RELATÓRIO - TORRE DE HANÓI

1. Introdução

Este projeto, desenvolvido para a disciplina de Estruturas de Dados do Centro Universitário Farias Brito, consiste na implementação do jogo Torre de Hanói.

2. Funcionamento do Jogo

O jogo consiste em mover todos os discos de uma haste inicial para uma haste final, usando uma terceira como apoio. As regras são:

- Mover um disco por vez.

- Nunca colocar um disco maior sobre um menor.

3. Estruturas Utilizadas

Cada haste foi modelada como uma pilha, implementada com lista encadeada. Essa escolha permite manipulação eficiente de discos, respeitando a lógica do jogo: mover apenas o disco do topo e verificar a validade dos movimentos.

4. Ações do Jogo

- **Inicialização**: O jogador escolhe de 3 a 5 discos, que são empilhados na Torre A.

- **Movimento**: O jogador escolhe a haste de origem e destino (ex: "A C"). O sistema valida e executa o movimento se for permitido.

- **Controle e Vitória**: O jogo continua até que todos os discos estejam corretamente empilhados na Torre C. Não há penalidades — movimentos inválidos são apenas ignorados.

5. Regras

- Mover apenas um disco por vez.
- Nunca colocar disco maior sobre um menor.
- A Torre B pode ser usada como apoio.
- Vitória: todos os discos na Torre C, na ordem correta.

6. Estrutura do Código (Java)

- **Node**: Representa um disco, com tamanho e ponteiro para o próximo.
- **Tower**: Representa uma haste, implementada como pilha ligada de Nodes. Possui métodos como `push`, `pop`, `peek` e `isEmpty`.
- **HanoiGame**: Gerencia as torres, controla os movimentos, verifica a vitória e interage com o jogador. Possui métodos para inicialização, movimentação e exibição do estado do jogo.

7. Partes Significativas do Código e Justificativas Técnicas

A seguir, trechos relevantes do código que ilustram a estrutura de dados e a lógica do jogo:

1. Classe `Node` – Representa um disco

```java
class Node {
    int data;      // Tamanho do disco
    Node next;     // Próximo disco na pilha

    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

Justificativa:

Cada disco é representado por um `Node`, que armazena seu tamanho e referência ao próximo. Essa estrutura forma a base da pilha com lista encadeada.

2. Classe `Tower` – Representa uma haste como pilha

```java
class Tower {
    private Node top; // Disco no topo

    public void push(int data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
    }

    public Node pop() {
        if (isEmpty()) return null;
        Node removed = top;
        top = top.next;
        return removed;
    }

    public boolean isEmpty() {
        return top == null;
    }

    public Node peek() {
        return top;
    }
}
```

Justificativa:

A pilha permite empilhar (`push`) e desempilhar (`pop`) discos de forma eficiente, respeitando a regra de só mover o disco do topo. A estrutura encadeada facilita alterações dinâmicas.

3. Método `moveDisk` – Movimentação entre torres

```
public void moveDisk(char source, char destination) {
    Tower src = getTower(source);
    Tower dest = getTower(destination);

    if (src == null || dest == null) {
        System.out.println("Torre inválida.");
        return;
    }

    if (src.isEmpty()) {
        System.out.println("Torre " + source + " está vazia.");
        return;
    }

    Node topSrc = src.peek();
    Node topDest = dest.peek();

    if (!dest.isEmpty() && topSrc.data > topDest.data) {
        System.out.println("Movimento inválido: disco maior sobre menor.");
        return;
    }

    dest.push(src.pop().data);
    System.out.println("Movido disco " + topSrc.data + " de " + source + " para " + destination +
}
```

Justificativa:

Controla a lógica principal do jogo: verifica validade do movimento (incluindo regra de tamanho), e executa a transferência entre torres com `pop` e `push`.

8. Compilação e Execução do Programa

Pré-requisitos

- Ter o **Java Development Kit (JDK)** instalado.

- Verifique com: `javac -version`.

Compilação do Código-Fonte

1. Salve as classes `Node`, `Tower` e `HanoiGame` em arquivos `Node.java`, `Tower.java` e `HanoiGame.java`, na mesma pasta.

2. No terminal, navegue até a pasta dos arquivos.

3. Compile com:

```
javac Node.java Tower.java HanoiGame.java
```

Se não houver erros, arquivos `.class` serão gerados no mesmo diretório.

Execução do Programa

Execute a classe principal com:

```
java HanoiGame
```

O programa será iniciado no terminal. O usuário poderá definir o número de discos e realizar os movimentos entre as torres.

Criação e Uso do Arquivo `.jar` (opcional)

Crie um arquivo `manifest.txt` com:

```
Main-Class: HanoiGame
```

(sem espaços após o nome da classe).

Gere o `.jar` com:

```
jar cfm HanoiGame.jar manifest.txt *.class
```

Execute o jogo com:

```
java -jar HanoiGame.jar
```

Essa opção facilita a distribuição e execução do jogo.

9. Capturas de Tela e Exemplo de Uso

```
Bem-vindo ao Jogo da Torre de Hanói!
Digite o número de discos (3 a 5): 3
Torre A: 1 2 3
Torre B:
Torre C:
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ac
Movendo disco 1 da Torre A para a Torre C.
Torre A: 2 3
Torre B:
Torre C: 1
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ab
Movendo disco 2 da Torre A para a Torre B.
Torre A: 3
Torre B: 2
Torre C: 1
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): cb
Movendo disco 1 da Torre C para a Torre B.
Torre A: 3
Torre B: 1 2
Torre C:
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ac
Movendo disco 3 da Torre A para a Torre C.
```

```
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ac
Movendo disco 3 da Torre A para a Torre C.
Torre A:
Torre B: 1 2
Torre C: 3
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ba
Movendo disco 1 da Torre B para a Torre A.
Torre A: 1
Torre B: 2
Torre C: 3
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): bc
Movendo disco 2 da Torre B para a Torre C.
Torre A: 1
Torre B:
Torre C: 2 3
Digite a torre de origem (A, B ou C) e a torre de destino (A, B ou C), separadas por espaço (ex: A B): ac
Movendo disco 1 da Torre A para a Torre C.
Torre A:
Torre B:
Torre C: 1 2 3
Parabéns! Você resolveu a Torre de Hanói!
```

10. Considerações Finais

O desenvolvimento do jogo **Torre de Hanói em Java**, com a implementação manual de pilhas por meio de listas encadeadas, demonstrou na prática a aplicação de estruturas de dados fundamentais. A limitação de não utilizar bibliotecas prontas da linguagem fortaleceu o entendimento do funcionamento interno das pilhas.

A escolha da pilha foi natural e adequada, considerando a mecânica do jogo baseada no princípio **LIFO (Last-In, First-Out)**. A estrutura `Node` representando cada disco, junto com a classe `Tower` para as hastes, proporcionou controle total sobre o encadeamento e manipulação de dados.

A lógica central do jogo foi encapsulada na classe `HanoiGame`, garantindo separação de responsabilidades e modularidade no código. Apesar de a interface ser textual, a estrutura adotada permitiria expansão futura para uma interface gráfica, sem necessidade de alterações na lógica principal.

Em suma, o projeto atingiu seu objetivo funcional e serviu como um exercício prático valioso para reforçar conceitos de estruturas de dados encadeadas, controle de fluxo, modularização e boas práticas de programação orientada a objetos em Java.