

Manual de Conceitos Java para Exercícios

1. Criação de Classes e Atributos

A criação de classes é fundamental para organizar o código. Uma classe deve conter atributos e métodos que representam o comportamento de seus objetos. Veja um exemplo de uma classe com atributos e métodos simples:

- Exemplo de classe Circulo:

```
public class Circulo {  
    private double raio;  
  
    // Construtor  
    public Circulo(double raio) {  
        this.raio = raio;  
    }  
  
    // Getter  
    public double getRaio() {  
        return raio;  
    }  
  
    // Setter  
    public void setRaio(double raio) {  
        this.raio = raio;  
    }  
  
    // Método para calcular a área  
    public double calcularArea() {  
        return Math.PI * Math.pow(raio, 2);  
    }  
}
```

2. Herança e Polimorfismo

Herança é um princípio onde uma classe herda propriedades e métodos de outra. Polimorfismo permite que métodos de uma subclasse sobrescrevam os da classe pai.

- Exemplo de classe Veiculo e Carro:

```

public class Veiculo {
    private String modelo;
    private String marca;

    public Veiculo(String modelo, String marca) {
        this.modelo = modelo;
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }

    public String getMarca() {
        return marca;
    }

    public void exibirDetalhes() {
        System.out.println("Modelo: " + modelo + ", Marca: " + marca);
    }
}

public class Carro extends Veiculo {
    private int numeroDePortas;

    public Carro(String modelo, String marca, int numeroDePortas) {
        super(modelo, marca);
        this.numeroDePortas = numeroDePortas;
    }

    @Override
    public void exibirDetalhes() {
        super.exibirDetalhes();
        System.out.println("Número de Portas: " + numeroDePortas);
    }
}

```

3. Encapsulamento

Encapsulamento protege os dados de uma classe tornando seus atributos privados e fornecendo métodos públicos para acessá-los e modificá-los.

- Exemplo de encapsulamento em Pessoa:

```

public class Pessoa {
    private String nome;
    private int idade;

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

```

```

        this.idade = idade;
    }

    public String getNome() {
        return nome;
    }

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

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }
}

```

4. Tratamento de Exceções

Exceções lidam com erros inesperados no código. Para evitar que o programa falhe, usamos blocos 'try-catch' para capturar e tratar esses erros.

- Exemplo de tratamento de exceção ao ler arquivos:

?

Exemplo:

java

Copiar código

```

public void lerArquivo(String nomeArquivo) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(nomeArquivo));
        String linha;
        while ((linha = reader.readLine()) != null) {
            System.out.println(linha);
        }
    } catch (IOException e) {
        System.out.println("Erro ao ler o arquivo: " + e.getMessage());
    }
}

```

5. Laços de Repetição

Laços de repetição como 'for', 'while', e 'do-while' são usados para executar código repetidamente enquanto uma condição for verdadeira.

Exemplo de `for` :

java

Copiar código

```
for (int i = 0; i < 10; i++) {  
    System.out.println("Valor de i: " + i);  
}
```

Exemplo de `while` :

java

Copiar código

```
int i = 0;  
while (i < 10) {  
    System.out.println("Valor de i: " + i);  
    i++;  
}
```

6. Uso de Map e HashMap

Map e HashMap são estruturas que armazenam pares de chave e valor. Útil para associar, por exemplo, nomes de produtos e suas quantidades no estoque.

- Exemplo de uso de HashMap:

Exemplo de `HashMap` :

java

Copiar código

```
import java.util.HashMap;  
  
public class Loja {  
    private HashMap<String, Integer> estoque = new HashMap<>();  
  
    public void adicionarProduto(String nomeProduto, int quantidade) {  
        estoque.put(nomeProduto, quantidade);  
    }  
  
    public void exibirEstoque() {  
        for (String produto : estoque.keySet()) {  
            System.out.println("Produto: " + produto + ", Quantidade: " + estoque.get(produto));  
        }  
    }  
}
```


7. Manipulação de Datas com LocalDate

LocalDate é uma classe para trabalhar com datas no Java. Ela é útil para fazer cálculos como a diferença entre duas datas ou verificar dias específicos.

- Exemplo de manipulação de datas com LocalDate:

Exemplo:

java

 Copiar código

```
import java.time.LocalDate;
import java.time.temporal.ChronoUnit;

public class Datas {
    public void calcularDiferenca(LocalDate data1, LocalDate data2) {
        long diasDiferenca = ChronoUnit.DAYS.between(data1, data2);
        System.out.println("Diferença em dias: " + diasDiferenca);
    }
}
```


8. Sobrecarga de Métodos

Sobrecarga de métodos permite definir múltiplos métodos com o mesmo nome, mas parâmetros diferentes. Isso é útil para quando queremos que o método se comporte de maneira diferente dependendo dos parâmetros.

- Exemplo de sobrecarga de métodos:

Exemplo:

java

 Copiar código

```
public class Calculadora {
    public int somar(int a, int b) {
        return a + b;
    }

    public double somar(double a, double b) {
        return a + b;
    }
}
```