Estrutura de Dados I – 2025.1 - Noturno

Nome: ALAN DELON SOUSA ROCHA **Matrícula:** 2320048

Nome: Marcelo Henrique de Oliveira Rocha **Matrícula:** 2410426

1. Introdução

O Blackjack, um dos jogos de cartas mais populares em cassinos ao redor do mundo, tem uma história um tanto nebulosa, mas acredita-se que suas raízes remontam a jogos europeus mais antigos.

O objetivo do Blackjack é ter uma mão com valor total mais próximo de 21 do que o dealer, sem ultrapassar este valor. Ultrapassar 21 resulta em uma derrota automática, conhecida como "estourar" ou "bust".

Valores das Cartas:

- Cartas numéricas (2-10): Valem seu valor nominal
- Figuras (J, Q, K): Valem 10 pontos cada
- Ás (A): Vale 1 ou 11 pontos (o valor que mais beneficiar a mão)
- Funcionamento Geral:

Início da Partida

- O jogador fornece seu nome
- As cartas são embaralhadas
- Cada jogador recebe duas cartas iniciais, viradas para cima
- O dealer recebe duas cartas - uma virada para cima (visível a todos) e outra virada para baixo

Turno do Jogador

- O jogador tem duas opções em cada turno
- Pedir carta (Hit): Recebe uma carta adicional
- Parar (Stand): Encerra seu turno, mantendo as cartas atuais
- O jogador pode continuar pedindo cartas até:
    - Decidir parar
    - Atingir exatamente 21 (Blackjack)
    - Ultrapassar 21 (estourar/bust), resultando em derrota automática

Turno do Dealer

- Após o jogador concluir seu turno (e se não tiver estourado):
- O dealer revela sua carta oculta
- O dealer segue regras pré-determinadas:
- Com 16 pontos ou menos: deve pedir carta (hit)
- Com 17 pontos ou mais: deve parar (stand)
- Se o dealer estourar (ultrapassar 21), o jogador vence automaticamente

Determinação do Vencedor

Jogador vence se:

Tiver um valor mais próximo de 21 que o dealer

Dealer estourar (ultrapassar 21)

Dealer vence se:

Tiver um valor mais próximo de 21 que o jogador

Jogador estourar

Empate (Push): Quando ambos têm o mesmo valor

Final da Rodada

Após cada rodada, o jogador pode escolher jogar novamente ou encerrar o jogo

Estruturas de Dados Utilizadas e Justificativas

Estrutura Principal:

1. **Lista (ArrayList)**:
    a. **Classe:** `Mão (Mao.java)`
    b. **Justificativa:** A classe `Mão` utiliza uma lista para armazenar as cartas de um jogador. A escolha de `ArrayList` é apropriada porque permite fácil adição de cartas (`add`) e acesso sequencial para calcular o valor da mão, operações que se beneficiam da eficiência da lista.
2. **Lista Duplamente Encadeada (Node)**:
    a. **Classe:** `Baralho (Baralho.java)`
    b. **Justificativa:** O baralho é implementado como uma lista duplamente encadeada. Essa escolha permite adição e remoção dinâmicas de cartas (no início ou no final) com complexidade constante (`O(1)`), ideal para simular o embaralhamento e manipulação de cartas.
3. **Classe Customizada: `Carta`**:
    a. **Classe:** `Carta (Carta.java)`
    b. **Justificativa:** A classe `Carta` encapsula detalhes como nome e valor, seguindo princípios de encapsulamento e modularidade. Isso facilita o cálculo do valor da mão e mantém clara a representação do baralho.

Exemplos de Uso:

- **Adição de Cartas na Mão:**

```java
public void adicionarCarta(Carta carta) {
    cartas.add(carta);
}
```
S

Motivação: Operação simples e eficiente em uma estrutura de lista.

- **Manipulação do Baralho:**

```java
private void adicionar(Carta carta) {
    Node novoNode = new Node(carta);
    if (cabeca == null) {
        cabeca = novoNode;
        cauda = novoNode;
    } else {
        cauda.proximo = novoNode;
        novoNode.anterior = cauda;
        cauda = novoNode;
    }
    tamanho++;
}
```

Motivação: Manipulação dinâmica do baralho com suporte para remoção e inserção no início/fim.Caso precise de mais detalhes ou explicações sobre os trechos específicos do código, é só avisar! Seguem os links para consulta dos arquivos.

Implementação

No projeto de Blackjack, a estrutura foi implementada de maneira modular e orientada a objetos, utilizando classes para representar os elementos centrais do jogo. Vamos detalhar:

Classe Jogo:

Gerencia o fluxo principal do jogo, incluindo a inicialização, distribuição de cartas e exibição dos resultados.

Exemplo de implementação:

```java
public class Jogo {
    private Baralho baralho;
    private Jogador jogador;
    private Jogador dealer;

    public Jogo() {
        this.baralho = new Baralho();
        this.jogador = new Jogador("Jogador");
        this.dealer = new Jogador("Dealer");
    }

    public void iniciar() {
        baralho.criarBaralho();
        baralho.embaralhar();
        distribuirCartas();
    }
}
```

Classe Baralho:

Representa o baralho utilizando uma estrutura de lista duplamente encadeada (Node), com métodos para criar, embaralhar e manipular cartas.

```java
private void criarBaralho() {
    String[] naipes = {"Copas", "Espadas", "Ouros", "Paus"};
    for (String naipe : naipes) {
        for (int valor = 2; valor <= 10; valor++) {
            adicionar(new Carta(valor + " de " + naipe, valor));
        }
    }
}
```

S

Classe Jogador:

Cada jogador possui um nome e uma mão de cartas (Mao), com métodos para adicionar cartas e calcular o valor total da mão.

```java
public class Jogador {
    private String nome;
    private Mao mao;

    public Jogador(String nome) {
        this.nome = nome;
        this.mao = new Mao();
    }
}
```

Classe Mao:

Gerencia as cartas na mão de um jogador e calcula o valor total, ajustando o valor dos Ases caso exceda 21.

```java
public int calcularValor() {
    int valorTotal = 0;
    int ases = 0;
    for (Carta carta : cartas) {
        valorTotal += carta.getValor();
        if (carta.getValor() == 11) {
            ases++;
        }
    }
    while (valorTotal > 21 && ases > 0) {
        valorTotal -= 10;
        ases--;
    }
    return valorTotal;
}
```

Classe Carta:

Representa uma carta com nome e valor, sendo a unidade básica do jogo.

```java
public class Carta {
    private String nome;
    private int valor;

    Continue
    public Carta(String nome, int valor) {
        this.nome = nome;
        this.valor = valor;
    }
```

A implementação orientada a objetos facilita a organização da lógica do Blackjack, permitindo modularidade e reuso de código. Aqui estão alguns impactos na dinâmica do jogo:

Fluxo Modular:

A classe Jogo centraliza o gerenciamento do jogo, chamando métodos das outras classes (Baralho, Jogador, Mao) para realizar as operações necessárias, como distribuir cartas e calcular valores.

Flexibilidade nas Regras:

A lógica para cálculo do valor das mãos (como ajustar o Ás entre 1 e 11) está encapsulada na classe Mao, tornando-a independente do restante do código.

Expansibilidade:

A estrutura modular permite adicionar novos recursos ou alterações no jogo, como suporte para múltiplos jogadores ou variações nas regras.

Com essa arquitetura, o jogo mantém uma lógica clara e organizada, refletindo as regras e dinâmicas do Blackjack de forma eficiente e expansível.

Ações do Jogo e Desenvolvimento

No jogo de Blackjack, as ações principais incluem:

- Criar o baralho: A estrutura do baralho é gerada com cartas contendo seus respectivos valores e naipes.
- Embaralhar o baralho: Após ser criado, o baralho é embaralhado.
- Distribuir cartas: Cada jogador (inclusive o dealer) recebe duas cartas no início do jogo.
- Decisões do jogador: O jogador pode decidir se deseja:
- Pedir mais cartas (hit).
- Parar e manter sua mão (stand).
- Ações do dealer: O dealer segue uma lógica fixa, geralmente baseada nas regras do Blackjack (por exemplo, pedir cartas até atingir 17 ou mais pontos).
- Cálculo do vencedor: Ao final, o valor das mãos é comparado para determinar o vencedor.

Regras e Mecânicas

- Cada jogador continua pedindo cartas até decidir parar ou até ultrapassar 21 pontos (estourar).
- Se passar de 21, perde automaticamente.
- Quando ambos param, compara-se os pontos:
    - Quem tiver mais pontos sem passar de 21, vence.
    - Se empatarem, é empate

Compilação e Execução

1- Certifique-se de ter o Java instalado em sua máquina.

2- Clone este repositório ou baixe os arquivos do projeto.

3- Navegue até o diretório do projeto.

4- Compile os arquivos Java usando o comando:

javac src/blackjack/*.java
5- Execute o jogo com o comando:

java src/blackjack/App

Tutorial de Compilação e Execução do Blackjack Java

Requisitos

- Java Development Kit (JDK) 8 ou superior instalado
- Acesso à linha de comando (Command Prompt ou PowerShell)

Compilação e Execução via Linha de Comando

Compilação

1. Abra o Command Prompt ou PowerShell
2. Navegue até o diretório do projeto:
3. Compile todos os arquivos Java: `javac *.java`

Esta instrução compila todos os arquivos Java no diretório, incluindo `App.java`, Carta.java, `Jogo.java` e `MinhaList.java`.

Execução

Após compilar com sucesso, execute o programa usando:

```
‘’java App’’
```

Compilação e Execução via VS Code

1. **Abra o projeto no VS Code**:
    a. Abra VS Code
    b. Selecione File > Open Folder...
    c. Navegue até a pasta do blackjack e clique em "Selecionar pasta"
2. **Para compilar e executar via VS Code**:
    a. Abra o arquivo `App.java`
    b. Clique no botão "Run Java" (ícone de play) no canto superior direito
    c. Ou pressione F5 para executar com depuração
3. **Alternativa**: Use o Terminal integrado
    a. Pressione Ctrl+` para abrir o terminal
    b. Execute os mesmos comandos de compilação e execução listados acima

Resolução de Problemas Comuns

Erro: 'javac' não é reconhecido

Significa que o Java JDK não está no PATH do sistema.

Solução:

1. Verifique se o JDK está instalado
2. Adicione o diretório bin do JDK ao PATH do sistema
3. Ou use o caminho completo para javac:`"C:\Program Files\Java\jdk-x.x.x\bin\javac" *.java`

Erro: ClassNotFoundException

Ocorre quando a VM Java não consegue encontrar uma classe.

Solução: Certifique-se de que está executando o programa no mesmo diretório onde os arquivos .class foram gerados.

Erro: Could not find or load main class App

Solução:

1. Verifique se está no diretório correto
2. Verifique se o arquivo App.class foi gerado
3. Execute com o nome completo: `java -cp . App`

Erro: Compilation errors

Se houver erros de compilação:

1. Leia cuidadosamente as mensagens de erro
2. Verifique se todos os arquivos necessários estão presentes:
    a. App.java
    b. Carta.java
    c. Jogo.java
    d. MinhaList.java
3. Corrija os erros indicados e tente novamente