**Tutorial Interactivo: Agregación de Clases en Java**

En este tutorial, aprenderás el concepto de **agregación de clases** en la programación orientada a objetos usando Java. A través de una serie de pasos, ejemplos y ejercicios prácticos, aplicarás este concepto para entender cómo las clases pueden interactuar entre sí.

**1. Concepto de Agregación**

**Definición:**

La **agregación** es un tipo de relación entre clases donde una clase contiene una referencia a otra clase. En términos simples, es cuando una clase es parte de otra, pero ambos objetos pueden existir de manera independiente.

En la agregación, la clase que contiene a otras clases se denomina **"contenedor"** o **"compuesta"**, mientras que las clases contenidas se denominan **"componentes"** o **"agregadas"**.

**Ejemplo:**

Imagina que tienes una clase **Universidad** y otra clase **Profesor**. La relación entre ambas es de agregación, porque una universidad tiene profesores, pero un profesor puede existir fuera de una universidad.

java

Copiar código

class Profesor {

String nombre;

Profesor(String nombre) {

this.nombre = nombre;

}

}

class Universidad {

String nombre;

Profesor[] profesores; // Relación de agregación

Universidad(String nombre, Profesor[] profesores) {

this.nombre = nombre;

this.profesores = profesores;

}

void mostrarProfesores() {

System.out.println("Profesores en " + this.nombre + ":");

for (Profesor profesor : profesores) {

System.out.println(profesor.nombre);

}

}

}

public class Main {

public static void main(String[] args) {

Profesor prof1 = new Profesor("Dr. Lopez");

Profesor prof2 = new Profesor("Dra. Martinez");

Profesor[] profesores = {prof1, prof2};

Universidad uni = new Universidad("Universidad XYZ", profesores);

uni.mostrarProfesores(); // Imprime la lista de profesores

}

}

En este ejemplo:

* La clase **Universidad** contiene una lista de objetos **Profesor**, pero los objetos Profesor pueden existir sin la clase Universidad.

**Tarea 1:**

Crea una clase **Curso** que contenga una lista de objetos de tipo **Estudiante**. Cada estudiante debe tener un nombre. En la clase Curso, crea un método que imprima los nombres de los estudiantes.

**Pregunta 1:**

¿Qué significa que la relación entre dos clases sea de **agregación**? ¿Cómo se diferencia de una relación de **composición**?

**2. Diferencia entre Agregación y Composición**

**Definición de Composición:**

La **composición** es otro tipo de relación entre clases, donde una clase está compuesta por otras, pero los objetos contenidos **no pueden existir de manera independiente**. Si la clase compuesta es destruida, las clases que contiene también lo son.

En comparación:

* **Agregación**: Los objetos pueden existir independientemente de la clase que los contiene.
* **Composición**: Los objetos no pueden existir de forma independiente. Si el objeto principal se destruye, los objetos que contiene también lo son.

**Ejemplo de Composición:**

java

Copiar código

class Motor {

Motor() {

System.out.println("Motor creado");

}

}

class Coche {

Motor motor;

Coche() {

motor = new Motor(); // El coche "posee" un motor

System.out.println("Coche creado");

}

}

Aquí, el **motor** es parte integral del **coche**, y si el coche es destruido, el motor también deja de existir. No puede haber un **motor** sin un **coche**.

**Tarea 2:**

Crea una clase **Casa** que tenga un objeto de tipo **Puerta**. Haz que la relación sea de composición, de tal manera que cuando un objeto Casa sea destruido, la puerta también lo sea.

**Pregunta 2:**

¿Qué tipo de relación hay entre las clases en la **composición** y por qué no se pueden usar independientemente?

**3. Implementación de Agregación**

Veamos un ejemplo más profundo de cómo implementar una relación de agregación. Supongamos que estás creando un sistema escolar en el que hay **Escuelas** que tienen **Cursos**, y cada **Curso** tiene **Profesores**.

java

Copiar código

class Profesor {

String nombre;

Profesor(String nombre) {

this.nombre = nombre;

}

}

class Curso {

String nombre;

Profesor profesor; // Relación de agregación

Curso(String nombre, Profesor profesor) {

this.nombre = nombre;

this.profesor = profesor;

}

void mostrarDetalles() {

System.out.println("Curso: " + this.nombre + ", Profesor: " + profesor.nombre);

}

}

class Escuela {

String nombre;

Curso[] cursos;

Escuela(String nombre, Curso[] cursos) {

this.nombre = nombre;

this.cursos = cursos;

}

void mostrarCursos() {

System.out.println("Cursos en " + this.nombre + ":");

for (Curso curso : cursos) {

curso.mostrarDetalles();

}

}

}

public class Main {

public static void main(String[] args) {

Profesor prof1 = new Profesor("Dr. Perez");

Profesor prof2 = new Profesor("Dra. Garcia");

Curso curso1 = new Curso("Matemáticas", prof1);

Curso curso2 = new Curso("Física", prof2);

Curso[] cursos = {curso1, curso2};

Escuela escuela = new Escuela("Escuela ABC", cursos);

escuela.mostrarCursos(); // Muestra los cursos y sus profesores

}

}

En este ejemplo:

* La clase **Escuela** tiene una relación de agregación con **Curso**.
* La clase **Curso** tiene una relación de agregación con **Profesor**.

**Tarea 3:**

Implementa un sistema en el que una clase **Biblioteca** tenga una relación de agregación con varias **Libros**, donde cada libro tenga un título y un autor. Crea un método en la clase Biblioteca para mostrar la lista de libros.

**Pregunta 3:**

¿Cómo puedes reconocer si una relación entre clases es de agregación en lugar de composición, solo viendo el código?

**4. Relaciones de Cardinalidad en la Agregación**

**Definición:**

En una relación de agregación, se puede especificar cuántas instancias de una clase pueden estar relacionadas con otra. Esto se conoce como **cardinalidad**. Por ejemplo:

* **Uno a uno**: Un curso tiene un profesor.
* **Uno a muchos**: Una escuela tiene muchos cursos.
* **Muchos a muchos**: Muchos cursos pueden tener muchos estudiantes.

**Ejemplo:**

java

Copiar código

class Estudiante {

String nombre;

Estudiante(String nombre) {

this.nombre = nombre;

}

}

class Curso {

String nombre;

Estudiante[] estudiantes; // Relación de agregación (muchos estudiantes)

Curso(String nombre, Estudiante[] estudiantes) {

this.nombre = nombre;

this.estudiantes = estudiantes;

}

void mostrarEstudiantes() {

System.out.println("Estudiantes en el curso " + this.nombre + ":");

for (Estudiante estudiante : estudiantes) {

System.out.println(estudiante.nombre);

}

}

}

En este caso, un curso tiene **muchos estudiantes**, lo que representa una relación de **uno a muchos**.

**Tarea 4:**

Implementa una relación **uno a muchos** donde una clase **Autor** escriba varios **Libros**. Cada libro debe tener un título y estar asociado a un autor.

**Pregunta 4:**

¿Cómo determinarías si una relación entre clases es de uno a uno, uno a muchos o muchos a muchos? ¿Cómo afecta esto la implementación en código?

**5. Ventajas de la Agregación**

1. **Reutilización de código**: La agregación permite reutilizar clases ya existentes, reduciendo la duplicación de código.
2. **Independencia de objetos**: Los objetos agregados pueden existir independientemente del objeto que los contiene, lo que mejora la modularidad del código.
3. **Flexibilidad**: Puedes modificar las clases agregadas sin afectar las clases que las contienen.

**Tarea 5:**

Expande el sistema de la tarea anterior para que un **Autor** pueda ser parte de una **Editorial**, manteniendo la relación de agregación entre todas las clases involucradas.

**Pregunta 5:**

¿Cuáles son las principales ventajas de utilizar agregación en lugar de composición? ¿Cuándo optarías por una sobre la otra?

**Ejercicio Final**

**Objetivo:**

Crea un sistema completo que simule una **Universidad** con las siguientes relaciones:

1. Una **Universidad** contiene muchos **Departamentos**.
2. Cada **Departamento** tiene muchos **Profesores** y muchos **Estudiantes**.
3. Cada **Profesor** puede enseñar varios **Cursos**.
4. Cada **Estudiante** puede estar inscrito en varios **Cursos**.

**Pasos:**

1. Define las clases **Universidad**, **Departamento**, **Profesor**, **Estudiante** y **Curso**.
2. Crea relaciones de agregación entre estas clases.
3. Implementa métodos en cada