Programación orientada a objetos

Para entender objetos no queda otra opción que empezar por el principio, en este caso es entender que trabajar con objetos es trabajar en un **paradigma**.
Un **paradigma** es un marco de referencia que impone reglas sobre cómo se deben hacer las cosas, indica qué es válido dentro del paradigma y qué está fuera de sus límites. La primera regla del paradigma trata sobre qué se intenta obtener como resultado del desarrollo de sistemas utilizando objetos:

"Un sistema hecho con objetos es un modelo computacional de una porción de la realidad (la porción que queremos sistematizar), denominada dominio"

Durante el proceso de modelización de la realidad que hacemos, no debemos preocuparnos por cuestiones que no pertenecen a ella como la performance, la persistencia[1] u otros problemas o cuestiones de otra realidad, como la computacional.

Tenemos que ocuparnos de "modelar bien" la realidad de nuestro problema. "Entonces, si no debemos ocuparnos de la performance ni la persistencia, ¿de qué debemos ocuparnos cuando modelamos con objetos?". Debemos ocuparnos de las "**responsabilidades**"de estos objetos, de **qué** hacen. Cada vez que se agrega una responsabilidad a un objeto debemos estar seguros que realmente corresponda a ese objeto.

Conceptos básicos

Programa: Un programa en el paradigma de objetos es un conjunto de objetos que colaboran entre sí enviando mensajes.

Objeto: Es la representación de un ente del dominio del problema. Tienen un determinado estado interno y responsabilidades. Los objetos responden a los pedidos interactuando con los otros objetos que conoce, y así se va armando una aplicación. A distintos observadores les van a interesar distintas características y formas de interacción con los mismos objetos.

Clase: representa un modelo abstracto o idea del dominio de problema. Definen el comportamiento y la forma de un conjunto de objetos (sus instancias). Todo objeto es instancia de alguna clase.

Mensaje: Es la especificación sobre qué puede hacer un objeto. Se debería poder decir qué representa un objeto a partir de los mensajes que sabe responder. Una responsabilidad es invocada a través de un mensaje.

Colaboración: Hecho por el cual dos objetos se comunican por medio de un mensaje.
En toda colaboración existe

- Un emisor del mensaje
- Un receptor del mensaje
- Un conjunto de objetos que forman parte del mensaje
- Una respuesta

Características de las colaboraciones:

- Dirigida (Es decir siempre existe un receptor determinado),
- Son sincrónicas, el emisor no continúa hasta que obtenga la respuesta.

[1] Se denomina persistencia al proceso por el cual se graban y obtienen objetos de medios persistentes (discos, cintas, etc.)

- El receptor desconoce al emisor y su reacción será siempre la misma no importa quién le envía el mensaje.
- Siempre hay respuesta

Método: sección de código que se evalúa cuando un objeto recibe un mensaje. Se asocia al mensaje mediante su nombre. Desde el punto de vista del comportamiento, un método implementa el cómo .

Pilares de la Programación Orientada a Objetos

1. **Encapsulamiento**: ocultamiento del estado interno de un objeto. Permite separar qué hacen los objetos (responsabilidad) de cómo lo hacen (implementación).

2. **Herencia**: jerarquía de clases. La herencia simple es cuando una clase tiene una única clase padre. Es una relación estática entre clases.

3. **Abstracción:** expresa las características esenciales de un objeto, las cuales distinguen al objeto de los demás.

4. **Polimorfismo**: permite al objeto emisor del mensaje despreocuparse de quién es exactamente su colaborador, sólo le interesa que sea responsable de llevar adelante la tarea que le encomienda a través del "mensaje". quién

Principios básicos de la Programación Orientada a Objetos (SOLID):

SOLID es un acrónimo que representa cinco principios básicos de la programación orientada a objetos Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de extender y ampliar con el tiempo.

S: Principio de responsabilidad única (Single responsibility principle):
Una clase debe tener una única razón para cambiar. Es decir, una clase no debe agrupar varias responsabilidades.

O: Principio de abierto/cerrado (Open/closed principle)
Las entidades de software (clases, módulos, métodos) deben estar abiertas para su extensión, pero cerradas para su modificación.

L: Principio de sustitución de Liskov (Liskov substitution principle)
Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

I: Principio de segregación de la interfaz (Interface segregation principle)
Muchas interfaces cliente específicas son mejores que una interfaz de propósito general.

D: Principio de inversión de la dependencia (Dependency inversion principle)
Se debe depender de abstracciones (es decir, qué quiero representar) y no depender de implementaciones (es decir, cómo lo voy a representar).

Conclusión:

Desarrollar con objetos es modelar la realidad utilizando objetos que deben colaborar entre sí enviándose mensajes. Estos objetos sólo se preocupan de que sus colaboradores sepan qué hacer (responsabilidad), no cómo

lo hacen, y no tienen problemas sobre quiénes son sus colaboradores (polimorfismo) siempre y cuando respondan sus mensajes.