Java Developer

Java Core

Sumário

1. Orientação a Objetos (OO) **6**
2. Diagrama de classe **12**
3. Exercícios **13**

Objetivos deste Módulo

Ao final deste módulo, objetiva-se que o aluno adquira:

Conhecimentos:

- Orientação a Objetos: classes, atributos, métodos, objetos, visibilidade, encapsulamento, construtores
- Diagrama de classe

1. Orientação a Objetos (OO)

A orientação a objetos é um paradigma de programação que se baseia no conceito de "objetos". É uma abordagem de design de software que modela sistemas complexos como uma coleção de objetos interconectados, cada um representando uma entidade no mundo real ou virtual.

Os objetos são instâncias de classes, que são como modelos ou planos para criar objetos. As classes definem os atributos (dados) e métodos (comportamentos) que os objetos podem ter. Os objetos podem interagir uns com os outros, chamando os métodos de outros objetos ou acessando seus atributos. Essa interação é central para a programação orientada a objetos e é conhecida como encapsulamento.

Principais conceitos da programação orientada a objetos:

- **Classes**: São os modelos para criar objetos. Elas definem os atributos (variáveis) e métodos (funções) que os objetos terão.

- **Objetos**: São instâncias de classes. Cada objeto possui seus próprios atributos e métodos, mas compartilha a mesma estrutura definida pela classe.

- **Encapsulamento**: É o princípio de esconder os detalhes internos de uma classe e fornecer uma interface pública para interagir com ela. Isso ajuda a manter o código mais organizado e a reduzir dependências.

O encapsulamento em Java é um conceito fundamental da programação orientada a objetos (POO) que envolve a restrição do acesso direto aos atributos (variáveis de instância) de uma classe e a promoção do acesso a esses atributos por meio de métodos públicos. Ele tem como objetivo principal proteger o estado interno de um objeto e fornecer um controle mais rigoroso sobre como os dados são manipulados e acessados.

Aqui estão os principais aspectos do encapsulamento em Java:

Atributos Privados: Os atributos de uma classe são declarados como privados usando o modificador private. Isso significa que eles só podem ser acessados diretamente dentro da própria classe.

Exemplo:

```
private String nome;
private int idade;
```

Métodos de Acesso (Getters e Setters): Para permitir o acesso controlado aos atributos privados, a classe fornece métodos públicos chamados "getters" e "setters". Os "getters" são usados para obter o valor de um atributo, enquanto os "setters" são usados para definir ou modificar o valor de um atributo.

Exemplo de um "getter" e um "setter":

```
public String getNome() {
    return nome;
}

public void setNome(String novoNome) {
    nome = novoNome;
}
```

Validações e Lógica Personalizada: Os métodos "setters" podem incluir validações e lógica personalizada para garantir que os dados atribuídos aos atributos estejam em um estado válido. Isso é útil para manter a consistência e a integridade dos dados.

Exemplo com validação em um "setter":

```
public void setIdade(int novaIdade) {

    if (novaIdade >= 0) {
        idade = novaIdade;
    } else {
        System.out.println("A idade não pode ser negativa.");
    }

}
```

Controle de Acesso: O encapsulamento permite que a classe defina como os atributos podem ser acessados e modificados. Por exemplo, você pode impedir que atributos sejam modificados de fora da classe ou permitir apenas leitura, conforme necessário.

Proteção contra Acesso Indesejado: O encapsulamento ajuda a evitar acesso não autorizado ou modificação inadequada dos dados de um objeto, uma vez que o acesso direto aos atributos é restrito.

Herança: Permite que uma classe herde os atributos e métodos de outra classe. Isso promove a reutilização de código e a criação de hierarquias de classes.

Polimorfismo: Permite que objetos de diferentes classes sejam tratados de maneira genérica. Isso é alcançado por meio de interfaces ou classes base comuns.

Abstração: É o processo de simplificar uma classe ou objeto para destacar apenas os detalhes relevantes para o contexto em questão. Ajuda a lidar com a complexidade.

A programação orientada a objetos é amplamente utilizada em muitas linguagens de programação, incluindo Java, Python, C#, e muitas outras. Ela ajuda a organizar e estruturar o código de maneira mais modular, facilitando o desenvolvimento, a manutenção e a escalabilidade de software complexo.

Exemplo de **classe** em Java:

```
public class Pessoa {

    // Atributos (variáveis de instância)
    private String nome;
    private int idade;

    // Construtor
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    // Método para obter o nome da pessoa
    public String getNome() {
        return nome;
    }

    // Método para definir o nome da pessoa
    public void setNome(String nome) {
        this.nome = nome;
    }

    // Método para obter a idade da pessoa
    public int getIdade() {
        return idade;
    }

    // Método para definir a idade da pessoa
    public void setIdade(int idade) {
        this.idade = idade;
    }

    // Método para exibir informações da pessoa
    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade + " anos");
    }
}
```

Neste exemplo, a classe "Pessoa" tem dois atributos (nome e idade), um construtor para inicializar esses atributos, métodos "get" e "set" para acessar e modificar os atributos, e um método "exibirInformacoes" para mostrar as informações da pessoa.

Você pode criar **objetos da classe "Pessoa"** usando o **construtor** e acessar seus métodos e atributos da seguinte forma:

```
public class ExemploPessoa {

    public static void main(String[] args) {

        // Criando um objeto Pessoa
        Pessoa pessoa1 = new Pessoa("João", 30);

        // Acessando os atributos e métodos do objeto
        System.out.println("Nome: " + pessoa1.getNome());
        System.out.println("Idade: " + pessoa1.getIdade() + " anos");

        // Modificando os atributos
        pessoa1.setNome("Maria");
        pessoa1.setIdade(25);

        // Exibindo as informações atualizadas
        pessoa1.exibirInformacoes();

    }
}
```

Este é um exemplo simples de como criar e usar uma classe em Java. As classes são componentes fundamentais na programação orientada a objetos e são usadas para modelar objetos e suas interações em sistemas de software.

Classe e **objeto** são dois conceitos fundamentais na programação orientada a objetos (POO), e eles desempenham papéis diferentes no design e na estrutura de um programa.

Classe:

- Uma classe é uma estrutura ou um modelo que define os atributos (variáveis) e os métodos (funções) que um objeto desse tipo terá.
- Ela atua como um plano ou uma especificação para criar objetos.
- Classes são a parte estática da POO e não têm existência real em tempo de execução.
- Podem ser reutilizadas para criar múltiplos objetos semelhantes.

Objeto:

- Um objeto é uma instância concreta ou real de uma classe.
- Ele é criado com base na definição da classe e representa uma entidade específica no mundo real.
- Objetos têm estado (valores de atributos) e comportamento (métodos) associados a eles.
- Cada objeto é único e pode ter seus próprios valores de atributos.

Agora, falando em **visibilidade**… Em Java, a visibilidade (também conhecida como controle de acesso) refere-se às regras que determinam quais partes de uma classe (atributos, métodos, construtores) podem ser acessadas por outras classes ou objetos.

Java fornece **quatro níveis de visibilidade**, que são controlados por **modificadores de acesso**. Esses modificadores de acesso são:

public: O membro (atributo ou método) é acessível de **qualquer lugar**, ou seja, de qualquer classe ou pacote.

Por exemplo, se um método ou atributo é declarado como público, outras classes podem acessá-lo diretamente, sem restrições.

Exemplo:

```
public class MinhaClasse {

    public int meuAtributo;

    public void meuMetodo() {
        // Código aqui
    }

}
```

protected: O membro (atributo ou método) é acessível por classes do **mesmo pacote** e por **subclasses** (mesmo que estejam em pacotes diferentes). Isso permite um nível de encapsulamento, onde os membros protegidos podem ser usados por classes relacionadas.

Exemplo:

```
public class MinhaClasse {

    protected int meuAtributo;

    protected void meuMetodo() {
        // Código aqui
    }

}
```

default (ou package-private): O membro (atributo ou método) é acessível apenas por classes no **mesmo pacote**. Se você não especificar um modificador de acesso (ou seja, não usar public, protected ou private), o membro será considerado "default". Isso restringe o acesso a classes dentro do mesmo pacote, mas não permite o acesso fora dele.

Exemplo:

```
class MinhaClasse {
    int meuAtributo;
    void meuMetodo() {
        // Código aqui
    }
}
```

private: O membro é acessível **apenas dentro da própria classe**. Isso fornece o mais alto nível de encapsulamento, garantindo que os detalhes internos da classe não sejam acessíveis a partir de outras classes. Os membros privados são frequentemente usados para esconder a implementação interna de uma classe.

Exemplo:

```
public class MinhaClasse {

    private int meuAtributo;

    private void meuMetodo() {
        // Código aqui
    }

}
```

É importante escolher o modificador de acesso apropriado com base nos requisitos de design e encapsulamento do seu programa. O uso adequado dos modificadores de acesso ajuda a manter a integridade dos dados e a tornar o código mais seguro e mais fácil de manter.

2. Diagrama de classe

Um diagrama de classe é uma representação gráfica de uma classe em programação orientada a objetos (POO). Ele faz parte das ferramentas de modelagem da UML (Unified Modeling Language - Linguagem de Modelagem Unificada) e é amplamente utilizado para visualizar a estrutura de classes e suas relações em um sistema de software.

Os diagramas de classe têm o objetivo de mostrar as principais características de uma classe, incluindo seus atributos (variáveis) e métodos (funções), bem como as relações entre diferentes classes no sistema. Eles são valiosos para comunicar o design e a arquitetura de um sistema de software de forma clara e compreensível.

Neste diagrama de classe simplificado:

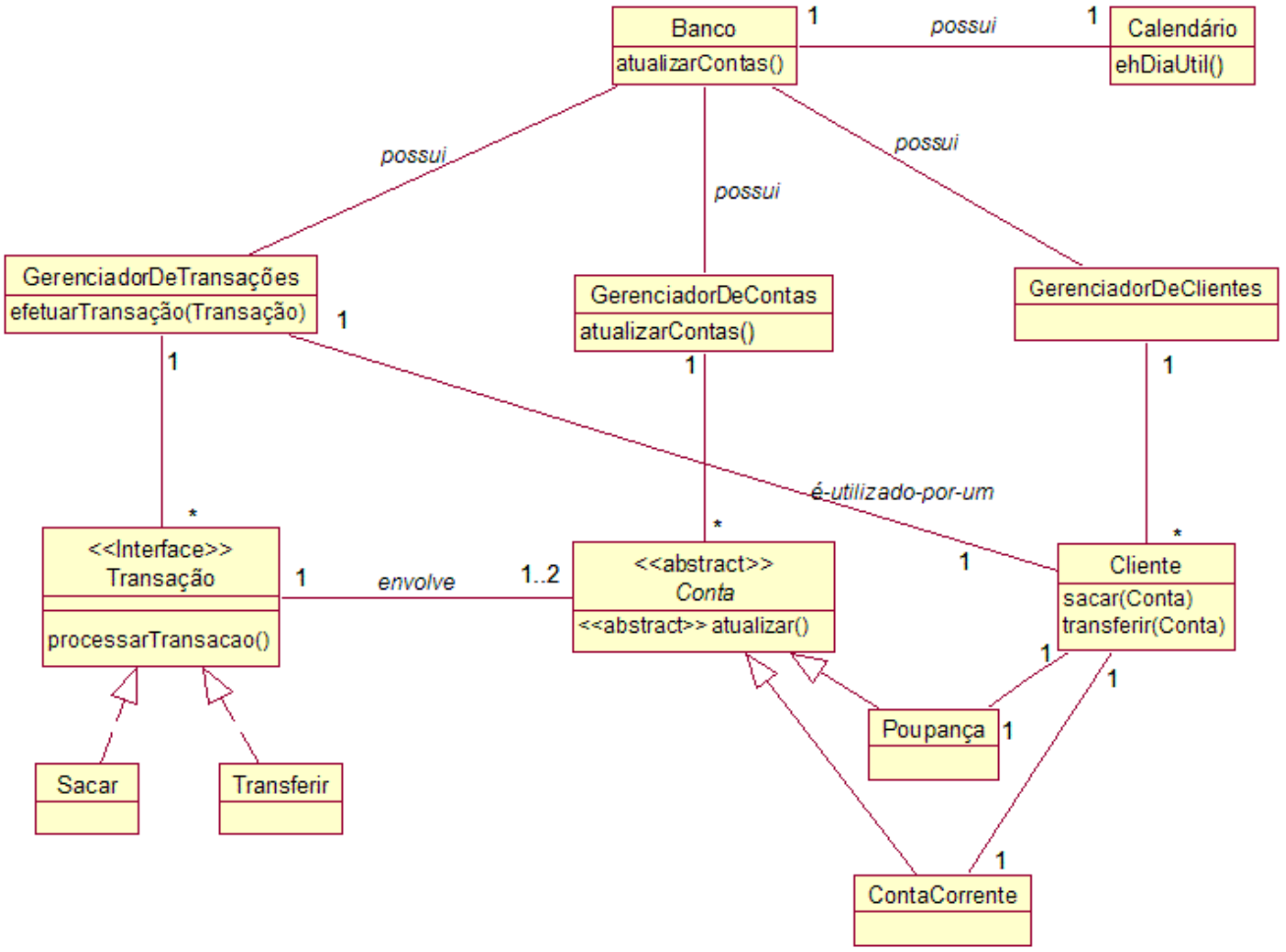
A classe Livro possui atributos como Título e ISBN, que representam o título e o número ISBN de um livro. Além disso, há atributos para relacionar um livro com seu autor e editora.

A classe Autor possui atributos como Nome e Data de Nascimento, que representam informações sobre o autor de um livro.

A classe Editora possui atributos como Nome e Local, que representam informações sobre a editora de um livro.

Existem métodos públicos nas classes Livro para acessar informações como título, ISBN, autor e editora, bem como métodos para emprestar e devolver um livro (essas são operações fictícias apenas para ilustração).

Lembre-se de que este é um diagrama de classe textual simplificado e que os diagramas reais são geralmente representados graficamente usando ferramentas de modelagem UML, o que torna a visualização e a compreensão muito mais fáceis. Como este, por exemplo:



Se você quiser ler mais sobre UML, seguem duas referências:

http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/uml/diagramas/classes/classes1.htm

https://www.omg.org/spec/UML/

3. Exercícios

1) Crie uma classe chamada ContaBancária com atributos como saldo e titular. Implemente métodos para depositar, sacar e verificar o saldo. Use encapsulamento para proteger o acesso direto aos atributos. Realize algumas operações em uma conta bancária e exiba os resultados.

2) Adicione a classe conta bancária, um atributo Correntista. Correntista deverá ser uma nova classe com os atributos correspondentes.

3) Defina uma classe Endereco com seus atributos e adicione-o como atributo na classe Correntista.