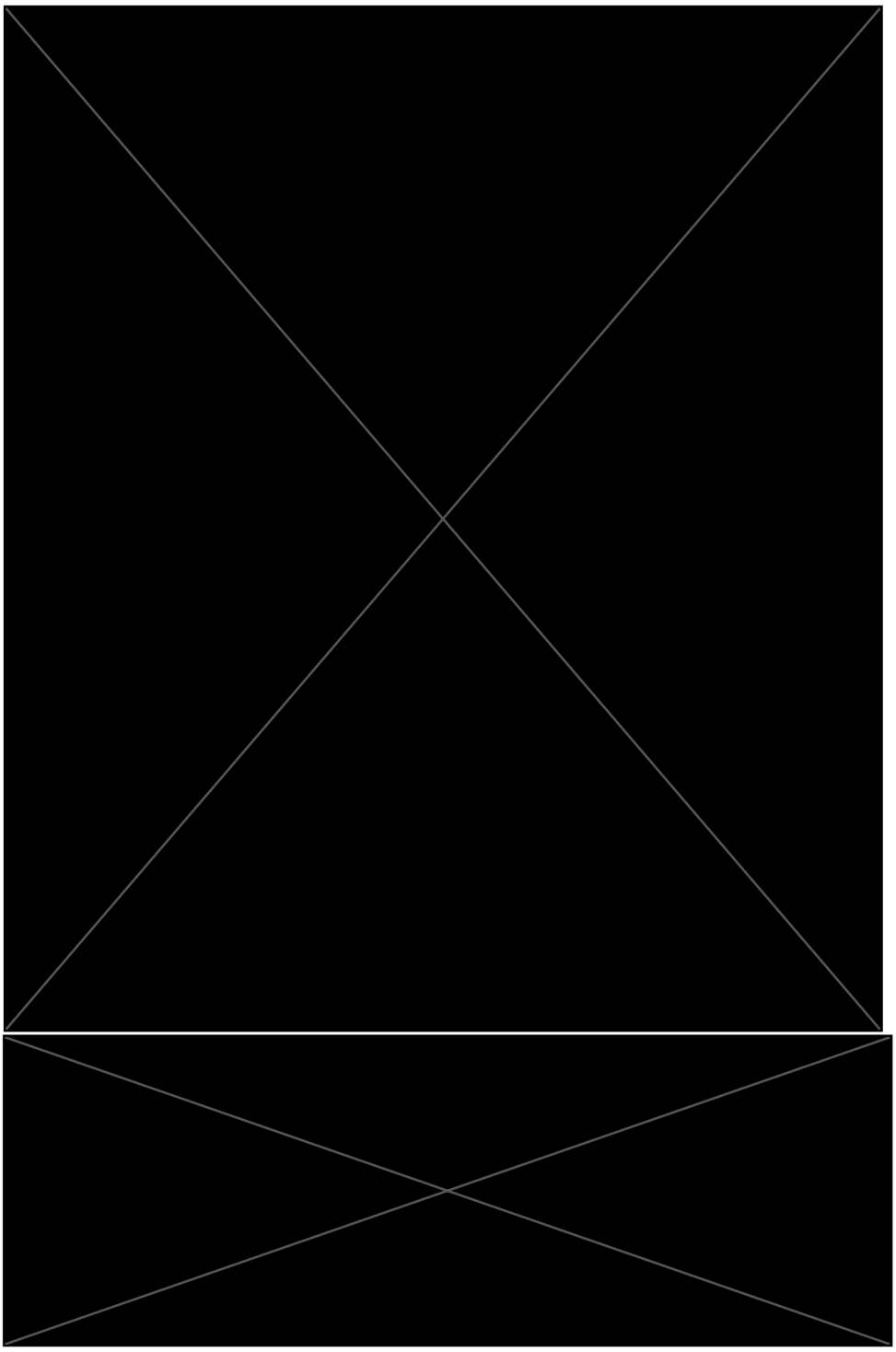

Cheatsheet de Programação Orientada a Objetos em Java

1. Fundamentos da Programação Orientada a Objetos (OOP)

1.1. Classes e Objetos

- **Classe**: Define o molde para criar objetos.
- **Objeto**: Instância de uma classe.

Exemplo:

```java
class Carta {
    String naipe;
    String rank;
    // Construtor
    Carta(String naipe, String rank) {
        this.naipe = naipe;
        this.rank = rank;
    }
}
```

- **Uso**:

```java
```

```
Carta minhaCarta = new Carta("Espadas", "Ás");
```

1.2. Construtores

- Método especial para inicializar objetos.

Exemplo:

```java
class Jogador {
    String nome;
    Jogador(String nome) {
        this.nome = nome;
    }
}
```

1.3. Encapsulamento

- **Modificadores de Acesso**: Controlam a visibilidade dos atributos e métodos. `private` impede o acesso direto.
- **Getters e Setters**: Métodos para acessar e modificar atributos privados.

Exemplo:

```java
class Jogador {
    private String nome;

    public String getNome() {
        return nome;
    }

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

1.4. Herança

- Permite que uma classe herde atributos e métodos de outra.

Exemplo:

```java
class Utilizador {
    String username;
    void fazerLogin() {
        System.out.println("Login de utilizador.");
    }
}

class Jogador extends Utilizador {
    int pontuacao;
    void jogar() {
        System.out.println("Jogando...");
    }
}
```

1.5. Polimorfismo

- Sobrescreve métodos da classe-pai.

Exemplo:

```java
class Utilizador {
    void fazerLogin() {
        System.out.println("Login genérico.");
    }
}

class Jogador extends Utilizador {
    @Override
    void fazerLogin() {
        System.out.println("Login de jogador.");
    }
}
```

---

2. Estruturas de Controle

2.1. Condicionais (if/else)

- Controla o fluxo de execução baseado em condições.

Exemplo:

```java
if (baralho.size() > 0) {
    System.out.println("Existem cartas no baralho.");
} else {
    System.out.println("O baralho está vazio.");
}
```

2.2. Laço for (tradicional)

- Itera por um número definido de vezes.

Exemplo:

```java
for (int i = 0; i < 10; i++) {
    System.out.println("Jogada número " + i);
}
```

2.3. Laço for-each

- Itera sobre coleções como arrays e `ArrayList`.

Exemplo:

```java
for (Carta carta : baralho) {
    System.out.println(carta.rank + " de " + carta.naipe);
}
```

2.4. While

- Executa enquanto uma condição for verdadeira.

Exemplo:

```java
while (baralho.size() > 0) {
    // Comprar uma carta
    baralho.remove(0);
}
```

---

3. Estruturas de Dados

3.1. Arrays

- Estrutura de dados fixa.

Exemplo:

```java
int[] numeros = {1, 2, 3, 4, 5};
```

3.2. ArrayList

- Coleção dinâmica que pode crescer ou diminuir conforme necessário.

Exemplo:

```java
ArrayList<Carta> baralho = new ArrayList<>();
baralho.add(new Carta("Espadas", "Rei"));
baralho.remove(0); // Remove a primeira carta
```

3.3. Random

- Gera números aleatórios, útil em jogos de cartas.

Exemplo:

```java
Random random = new Random();
int cartaAleatoria = random.nextInt(baralho.size());
```

---

4. Boas Práticas e Dicas

4.1. Uso de Modificadores de Acesso

- Sempre que possível, torne atributos privados e use `getters` e `setters`.

4.2. Reutilização de Código com Herança

- Crie uma classe genérica para reutilizar comportamento, como `Utilizador`.

4.3. Teste o Código com Cenários Reais

- Use loops para simular jogadas de cartas, removendo cartas do baralho conforme elas são compradas:

```java
for (int i = 0; i < 5; i++) {
    if (baralho.size() > 0) {
        Carta carta = baralho.remove(0);
        System.out.println("Você comprou " + carta.rank + " de " + carta.naipe);
    }
}
```

---

5. Exercícios Práticos

1. **Criação de Objetos**: Crie um baralho de cartas e simule a compra de 5 cartas.
2. **Controle de Fluxo**: Crie um loop `for` para iterar sobre um array de cartas e imprimir cada uma.
3. **Manipulação de ArrayList**: Simule a remoção de cartas de um baralho até que ele esteja vazio.
4. **Herança e Polimorfismo**: Crie uma subclasse de `Jogador` que seja um "Jogador VIP" e modifique o comportamento de alguns métodos.

Referências Importantes

- `ArrayList`: Para coleções dinâmicas.
- `Random`: Para sorteios aleatórios.
- Modificadores de acesso (`private`, `public`): Para encapsulamento.
- Laços `for`, `while`: Para repetição de ações.
- Condicionais `if/else`: Para controle do fluxo de execução.

Essa cheatsheet irá te ajudar a praticar os conceitos fundamentais da programação orientada a objetos em Java de forma organizada e prática.