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

Exercício 08

1) Enumere os 3 tipos mais comuns de tratamento de erros e exemplifique com códigos seus ou pesquisados na internet.
2) Explique por que cada um dos 3 métodos acima possui limitações de uso.
3) Implemente 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.
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.
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) 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

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.