Programação orientada a objetos

Para entender objetos, precisamos começar pelo princípio. Neste caso, isso significa compreender que trabalhar com objetos é trabalhar com um **paradigma**.
Um **paradigma** é um quadro de referência que impõe regras sobre como as coisas devem ser feitas e indica o que é válido dentro do paradigma e o que está fora de seus limites. A primeira regra do paradigma trata do que queremos obter como resultado do desenvolvimento de sistemas usando objetos:

"Um sistema feito com objetos é um modelo computacional de uma porção da realidade (a porção que queremos sistematizar) denominada 'domínio'."

Durante o processo de modelagem de uma realidade, não devemos nos preocupar por questões que não pertençam a ela, como o desempenho, a persistência[1] ou outros problemas ou questões de outra realidade, como a computacional.

Temos que nos dedicar a "modelar corretamente" a realidade do nosso problema. "Então, se não temos que nos preocupar com o desempenho e a persistência, a que devemos nos dedicar ao modelar com objetos?". Debemos trabalhar com as "**responsabilidades**"destes objetos, o **que** fazem. Cada vez que uma responsabilidade é acrescentada a um objeto, devemos ter certeza de que ela realmente se aplique a esse objeto.

Conceitos básicos

Programa: No paradigma de objetos, um programa é um conjunto de objetos que colaboram entre si por meio do envio de mensagens.

Objeto: Trata-se da representação de uma entidade do domínio do problema. Os objetos têm um determinado estado interno e responsabilidades. Eles respondem aos pedidos interagindo com os outros objetos que conhecem e, assim, um aplicativo vai sendo criado. Diferentes observadores se interessarão por diferentes características e formas de interação com os mesmos objetos.

Classe:Representa um modelo abstrato ou ideia do domínio de um problema. Uma classe define o comportamento e a forma de um conjunto de objetos (suas instâncias). Todo objeto é uma instância de alguma classe.

Mensagem: É a especificação do que um objeto pode fazer. Com base nas mensagens que um objeto sabe responder, deveria ser possível dizer o que ele representa. Uma responsabilidade é invocada por meio de uma mensagem.

Colaboração: Quando dois objetos comunicam entre si por meio de uma mensagem.
Toda colaboração envolve o seguinte:

- Um emissor da mensagem
- Um receptor da mensagem
- Um conjunto de objetos que fazem parte da mensagem
- Uma resposta

Características das colaborações:

---

[1] "Persistência" se refere ao processo usado para gravar e obter objetos de formatos persistentes (discos, fitas, etc.)

- São direcionadas (ou seja, sempre existe um receptor determinado).
- São síncronas e o emissor não continua até obter uma resposta.
- O receptor desconhece o emissor e sua reação sempre será a mesma, não importa quem enviar a mensagem.
- Sempre há uma resposta.

Método: A seção de código que é avaliada quando um objeto recebe uma mensagem. A mensagem é associada com base no seu nome. Do ponto de vista de comportamento, um método implementa o "como".

Pilares da Programação Orientada a Objetos

1. **Encapsulamento**: Ocultamento do estado interno de um objeto. Permite separar o que os objetos fazem (responsabilidade) da maneira como é feito (implementação).

2. **Herança**: Hierarquia de classes. A herança simples é quando uma classe tem uma única "classe pai". É uma relação estática entre classes.

3. **Abstração:** Expressa as características essenciais de um objeto, que o distinguem dos outros objetos.

4. **Polimorfismo**: Permite que o objeto emissor de uma mensagem se despreocupe com quem exatamente é seu colaborador. Para esse objeto, só interessa que ele seja responsável por executar a tarefa encomendada pela "mensagem".

Fundamentos da Programação Orientada a Objetos (SOLID):

SOLID é um acrônimo que representa cinco princípios básicos da programação orientada a objetos. Quando estes princípios são aplicados em conjunto, é mais provável que um desenvolvedor crie um sistema que seja fácil de se estender e ser ampliado com o tempo.

S: Princípio da responsabilidade única (Single Responsibility Principle):
Uma classe deve ter somente uma razão para mudar. Em outras palavras, uma classe não deve juntar várias responsabilidades.

O: Princípio do aberto/fechado (Open/Closed Principle)
As entidades de software (classes, módulos, métodos) devem estar abertas para sua extensão, mas fechadas para sua modificação.

L: Princípio de substituição de Liskov (Liskov Substitution Principle)
Cada classe herdada (derivada) de outra pode ser usada como sua classe pai, sem necessidade de conhecer as diferenças entre elas.

I: Princípio de segregação da interface (Interface Segregation Principle)
Muitas interfaces de cliente específicas são melhores que uma interface de propósito geral.

D: Princípio de inversão da dependência (Dependency Inversion Principle)
Deve-se depender de abstrações (ou seja, do que eu quero representar), e não de implementações (ou seja, de como vou representar isso).

Conclusão:

Programar com objetos significa modelar a realidade utilizando objetos que devem colaborar entre si por meio do envio de mensagens. Para esses objetos, a única coisa que interessa é que seus colaboradores saibam o que fazer (responsabilidade), independentemente de como será feito e de quem são seus colaboradores (polimorfismo), contanto que suas mensagens sejam respondidas.