	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ** **Curso: ADS** **Disciplina: Programação Orientada a** **Objetos Professor: Ely** **Discente: Sâmia Dantas Braga**

Exercício 08

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

Resposta:

1. Exibir mensagem de erro;
2. Retornar um código de erro;
3. Exceções.

Exemplos, no git.

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

Resposta:

1. Exibir mensagem de erro:

- Não há maneira de capturar e tratar o erro em outro ponto do código. Uma vez que a mensagem foi exibida, o programa continua sua execução sem uma maneira eficiente de corrigir ou reagir ao erro.
- Se o erro for exibido diretamente no console ou interface gráfica, pode ser difícil rastrear ou depurar a causa do erro.
- Se o programa precisar ser adaptado para outras interfaces (por exemplo, API ou interface gráfica), exibir uma mensagem no console não será útil. Além disso, o usuário final pode não compreender mensagens técnicas.
- Se a mensagem de erro não for visível para o desenvolvedor ou usuário (como em sistemas web), o erro pode passar despercebido, dificultando a detecção e correção.

2. Retornar um código de erro:

- O código acaba cheio de verificações de retorno. Para cada função que pode

gerar um erro, é necessário verificar o código de erro e tratá-lo, o que pode deixar o código denso e difícil de manter, especialmente em sistemas complexos.

- Se uma função chama outra que também pode retornar um código de erro, o desenvolvedor precisa sempre verificar e passar esse código para o próximo nível. Isso resulta em código repetitivo e propenso a erros.
- Os códigos de erro muitas vezes não são descritivos o suficiente para explicar a causa exata do erro. Um código de erro numérico como -1 ou 2 não é intuitivo e pode precisar de documentação extensa para ser compreendido.
- O retorno de códigos de erro só captura os erros que foram explicitamente programados. Qualquer erro inesperado ou não tratado (como um erro de execução ou falha de alocação de memória) não será capturado adequadamente.

3. Exceções:

- Lançar e capturar exceções pode ser mais custoso em termos de desempenho em comparação com o retorno de códigos de erro.
- Exceções devem ser usadas apenas para lidar com condições verdadeiramente excepcionais, mas às vezes os desenvolvedores as utilizam para tratar situações esperadas (como validação de entrada), o que pode resultar em código confuso e menos eficiente.
- A facilidade de capturar exceções pode levar a capturas excessivas ou generalizadas de erros, onde erros importantes são ignorados ou mascarados, tornando mais difícil identificar problemas críticos.
- Se as exceções forem tratadas de forma inadequada, pode ser difícil determinar se a aplicação está se comportando corretamente após um erro.

3) Com o código repassado, implemente o como nos slides o lançamento da exceção no método sacar e realize um teste para saques que deixariam o saldo negativo.

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.

Resposta:

Emitiu um erro com uma mensagem informando saldo insuficiente.

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.

Resposta:

- **Na classe Conta**: A exceção é lançada no método transferir quando o saldo é insuficiente.
- **Na classe Banco**: O método Banco.transferir chama o método Conta.transferir. Se a exceção for lançada na classe Conta, ela será "propagada" para o método Banco.transferir automaticamente.
- **No Script Principal (app)**: A exceção continua a ser propagada até o try-catch no script principal, onde você pode tratá-la. Isso demonstra que a exceção foi corretamente propagada pelas camadas: primeiro na Conta, depois no Banco, e finalmente no script.

Essa implementação é confiável, pois usa exceções para garantir que erros críticos (como saldo insuficiente) sejam tratados. O uso de exceções facilita a separação do fluxo de controle normal e o fluxo de erro, permitindo uma manipulação robusta dos casos em que a transferência não pode ser realizada.

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.

7) Crie as classes AplicacaoError descendente de Error. Crie também classes ContaInexistenteError e SaldoInsuficienteError. Todas decendentes da classe AplicacaoError.

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

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

x(numero: string): void {
let procurada =
consultar(numero); if (procurada
!= null) {
conta.metodoY(...);
}
}
	x(numero: string): void {
	let procurada =
	consultar(numero);
	çconta.metodoY(...);
	}

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.

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 render juros da classe Banco para que caso a conta não seja uma poupança, a exceção criada seja lançada.

13) Altere a aplicação “app.ts” para que tenha um tratamento de exceções no do {} while mostra a estrutura do slide “Aplicação Robusta”.

14) 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.