S T Q Q S S D Programação Orientada a objetos __/__/__

Resumo: Introduction to Object-Oriented Concepts.
Marc Guiderm de Oliveira Porto.

Historicamente, linguagens orientadas à objeto são definidas pelo uso de encapsulação, herança, polimorfismo; então, a uma linguagem não implementar todos esses, geralmente não é considerada orientada à objetos.

Encapsuladores de objeto (Object wrappers) são código orientado a objetos que inclui um outro código dentro dele. Por exemplo, você pode pegar um código estruturado (como loops e condição) para embalar ele dentro de um objeto para fazer ele parecer com um objeto.

Mas o que é um objeto? Um objeto é definido por seus componentes, atributos e comportamentos, por exemplo, uma pessoa contém atributos que a define fisicamente, como cor da pele, olho, cabelo, e ela também possui comportamentos, como andando, falando e respirando. Assim, um objeto é definido por uma entidade que contém dados e comportamentos juntos. (Diferença entre OO e Procedural: OO - os atributos e comportamentos estão no mesmo objeto. Procedural - Eles são normalmente separados.)

Em programação estruturada, os dados geralmente são operados de ações, e normalmente os dados são globais. Já em programação orientada a objetos, os objeto possuem atributos (o tipo da variável) e métodos, que representam o comportamento, para manipular os dados. Além disso, em POO, você pode controlar o acesso para membros de um objeto. Restringindo o acesso a atributos e/ou métodos é chamado de "Data hiding", enquanto juntar atributos e métodos em uma única entidade é chamado de encapsulação.

Na programação procedural, os dados e suas manipulações são separados, enquanto na POO eles são encapsulados a objetos. Neste, utilizamos também classes, que são consideradas como os moldes para objetos. É possível usar o atributo "private" para implementar o "data hiding" no código, enquanto outros globais podem ser atribuídos como "public". Existem também

*É por causa do Data hiding que utilizamos getters e setters.

S T Q Q S S D

__/__/__

interfaces, ou seja, definem como os usuários interagem com a classe, e as implementações que estão dentro do classe que o usuário não precisam ver.

A Herança permite que uma classe receba atributos e métodos de outra classe, promovendo reutilização de código e criação de hierarquias, possibilitando a redução de código duplicado, diferente da composição. Há também a alteração de classe e métodos, que divide o código em partes boas, basicamente simplifica a interação com objeto complexo, fornecendo implementação concreta.

A composição é uma técnica para construir objeto a partir de outros objeto, diferente da herança, ela possui uma relação de "Ter um", ao invés de "ser um". Por último, entre o polimorfismo, no qual objetos similares podem responder à mesma mensagem de maneira diferente.

spiral