

 <p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PIAUÍ</p>	<p>INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ Curso: ADS Disciplina: Programação Orientada a Objetos Professor: Ely</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Exercício 08

- 1) Enumere os 3 tipos mais comuns de tratamento de erros e exemplifique com códigos seus ou pesquisados na internet.

Os tratamentos de erros mais comuns:

- Exibir mensagem de erro;
- Retornar um código de erro;
- Exceções.

Os códigos dessa questão estão no anexadas em arquivo typescript.

- 2) Explique por que cada um dos 3 métodos acima possui limitações de uso.

As principais limitações são:

1 - Limitação de Compreensão:

As mensagens de erro ao usuário geralmente são destinadas a usuários finais e devem ser compreensíveis para eles. Isso pode limitar a quantidade de detalhes técnicos que você pode fornecer, o que pode dificultar a depuração para desenvolvedores.

2 - Registros de Erros no Console ou em Arquivos de Log:

Limitação de Visibilidade:

Dependendo da configuração do ambiente de produção, os registros de erro podem não estar visíveis para os usuários finais, o que pode dificultar a identificação e resolução de problemas em ambientes de produção.

3 - Tratamento de Exceções e Fluxo de Controle:

Limitação de Performance:

Em algumas linguagens, o tratamento de exceções pode ter um custo de desempenho significativo quando comparado a lógica de controle de fluxo tradicional. O uso inadequado de exceções para casos não excepcionais pode impactar negativamente a performance.

- 3) Implemente como nos slides o lançamento da exceção no método sacar e realize um teste para saques que deixariam o saldo negativo.

Os códigos dessa questão estão no anexadas em arquivo typescript.

- 4) Crie duas contas e teste o método transferir de modo que a conta a ser debitada não possua saldo suficiente. Explique o que ocorreu.

O método transferir chama internamente o método sacar da Conta Origem. Se a Conta Origem não tiver saldo suficiente, uma exceção será lançada e capturada pelo bloco catch.

- 5) Instancie uma classe banco e crie duas contas. Adicione-as à instancia do banco. Chame o método transferir novamente passando um valor que lance a exceção na classe conta. Você considera que o lançamento da exceção foi “propagado” para o método conta.transferir(), banco.transferir() e o método transferir do script app? Como você avalia a confiabilidade dessa implementação.

Sim, se uma exceção é lançada no método sacar dentro do método transferir da classe Conta, ela será propagada para métodos superiores, incluindo banco.transferir() e app.transferir().

Isso ocorre porque, ao tentar realizar a transferência, banco.transferir() chama conta.transferir(), e qualquer exceção que não foi tratada será propagada para métodos superiores.

- 6) Lance um erro no construtor e nos métodos sacar e depositar para que, caso o valor passado seja menor que zero uma exceção seja lançada. Reexecute os testes da questão anterior com valores que “passem” pelo saldo insuficiente, e teste também a chamada dos métodos passando como parâmetro valores < 0.

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 7) Crie as classes AplicacaoError descendente de Error. Crie também classes ContaNexistenteError e SaldoInsuficienteError. Todas decedentes da classe AplicacaoError.

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 8) Implemente na classe Banco os métodos consultar e consultarPorIndice para que, caso a conta procurada não seja encontrada, a exceção ContaNexistente seja lançada.

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 9) Altere os métodos alterar, depositar, sacar, transferir, renderJuros removendo os “ifs/elses”, pois caso haja exceção no método consultar, os respectivos códigos não serão mais necessários. Ex:

Antes	Depois
<pre>x(numero: string): void { let procurada = consultar(numero); if (procurada != null) { conta.metodoY(...); } }</pre>	<pre>x(numero: string): void { let procurada = consultar(numero); conta.metodoY(...); }</pre>

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 10) Crie uma exceção chamada `ValorInvalidoError` que herda de `AplicacaoException` e altere a classe `Conta` para que ao receber um crédito/depósito, caso o valor seja menor ou igual a zero, seja lançada a exceção `ValorInvalidoError`. Altere também o construtor da classe `Conta` para que o saldo inicial seja atribuído utilizando o método `depositar`.

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 11) Você percebeu que o código que valida se o valor é menor ou igual a zero se repete nos métodos `sacar` e `depositar`? Refatore o código criando um método privado chamado `validarValor` onde um valor é passado como parâmetro e caso o mesmo seja menor ou igual a zero, seja lançada uma exceção. Altere também os métodos `sacar` e `depositar` para chamar esse método de validação em vez de cada um lançar a sua própria exceção, evitando assim a duplicação de código.
- 12) Crie uma exceção chamada `PoupancaInvalidaError` que herda de `AplicacaoError`. Altere então o método `renderJuros` da classe `Banco` para que caso a conta não seja uma poupança, a exceção criada seja lançada.
- 13) Crie uma validação para não cadastrar mais de uma conta com o mesmo número. Para isso, chame o método `consultar` no método `inserir` da classe `banco`. Apenas se a exceção do método `consultar` for lançada, você deve incluir a conta. Para isso, consulte a conta dentro de um `try` e o faça a inclusão no `catch`

Os codigos dessa questão estão no anexadas em arquivo typescript.

- 14) Altere a aplicação `“app.ts”` para que tenha um tratamento de exceções no `do {} while` mostra a estrutura do slide `“Aplicação Robusta”`.
- 15) Crie exceções relacionadas a valores obtidos da entrada de dados que não sejam aceitáveis, como valores vazios, números inválidos etc. Na aplicação, trate todas as entradas de dados para que, caso o usuário infrinja regras de preenchimento, o sistema lance e trate as exceções e informe que a entrada foi inválida.
- Nota: nenhuma das exceções lançadas por você ou pela aplicação deve `“abortar”` o programa. Elas devem ser obrigatoriamente tratadas.