

I.2 Relaciones entre clases

Acceso a Datos

Alejandro Roig Aguilar

alejandro.roig@iesalvarofalomir.org

IES Álvaro Falomir

Curso 2023-2024

Típos de relaciones

Para diseñar clases, necesitamos explorar **cómo los objetos interactúan y se relacionan** entre ellos.

En este apartado estudiaremos las siguientes **relaciones**:

- **Asociación** (**agregación** y **composición**)
- **Herencia**
- **Polimorfismo**
- **Clases abstractas**
- **Interfaces**

Asociaciones

La **asociación** es un tipo de relación entre clases cuyos **objetos interactúan entre sí**. Estas relaciones presentan **navegabilidad** (**unidireccional o bidireccional**) y **multiplicidad** (**1-1, 1-N o N-N**).

```
public class Estudiante {  
    private Curso[] cursos;  
  
    public void addCurso(Curso c) {...}  
}
```

```
public class Curso {  
    private Estudiante[] estudiantes;  
  
    public void addEstudiante(Estudiante e) {...}  
}
```

La **agregación** representa una **asociación opcional**, donde los componentes pueden existir por sí mismos. Sería la relación entre un curso y un/a docente.

Si el objeto contenido **no puede existir por sí mismo**, se denomina **composición**. Por ejemplo, un docente y sus detalles de contacto.

Herencia

La **herencia** permite crear **nuevas clases** (**subclases**) a partir de **otras** (**superclases**), conservando los miembros no privados de la clase original e incorporando los suyos propios.

```
public class Persona {  
    protected String nombre;  
  
    public String mostrar() {  
        return nombre;  
    }  
}
```

```
public class Estudiante extends Persona {  
    @Override  
    public String mostrar() {  
        return "Estudiante" + super.mostrar();  
    }  
    public String saludar() {  
        return "Hola" + nombre();  
    }  
}
```

Polimorfismo

El **polimorfismo** alude a que una **variable de una superclase** puede hacer **referencia a un objeto de una subclase**.

```
// Asignación polimorfa
```

```
Persona estudiante = new Estudiante();
```

```
// Ejecución polimorfa
```

```
System.out.println(estudiante.mostrar());
```

```
// Casting
```

```
System.out.println(((Estudiante) estudiante).saludar());
```

Clases abstractas

Las **clases abstractas** no pueden ser instanciadas ya que su funcionalidad no está completamente definida.

Estas clases pueden tener **métodos abstractos**, los cuáles no son implementados, responsabilidad que recae en las subclases que la hereden.

```
public abstract class Persona {  
    protected String nombre;  
  
    public abstract String mostrar();  
}
```

```
public class Estudiante extends Persona {  
    @Override  
    public String mostrar() {  
        return "Estudiante: " + nombre;  
    }  
}
```

Interfaces

Una **interfaz** es una **clase abstracta** donde todos sus métodos son **abstractos**.

El objetivo es crear una **plantilla de comportamientos comunes** para las clases que la implementen.

El uso de interfaces permite simular la **herencia múltiple** que muchos lenguajes de programación no soportan.

```
public interface Persona {  
    public String mostrar();  
}
```

```
public class Estudiante implements Persona {  
    @Override  
    public String mostrar() {  
        return "Estudiante: " + nombre;  
    }  
}
```

Interfaz funcional

Una **interfaz** es **funcional** si **solo define un método**.

Por ejemplo, la interfaz **Comparable** define un método **compareTo** para comparar un objeto propio (**this**) con otro de la misma clase (**o**).

int compareTo(T **o**)

- Si devuelve un entero **negativo**, el objeto **this** es **menor que o**.
- Si devuelve un **cero**, el objeto **this** es **igual a o**.
- Si devuelve un entero **positivo**, el objeto **this** es **mayor que o**.

```
public class Estudiante implements Comparable<Estudiante> {  
    @Override  
    public int compareTo(Persona o) {  
        return o.edad - this.edad; // Ordenación descendente  
        // return this.edad - o.edad; sería ordenación ascendente  
    }  
}
```