Jerarquía de clases

El paradigma de programación orientada a objetos se centra en el vocabulario original del problema. Utiliza métodos de análisis que examinan los requerimientos del sistema desde la perspectiva de objetos cooperantes extraídos directamente del vocabulario original del problema.

Un programa orientado a objetos que sirva para gestionar un taller mecánico utilizará objetos de las clases *Coche*, *Reparacion*, *Mecanico*, *Cliente*, *Pieza*, etc., nombres que extraemos del análisis del funcionamiento del propio taller mecánico.

Los objetos que forman parte de un proceso en ejecución, no actúan de manera aislada, sino que colaboran con otros objetos. Esto da lugar a una jerarquía de clases y objetos que se forma a partir de interrelaciones entre clases que pueden ser de distinto tipo:

* **Composición** (la propiedad de una clase es un objeto de otra clase).

Basado en los conceptos de compuesto y componente (un motor es un componente de un coche).

* + Composición débil (agregación): al eliminar un compuesto no se elimina el componente.
  + Composición fuerte: al eliminar un compuesto se elimina también el componente.
* **Generalización / Especialización** (que se relaciona con la herencia).
  + Se cumple la regla “es un”.
  + Se va desde lo más abstracto (*Figura*) hasta lo más concreto (*Circulo*, *Cuadrado*, *Rombo*).

## Composición

La composición es un tipo de interrelación entre clases donde una propiedad de una clase es un objeto de otra clase. Está muy estrechamente relacionado con las palabras componente y compuesto. Un objeto *Coche* podría tener las propiedades *matricula*, *marca*, *modelo* y *motor*, siendo *motor* un objeto con sus propiedades; *tipoCombustible*, *caballos*, etc. Diríamos que un *motor* es un componente del *Coche* que es el compuesto.

Estas relaciones tienen cardinalidad para poder expresar el número de ocurrencias de un objeto contenido dentro de un objeto contenedor.

1 / 0..\* (Un objeto *Fiesta* contiene muchos objetos *Invitado*)

1 / 1 (Un objeto *Nomina* contiene un objeto *Empleado*)

1 / 0..1 (Un objeto *Piso* tiene cero o un objeto *Frigorifico*)

### Composición débil o agregación

Al eliminar el compuesto (Coche) no se elimina el componente (Motor).

**public** **class** Motor {

**private** **int** caballos;

**private** **int** tipoCombustible; // 0=gasolina, 1=diesel

**public** Motor(**int** caballos, **int** tipoCombustible) {

**this**.caballos = caballos;

**this**.tipoCombustible = tipoCombustible;

}

**public** **int** getCaballos() {

**return** caballos;

}

**public** **int** getTipoCombustible() {

**return** tipoCombustible;

}

**public** String estado() {

**return** "Caballos: " + caballos + ", Combustible: " + tipoCombustible;

}

}

**public** **class** Coche {

**private** String matricula;

**private** String marca;

**private** String modelo;

**private** Motor motor;

**public** Coche(String matricula, String marca, String modelo,

Motor motor) {

**this**.matricula = matricula;

**this**.marca = marca;

**this**.modelo = modelo;

**this**.motor = motor;

}

**public** String estado() {

**return** "Matricula=" + matricula + ", Marca=" + marca +

", Modelo=" + modelo + ", Motor: (" + motor.estado() + ")";

}

}

**public** **class** Principal {

**public** **static** **void** main(String[] args) {

Motor miMotor = **new** Motor(100, 1);

Coche miCoche = **new** Coche("5568FJK","Ford","Fiesta", miMotor);

System.***out***.println(miCoche.estado());

miCoche = **null**;

System.***out***.println("El objeto motor sigue activo con caballos "

+ miMotor.getCaballos());

}

}

### Composición fuerte

Al eliminar el compuesto (Coche) se elimina el componente (Motor).

**public** **class** Coche {

**private** String matricula;

**private** String marca;

**private** String modelo;

**private** Motor motor;

**public** Coche(String matricula, String marca, String modelo,

**int** caballos, **int** tipoCombustible) {

**this**.matricula = matricula;

**this**.marca = marca;

**this**.modelo = modelo;

**this**.motor = **new** Motor(caballos, tipoCombustible);

}

**public** String estado() {

**return** "Matricula=" + matricula + ", Marca=" + marca +

", Modelo=" + modelo + ", Caballos=" +

motor.getCaballos() +

", Tipo Combustible=" + motor.getTipoCombustible();

}

}

**public** **class** Principal {

**public** **static** **void** main(String[] args) {

Coche miCoche = **new** Coche("5568FJK","Ford","Fiesta", 100, 1);

System.***out***.println(miCoche.estado());

}

}

La clase Motor no cambia.

## Generalización / Especialización (herencia)

Basada en una interrelación entre clases donde una clase derivada o hija hereda las propiedades y métodos de otra clase base o madre. Profundizaremos en otro documento.