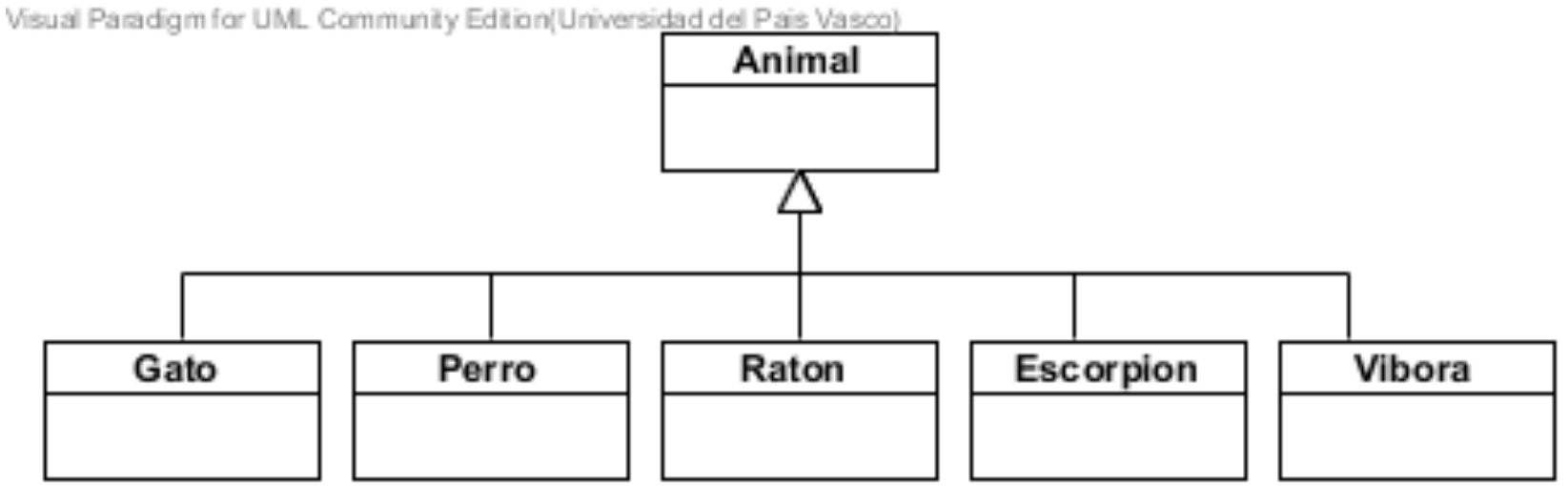
Ventajas:

Marcar clases haciendo que implementen una interface, permite distinguirlas del resto (como si la interface fuera un lápiz marcador).

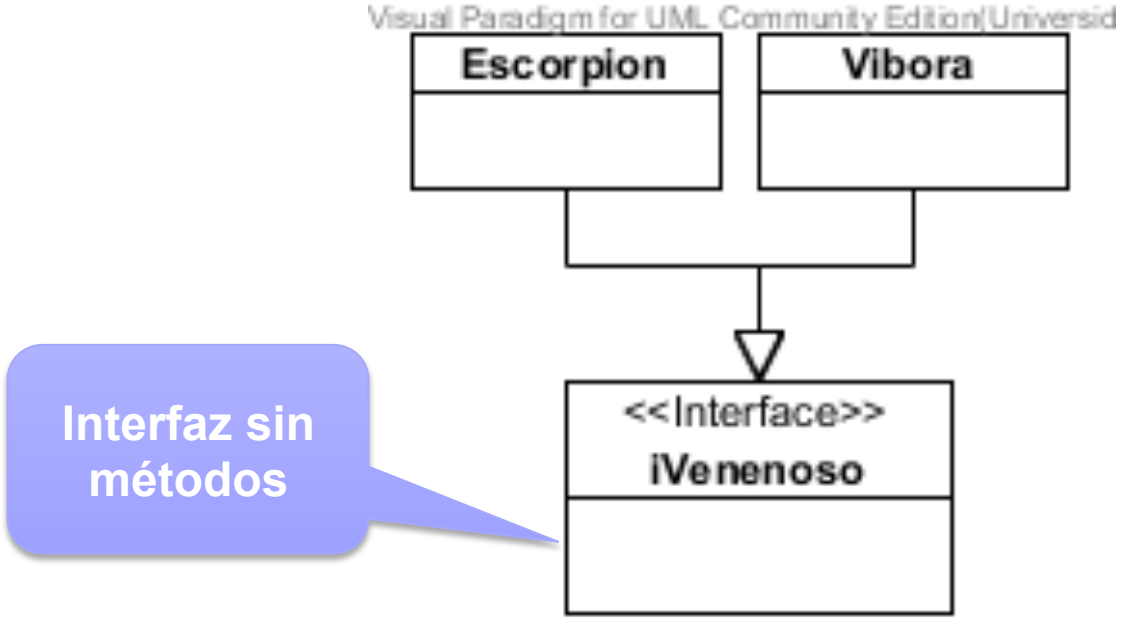
Interfaces

Ejemplo*:*

Se desea construir una aplicación para tratar con una jerarquía de animales. Esta jerarquía agrupa los animales en tres grades clases: carnívoros, herbívoros y omnívoros. Además, algunos anímales utilizan el veneno para cazar y la aplicación necesita distinguir los animales venenosos de los que no son.



Interfaces




```
public class Vibora implements IVenenoso {
        ...
}
```

```
public class Escorpion implements IVenenoso {
        ...
}
```

Interfaces

```
public class Prueba {

  public static void main(String[] args) {
    Animal[] animales = {
                new Vibora(), new Escorpion(),
                     new Vibora(), new Raton()};

  for (Animal a : animales)
   if (a instanceof IVenenoso)
      System.out.println(a + " es un animal Venenoso");

  }
}
```