Java Júnior

Conceitos Básicos sobre Orientação a Objetos

O ser humano tenta organizar um problema de forma a conseguir focar em uma única parte por vez, assim, torna-se mais fácil compreender uma informação se essa estiver dividida em várias partes menores em vez de uma única parte.

A Orientação a Objetos é um paradigma de programação, isto é, um modelo, um padrão para projetar software, cuja essência consiste em desmembrar um problema complexo em uma série de problemas menores, mais simples e mais fáceis de entender.

Principais Conceitos acerca da Orientação a Objetos

- **Objeto**
  - É a unidade da qual queremos representar informações no sistema. Na prática, todo objeto é instância de uma classe.

- **Classe**
  - Consiste em um conjunto de objetos com características comuns, sendo estas, os atributos ou propriedades - variáveis - e os métodos ou operações - funções.

- **Variáveis**
  - Um conjunto de variáveis forma o estado de um determinado objeto.
  - Existem apenas dois tipos de variáveis que podem ser definidas em uma classe: as variáveis de instância e as variáveis estáticas.

- **Métodos**
  - As funções definidas em uma classe são chamadas comumente de métodos e têm como objetivo manipular o estado do objeto que será instanciado.
  - Assim como acontece para as variáveis, os métodos podem ser de instância ou estáticos.

- **Escopo**
  - O escopo de uma variável se refere ao local no qual ela pode ser acessada, ou seja, de acordo com o bloco no qual ela tenha sido declarada, ele será local - dentro de um método, por exemplo - ou global - dentro de uma classe, porém fora de qualquer método.

O modelo conceitual de objeto, diz que existem quatro elementos essenciais para que qualquer código seja considerado orientado a objetos, são eles: **abstração**, **encapsulamento**, **modularidade** e **hierarquia**.

- **Abstração**
  - Tem foco na visão externa do objeto. Abstrações são úteis para separar da implementação o comportamento essencial de cada objeto.

- **Encapsulamento**
  - Característica que permite esconder a visão interna de uma abstração.
  - Cada componente de um sistema deve encapsular, ou seja, esconder, os detalhes de sua implementação, de forma a revelar sempre o mínimo possível sobre o seu funcionamento interno.

- **Modularidade**
  - Corresponde à capacidade de se decompor um sistema em um conjunto coeso e fracamente acoplado de módulos.

- **Hierarquia**
  - É a classificação ordenada de abstrações.

Assim, podemos compreender que essencialmente devemos entender também alguns outros conceitos para conseguirmos esclarecer a ideia por detrás da programação orientada a objetos, sabendo é claro que cada um desses conceitos pode ser explorado ainda mais, mas aqui serão abordados apenas de forma introdutória para que se possa obter uma compreensão geral acerca dos mesmos.

- **Acoplamento vs Coesão**
  - **Acoplamento** é a medida do quão fortemente um elemento está conectado, tem conhecimento ou depende de outro ou outros elementos. Um elemento pode ser uma classe, um módulo, um sistema (e assim por diante).
  - **Coesão** é a medida do quão fortemente relacionadas e focadas são as responsabilidades de um elemento.
  - Assim, quanto mais coeso um elemento, menor a tendência desse elemento se tornar fortemente acoplado a outro, e este é o ideal de um projeto.
  - Em um sistema, não existirão elementos que possuam zero acoplamento, pois é natural que os elementos de um sistema dependam uns dos outros. O problema acontece quando essa dependência é tão forte que dificulta o reuso e a mudança do código em questão.

- **Interfaces**
  - Estrutura que pode conter apenas assinaturas de métodos, ou seja, não que não pode conter nenhum tipo de implementação.

- **Composição vs Herança**
  - Quando você compõe objetos, você utiliza um objeto em sua forma existente, como parte de um novo objeto. Para isso utilizamos interfaces.
  - Quando você utiliza a herança, você muda ou refina o comportamento de um objeto e assim precisa garantir que o contrato entre as classes - mãe e filhas -, ou seja, aquilo que foi previsto entre elas, seja respeitado. Nesse caso, utilizamos classes.
  - Na Orientação a Objetos, é devido ao princípio do **polimorfismo** que classes derivadas (filhas) de uma única classe base (mãe) sejam capazes de invocar métodos com as mesmas assinaturas porém com comportamentos distintos para cada uma delas.

O Projeto de Software

Um projeto orientado a objetos visa decompor uma aplicação em uma quantidade de componentes capazes de trabalharem em conjunto, porém de forma a permanecerem independentes.

Algumas orientações gerais para um bom projeto de software são:

- Construa um objeto de forma a revelar sua intenção, ou seja, pense em um objeto em termos de sua interface e não de sua implementação.

- Encapsule! Oculte e resuma o máximo possível da sua implementação, evitando assim variáveis públicas em seus objetos e estabelecendo métodos acessadores para definir e retornar valores.

- Especialize objetos somente quando necessário. Opte sempre que possível pela composição em vez da herança.

- Minimize os relacionamentos entre os objetos de forma a reduzir o acoplamento entre eles.