Interfaces

CESSI #ArgentinaPrograma #YoProgramo

Leonardo Blautzik - Federico Gasior - Lucas Videla

Agosto -Diciembre de 2021

Interfaces

- La “interfaz pública” de una clase es un contrato entre el código del cliente y la clase que provee el servicio.

Interfaces

- La “interfaz pública” de una clase es un contrato entre el código del cliente y la clase que provee el servicio.
- Las clases concretas implementan cada método para cumplir con sus responsabilidades.

Interfaces

- La “interfaz pública” de una clase es un contrato entre el código del cliente y la clase que provee el servicio.
- Las clases concretas implementan cada método para cumplir con sus responsabilidades.
- Una clase abstracta puede diferir la implementación de los métodos declarándolos abstractos.

Interfaces

- Una Interfaz Java **solamente realiza la declaración del contrato** y no la implementación.

Interfaces

- Una Interfaz Java **solamente realiza la declaración del contrato** y no la implementación.
- Una clase concreta implementa una interfaz definiendo todos los métodos declarados en esa interfaz.

Interfaces

- Una Interfaz Java **solamente realiza la declaración del contrato** y no la implementación.
- Una clase concreta implementa una interfaz definiendo todos los métodos declarados en esa interfaz.
- Muchas clases (no relacionadas) pueden implementar la misma interfaz.

Interfaces

- Una Interfaz Java **solamente realiza la declaración del contrato** y no la implementación.
- Una clase concreta implementa una interfaz definiendo todos los métodos declarados en esa interfaz.
- Muchas clases (no relacionadas) pueden implementar la misma interfaz.
- Una clase puede implementar varias interfaces.

Interfaces

- Una Interfaz Java **solamente realiza la declaración del contrato** y no la implementación.
- Una clase concreta implementa una interfaz definiendo todos los métodos declarados en esa interfaz.
- Muchas clases (no relacionadas) pueden implementar la misma interfaz.
- Una clase puede implementar varias interfaces.
- Todos los métodos de una interfaz son public abstract por defecto.

Sintáxis de la declaración de una clase en Java:

```
<modifier> class <name> [extends <superclass>]
                        [implements <interface> [,<interface>]* ] {
    <declarations>*
}
```

Ejemplo de Interface

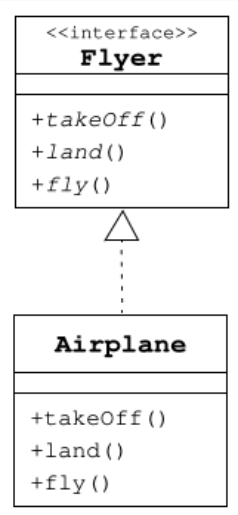

Figure 1: La Interface Flyer (Volador)

Ejemplo de Interface

```java
public interface Flyer {

  public void takeOff(); //despegar()
  public void land(); //aterrizar()
  public void fly(); //volar()

}
```

Ejemplo de Interface

```
public class Airplane implements Flyer {
    public void takeOff() {
       // accelerate until lift-off
       // raise landing gear
    }
    public void land() {
        // lower landing gear
        // deccelerate and lower flaps until touch-down
        // apply breaks
    }
    public void fly() {
        // keep those engines running
    }
}
```

Ejemplo de Interface

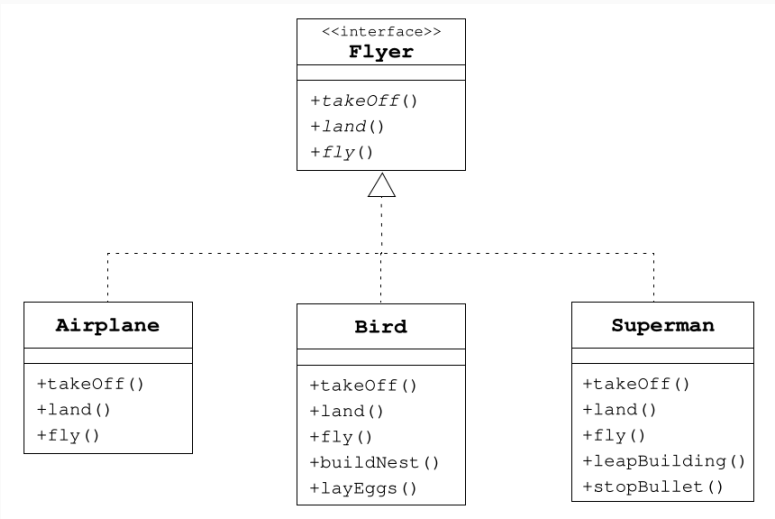

Figure 2: Varias clases implementando la interface Flyer

Ejemplo de Interface

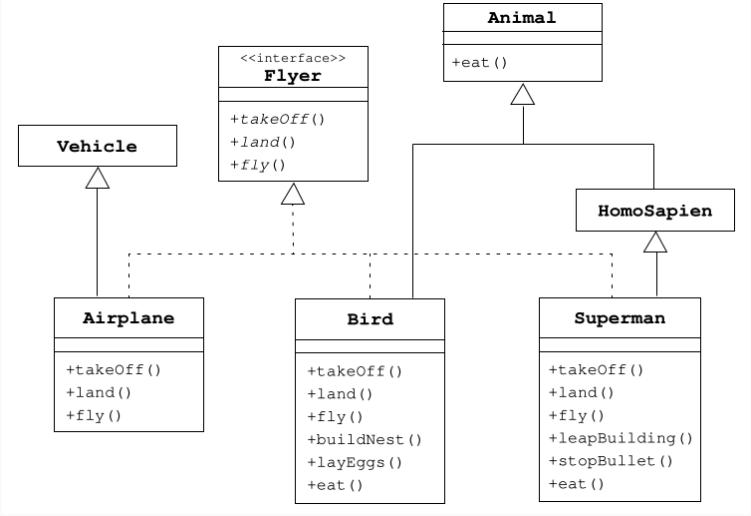

Figure 3: Varias clases combinando extend con implements

En la clase Bird (Pájaro) se deben implementar los métodos de Flyer y sobreescribir eat() de la superclase Animal.

```
public class Bird extends Animal implements Flyer {
    public void takeOff()   { /* take-off implementation */}
    public void land()      { /* landing implementation */}
    public void fly()       { /* land implementation */}
    public void buildNest() { /* nest building behavior */}
    public void layEggs()   { /* egg laying behavior */}
    public void eat()       { /* override eating behavior */}
}
```

Ejemplo de Interface

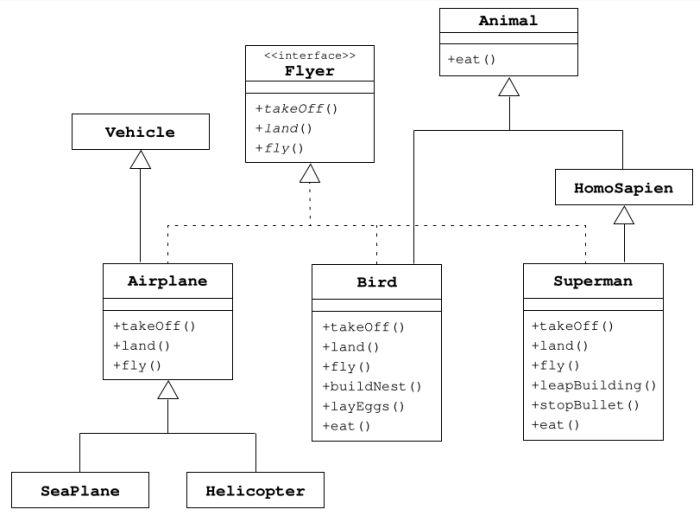

Figure 4: Agregamos SeaPlane y Helicopter

En un Aeropuerto...

```
public class Airport {
    public static void main(String[] args) {
        Airport metropolisAirport = new Airport();
        Helicopter copter = new Helicopter();
        SeaPlane sPlane = new SeaPlane();
        Flyer S = Superman.getSuperman(); // Superman is a Singleton
        metropolisAirport.givePermissionToLand(copter);
        metropolisAirport.givePermissionToLand(sPlane);
        metropolisAirport.givePermissionToLand(S);
    }
    private void givePermissionToLand(Flyer f) {
        f.land();
    }
}
```

Ejemplo de Interfaces Múltiples

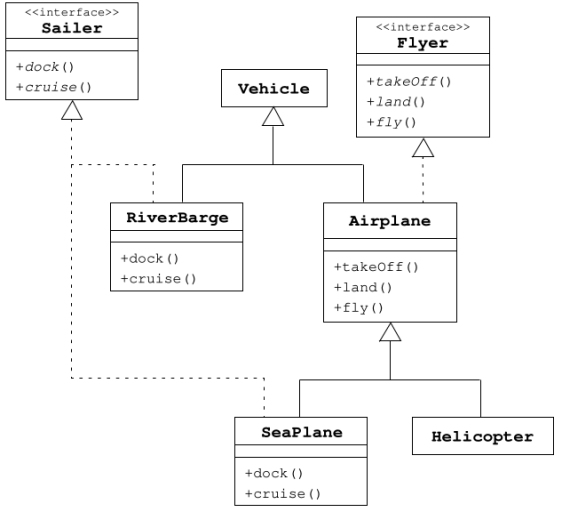

Figure 5: Interfaces Múltiples

Ahora implementamos un Puerto (Harbor)

```
public class Harbor {
    public static void main(String[] args) {
        Harbor bostonHarbor = new Harbor();
        RiverBarge barge = new RiverBarge();
        SeaPlane sPlane = new SeaPlane();
        bostonHarbor.givePermissionToDock(barge);
        bostonHarbor.givePermissionToDock(sPlane);
    }

    private void givePermissionToDock(Sailer s) {
        s.dock();
    }
}
```

Los Usos de las Interfaces

- Declarar los métodos que, se espera, implementen una o más clases.

Los Usos de las Interfaces

- Declarar los métodos que, se espera, implementen una o más clases.
- Determinar la interfaz de programación de un objeto sin revelar el cuerpo real de la clase.

Los Usos de las Interfaces

- Declarar los métodos que, se espera, implementen una o más clases.
- Determinar la interfaz de programación de un objeto sin revelar el cuerpo real de la clase.
- Capturar similitudes entre clases no relacionadas sin forzar una relación de clases.

Los Usos de las Interfaces

- Declarar los métodos que, se espera, implementen una o más clases.
- Determinar la interfaz de programación de un objeto sin revelar el cuerpo real de la clase.
- Capturar similitudes entre clases no relacionadas sin forzar una relación de clases.
- Simular herencia múltiple declarando una clase que implementa varias Interfaces.

Problema:

Implementar un módulo para una aplicación de geometría plana. Nos piden que modelemos la class Figura y además como extensión de Figura, Círculo y Cuadrado, los círculos están determinados por su centro (un Punto) y su radio. Los cuadrados por un Punto (su extremo inferior izquierdo) y el valor de sus lados. Además debemos implementar la class Segmento, cada segmento está determinado por sus dos extremos (Puntos). Las figuras, los puntos y los segmentos se deben poder desplazar a partir de un corrimiento en x y un corrimiento en y. Las figuras son comparables por su área y los segmentos por su longitud. También nos piden que, dado un segmento, se pueda saber si es paralelo al eje x, si es paralelo al eje y, y si dos segmentos son paralelos entre sí.

Problema

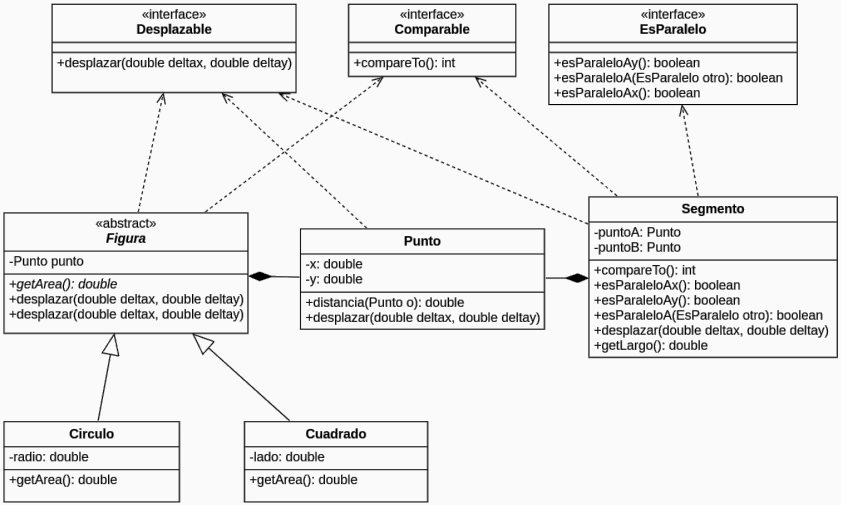

Figure 6: Diagrama UML Figuras y Segmentos

Veamos como resolverlo...

Vamos a eclipse y pasamos el diagrama UML a código java.