Programação Orientada a Objetos

A Programação Orientada a Objetos (POO) é um paradigma que organiza o desenvolvimento de sistemas em torno de objetos, que são unidades que encapsulam dados e métodos, facilitando a estruturação modular e reutilizável do software. Esse paradigma surgiu para enfrentar as limitações das técnicas de programação processual e é particularmente eficaz para modelar sistemas complexos.

Fundamentos do POO

O conceito de POO foi consolidado a partir dos trabalhos de **Alan Kay** , que é considerado um dos pioneiros dessa abordagem e idealizou a linguagem **Smalltalk** nos anos 1970. Para Kay, a POO seria uma maneira de criar sistemas complexos baseados em pequenas unidades de código que interagem entre si, o que ele comparou a um ecossistema de pequenas “máquinas” (KAY, 1993).

Paralelamente, **Grady Booch** destacou que a POO organiza o código em torno de abstrações do mundo real, agrupando dados e comportamentos relacionados a objetos (BOOCH, 1994). Essa estrutura melhora a coesão do software e facilita a manutenção. Booch também contribuiu para o desenvolvimento da **Linguagem de Modelagem Unificada (UML)** , uma linguagem que auxilia na modelagem e documentação de sistemas orientados a objetos.

Os 4 pilares da Programação Orientada a Objetos

1. Abstração

A abstração é a prática de definir **classes** em um nível mais alto, focando nos aspectos essenciais do objeto sem expor detalhes complexos. Esse conceito permite simplificar a modelagem do sistema. Para Booch, a abstração é a base do POO, pois permite ao desenvolvedor focar no “o que” uma classe deve fazer, em vez de “como” ela deve fazer (BOOCH, 1994).

- **Exemplo** : Uma classe `Conta` é uma abstração que omite os detalhes específicos de uma conta corrente ou poupança, focando apenas nos métodos essenciais que ambas as particularidades.

2. Encapsulamento

O encapsulamento protege os dados internos de um objeto, permitindo o acesso e modificação apenas por meio de métodos específicos (GETs e SETs). **Robert C. Martin** enfatiza que o encapsulamento é fundamental para o princípio de design "programar para uma interface, não uma implementação", ou que reduz a proteção entre partes do código (MARTIN, 2008).

- **Exemplo** : Em uma classe `Conta` , o atributo `saldo` pode ser privado e acessado apenas por métodos de consulta, garantindo que o saldo não seja alterado diretamente e prevenindo inconsistências nos dados.

3. Herança

A herança é um mecanismo que permite que uma classe derive de outra, aproveitando e especializando suas funcionalidades. Esse conceito permite o uso de código e a criação de instruções. Para **Bjarne Stroustrup** , criador da linguagem C++, a herança é fundamental para a criação de famílias de classes com comportamentos relacionados, o que facilita a extensão de software sem modificação do código original (STROUSTRUP, 2000).

- **Exemplo** : Uma classe `ContaCorrente` pode herdar de `Conta` , adição de características específicas, como taxas de manutenção. Isso permite que você `ContaCorrente` aproveite os métodos e atributos `Conta` e ao mesmo tempo adicione comportamentos próprios.

4. Polimorfismo

O polimorfismo permite que métodos sejam sobrescritos para classes derivadas e que objetos de classes diferentes possam ser tratados de maneira uniforme. Essa característica é essencial para a flexibilidade do código. **Bertrand Meyer** , criador de **Eiffel** , popularizou o conceito de "Design by Contract", defendendo que o polimorfismo auxilia no cumprimento de contratos entre classes, promovendo código confiável (MEYER, 1988).

- **Exemplo** : Se `Conta` tem um método `atualizarSaldo` , tanto `ContaCorrente` quanto `ContaPoupanca` podem sobrescrever esse método para adaptar a lógica de atualização ao tipo de conta específica, mas ambas as classes ainda podem ser tratadas como `Conta` .

Classes e Objetos

O que são Classes?

Uma **classe** é uma estrutura que serve como modelo ou "molde" para criar objetos. Ela define os **atributos** (dados ou propriedades) e **métodos** (funções ou comportamentos) que os objetos desse tipo possuirão. É como uma planta arquitetônica: nela estão especificados os elementos e funcionalidades que todo objeto criado a partir dela terá, mas ela mesma não representa um objeto concreto.

O que são Objetos?

Um **objeto** é uma **instância** de uma classe, ou seja, uma entidade concreta que é criada a partir da estrutura definida pela classe. Cada objeto pode ter valores específicos para os atributos, tornando-o único. Ele representa uma “cópia” da classe com dados próprios, possibilitando a criação de várias instâncias (objetos) que unem a mesma estrutura, mas com características individuais.

Diferença entre Classe e Objeto

- A **classe** é o plano geral que define o que os objetos criados a partir dela terão.
- O **objeto** é uma instância específica que tem valores específicos e pode interagir com outros objetos e métodos do sistema.

Resumo

- **Classe** : Define os atributos e métodos; é o "molde".
- **Objeto** : É uma instância da classe; é o "produto" do molde com dados específicos.

Entender essas diferenças é crucial no POO, pois classes e objetos permitem organizar o código de forma modular, facilitando a criação de sistemas escaláveis e reutilizáveis.

Vantagens do POO

1. **Modularidade** : Um POO facilita a criação de software modular, onde cada classe ou objeto representa uma unidade coesa e independente. Como enfatizado por Martin, isso reduz o custo de manutenção, pois problemas podem ser isolados e corrigidos localmente (MARTIN, 2008).

2. **Reusabilidade** : O uso de herança e polimorfismo permite a reutilização de código, promovendo o desenvolvimento de sistemas robustos e menos propensos a erros, como destacado por **Bruce Eckel** (2006) em "Thinking in Java".

3. **Flexibilidade** : Um POO permite que sistemas sejam flexíveis e extensíveis, facilitando a adaptação de software a mudanças futuras.

4. **Simplicidade no Design** : Ao utilizar abstração e encapsulamento, o POO melhorou a compreensão e legibilidade do código, algo que Booch considera essencial para qualquer projeto de software de grande porte (BOOCH, 1994).

Referências

- BOOCH, Grady. *Análise e Projeto Orientados a Objetos com Aplicações* . Addison-Wesley, 1994.
- KAY, Alan. *A História Inicial do Smalltalk* . Avisos do ACM SIGPLAN, 1993.
- MARTIN, Robert C. *Código Limpo: Um Manual de Artesanato Ágil de Software* . Prentice Hall, 2008.
- MEYER, Bertrand. *Construção de Software Orientada a Objetos* . Prentice Hall, 1988.
- STROUSTRUP, Bjarne. *A Linguagem de Programação C++* . Addison-Wesley, 2000.
- ECKEL, Bruce. *Pensando em Java* . Prentice Hall, 2006.