Aluno: Kayke Queiroz dos Santos RA: 24101284

A Programação orientada a objeto ficou popular em 1990, isso porque assim ela ficou ideal para aplicações Web. Os conceitos principais da (POO) são:

- Encapsulamento: Protege dados
- Herança: Reutiliza código
- Polimorfismo: Polimorfismo adapta método
- Composição: Organiza objetos

Não são necessariamente essenciais, mas é importante saber.

É importante entender os conceitos, pois exemplo herança entre outros, a (POO) surgiu para o desafio de integrá-la em sistemas antigos (legados), escritos em paradigmas estruturados, mas é realmente necessário a aplicações web e móveis, a (OO), pode ser entregado usado wrappers de objetos, que encapsulam códigos estruturados dentro de objetos, tornando-os mais compatíveis com sistemas modernos, programação Procedural vs Orientada a objetos.

- Objeto: Entidade com atributos (dados) e comportamentos (métodos) + Procedural: dados e função são separados, o código é estruturado e procedimentos independentes.

No início do desenvolvimento do (OO), podia substituir os bancos relacionais, porém, isso não acontecer porque as empresas já haviam investido muito dinheiro em bancos relacionais e eles funcionavam bem, atualmente, o desenvolvimento de software adota uma abordagem híbrida, combinando OO e estruturada.

Os objetos em programação orientada a objetos (OO) vão além de simples estruturas de dados ou tipos primitivos, como inteiros e strings, além de armazenar atributos, eles possuem métodos que definem comportamentos, um benefício essencial dos objetos é o controle de acesso, permitindo ocultar atributos e métodos de outros objetos.

Na programação OO, objetos encapsulam dados e comportamentos. Quando transmitidos pela rede, podem levar ambos ou apenas os dados, caso o código já exista nos dois lados. No design adequado, um navegador pode executar um objeto sem conhecê-lo previamente. Objetos são os blocos fundamentais da programação orientada a objetos, e pode representar o mundo real.

O comportamento de um objeto define suas ações e é implementado por métodos na programação OO. Métodos, como setGender() e getGender(), permitem acessar e modificar atributos de um objeto.

Getters e setters controlam o acesso aos atributos de um objeto, garantindo o princípio de encapsulamento e ocultação dos dados. Também chamados de métodos acessores (getters) e modificadores (setters), eles evitam que objetos modifiquem dados diretamente. Para usá-los, basta conhecer seu nome, parâmetros e tipo e retorno.

Um objeto pode recorrer a mensagens de outros objetos, como no caso de Payroll, que solicita o número de segurança social de objetos Employee. Diagramas de classe UML representam classes em três seções: nome, atributos e métodos.

Ferramentas de modelagem ajudam a criar e visualizar essas classes e suas relações. Classes funcionam como templates, e cada objeto cria uma instância única dessa classe, contendo seus próprios atributos e métodos.

Objetos possuem seus próprios atributos, mas compartilham a implementação de métodos, dependendo da plataforma. Conceitualmente, cada objeto é independente. Uma classe é um modelo para criar objetos, funcionando como um "blueprint". Objetos são instâncias a partir de classes na programação OO.

Uma classe é sempre criada antes de um objeto pois serve como um modelo para sua construção. Assim como variáveis de tipos primitivos (int, float), objetos são instâncias a partir de classes. O livro usa exemplos de código, principalmente em Java, para ilustrar conceitos de programação orientada a objetos.

Uma classe define os atributos e comportamentos que os objetos criados a partir dela terão. No exemplo da classe Pessoa, os atributos name e address são privados e acessados por métodos (getName(), setName(), etc), garantindo encapsulamento. Métodos podem ser públicos, privados ou protegidos, controlando como os objetos interagem. Em POO, classes orientadas a objetos, os atributos de um objeto devem ser modificados apenas por seus próprios métodos, evitando acesso de outros programas.

No exemplo, um objeto payroll chama getName() de um objeto Pessoa, enviando uma mensagem para recuperar o Name.

Diagramas de classe UML são usados para representar visualmente classes, separando atributos e métodos. Um dos avanços do design OO, como diagramas UML, são mais complexos para detalhar interações entre classes.

A encapsulamento e ocultamento de dados garantem que o objeto exponha apenas o necessário para sua interação.

Encapsulamento é o princípio da programação orientada a objetos que mantém atributos e comportamentos dentro de um objeto, garantindo a ocultação de dados.

• Interfaces: São os meios pelos quais os objetos se comunicam, apenas métodos públicos fazem parte da interface.

• Dados privados: atributos devem ser privados para impedir acesso direto, garantindo segurança e controle. Métodos como getters e setters permitem acesso controlado.

• Implementação: Apenas a interface pública deve ser visível ao ~~contrário,~~ usuário, a lógica interna e atributos privados permanecem inacessíveis, facilitando manutenção e segurança.

O Paradigma de interface/implementação separa a forma como o objeto é acessado da forma como ele funciona internamente.

• Exemplo real: Uma torradeira precisa de eletricidade, mas não importa se a fonte é uma usina a carvão ou nuclear.

• Exemplo em código: A classe square deve expor apenas a interface para calcular o quadrado.

• Benefício: Permite modificar a implementação sem afetar o código do usuário, garantindo flexibilidade e manutenção do cliente.

A interface de uma classe define como os usuários interagem com ela, enquanto a implementação interna permanece oculta. No exemplo do Square, o método público getSquare() fornece o resultado, mas o cálculo real calculeSquare() é privado.

·Herança:

A herança permite que uma classe reutilize atributos e métodos de outra, evitando redundância. Por exemplo, ao invés de Dog e Cat terem atributos repetidos, pode reutilizar atributos.

·Subclasse: Classe que herda características da superclasse.

·Superclasse: Classe que mantém atributos e comportamentos gerais para outras classes.

·Herança simples: Cada classe pode ter apenas um pai (Java)

·Herança múltipla: Uma classe pode herdar de várias classes (C++) ↓

Pode gerar conflitos.

·Polimorfismo: Permite que objetos de diferentes classes possam ser tratados de maneira uniforme.

·Sobrecarga: Vários métodos com o mesmo nome.

·Sobreposição: Um método de classe filha sobrescreve da classe mãe.