1.1. Tratamento com `try-catch`

- **Descrição**:
  - O bloco `try` é usado para encapsular o código que pode gerar exceções.
  - Caso uma exceção seja lançada, o bloco `catch` é executado para tratar o erro.
- **Exemplo:**

```
class Conta {
  numero: string;
  saldo: number;

  constructor(numero: string, saldo: number = 0) {
    this.numero = numero;
    this.saldo = saldo;
  }

  transferir(valor: number, contaDestino: Conta): void {
    if (valor > this.saldo) {
      throw new Error("Saldo insuficiente para transferência.");
    }
    this.saldo -= valor;
    contaDestino.saldo += valor;
  }
}

// Exemplo de uso
const conta1 = new Conta("123", 100);
const conta2 = new Conta("456", 50);

try {
  conta1.transferir(200, conta2);
} catch (error) {
  console.error(`Erro: ${(error as Error).message}`);
}

console.log(`Saldo conta 1: ${conta1.saldo}`);
console.log(`Saldo conta 2: ${conta2.saldo}`);
```

1.2. Tratamento com `if-else`

- **Descrição**:
  - Usa condições (`if`) para verificar erros antes que eles ocorram.
  - Permite lidar de forma simples com cenários esperados.
- **Exemplo:**

```
class Conta {
  numero: string;
  saldo: number;

  constructor(numero: string, saldo: number = 0) {
    this.numero = numero;
    this.saldo = saldo;
  }

  transferir(valor: number, contaDestino: Conta): void {
    if (valor > this.saldo) {
      console.error("Erro: Saldo insuficiente para transferência.");
      return;
    }
    this.saldo -= valor;
    contaDestino.saldo += valor;
    console.log(`Transferência de R$${valor} realizada com
sucesso.`);
  }
}

// Exemplo de uso
const conta1 = new Conta("123", 100);
const conta2 = new Conta("456", 50);

conta1.transferir(200, conta2);

console.log(`Saldo conta 1: ${conta1.saldo}`);
console.log(`Saldo conta 2: ${conta2.saldo}`);
```

1.3. Tratamento com `assert`

- **Descrição**:
  - Garante que uma condição seja verdadeira durante a execução.

    - Se a condição falhar, uma exceção é levantada.
    - Em TypeScript, usa a biblioteca `assert` do Node.js.
- **Exemplo:**

```
import assert from "assert";

class Conta {
  numero: string;
  saldo: number;

  constructor(numero: string, saldo: number = 0) {
    this.numero = numero;
    this.saldo = saldo;
  }

  transferir(valor: number, contaDestino: Conta): void {
    assert(valor <= this.saldo, "Erro: Saldo insuficiente para
transferência.");
    this.saldo -= valor;
    contaDestino.saldo += valor;
    console.log(`Transferência de R$${valor} realizada com
sucesso.`);
  }
}

// Exemplo de uso
const conta1 = new Conta("123", 100);
const conta2 = new Conta("456", 50);

try {
  conta1.transferir(200, conta2);
} catch (error) {
  console.error(`Erro: ${(error as Error).message}`);
}

console.log(`Saldo conta 1: ${conta1.saldo}`);
console.log(`Saldo conta 2: ${conta2.saldo}`);
```

2. Limitações de uso dos métodos acima

1. **`try-catch`**:
    - Pode ser usado em excesso, deixando o código menos legível.
    - Se o bloco `catch` for genérico, pode capturar erros que não deveriam ser tratados naquele contexto.
2. **`if-else`**:
    - Torna o código mais verboso quando há várias condições.
    - Não trata exceções inesperadas, apenas aquelas previstas no código.
3. **`assert`**:
    - Não é apropriado para produção, pois pode parar a execução abruptamente.
    - Depende de bibliotecas externas no caso do TypeScript (como `assert` do Node.js).