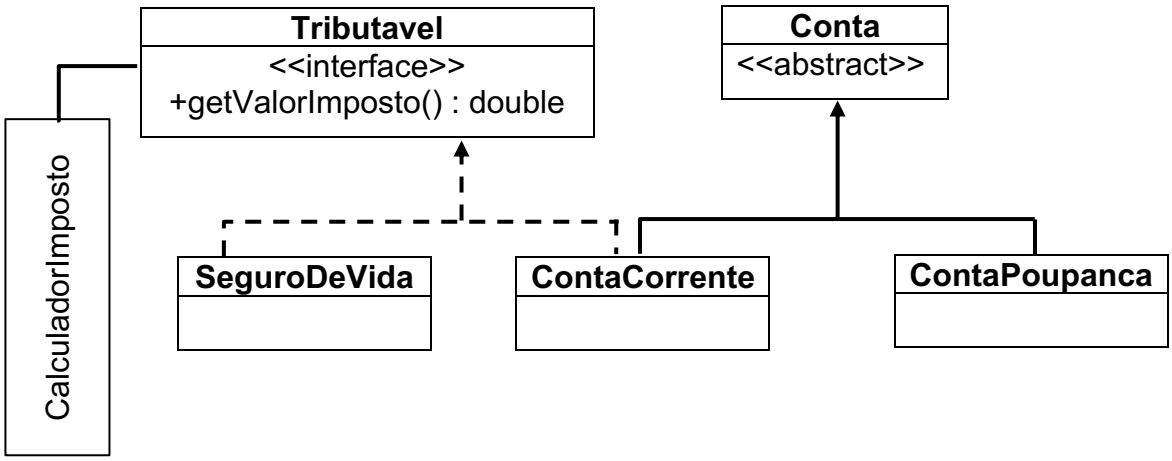
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - AMS		
AVALIAÇÃO OFICIAL	**DISCIPLINA: TÉCNICAS AVANÇADAS DE PROGRAMAÇÃO**	**NOTA**
DATA: 06/11/2024 ☐ N1 ☐ N2 ☐ N3 ☒ N4	**TURMA: ADS - AMS** **PROFESSOR: FERNANDO TONIOLLI**	
ALUNO:	**RA:**	
Data devolução para o aluno: _____/_____/_____	*Ciência do Aluno (vista de prova)*	
INSTRUÇÕES: **INSTRUMENTO DE AVALIAÇÃO: Projeto Sistema Bancário – Parte 4** **CONDIÇÕES: Trabalho individual** **TEMPO MÁXIMO DE DURAÇÃO: Postar no Teams até 19/novembro/2024**		

1) Projeto Sistema Bancário - Parte 4 – JAVA Exceções

Prazo de entrega: até 19/novembro/2024

Essa é a Parte 4 do Projeto do Sistema Bancário que trabalhamos com avaliação N3.
Você deve dar continuidade às classes criadas na Parte 3 que possui a seguinte estrutura de classes:



Nesse Projeto, Parte 4, você deve explorar os conceitos de Java Exception que estudamos nas últimas aulas. Para isso, reveja o material postado no TEAMS e o resumo dos pontos principais abaixo:

Na Parte 3 do Projeto, foi criado o método sacar, que testa a disponibilidade de saldo antes de fazer o saque, considerando a somatória do valor desejado para saque, mais a taxa de R$0,20 por saque (caso necessário, ajuste o código de suas classes da Parte 3 antes de começar a fazer a Parte 4):

```
public class ContaCorrente extends Conta implements Tributavel{

	public ContaCorrente(int agencia, int numero) {
		super(agencia, numero);
	}

	@Override
	public boolean sacar(double valor) {
	   double valorASacar = valor + 0.2;
	   return super.sacar(valorASacar);
	}

	@Override
	public void depositar(double valor) {
		this.saldo += valor;
	}

	@Override
	public double getValorImposto(){
	   return super.saldo * 0.01;
	}
}
```

```
public abstract class Conta {

	private int numeroAgencia;
	private int numeroConta;
	private String tipoConta;
	protected double saldo;
	private String nomeTitular;
	private static int quantidadeContas;

	public Conta(int agencia, int conta) {
		Conta.quantidadeContas++;
		System.out.println("O total de contas abertas é " +
Conta.quantidadeContas);

		this.numeroAgencia = agencia;
		this.numeroConta = conta;
		System.out.println("Número da conta criada é " +
this.numeroConta);
	}

	public abstract void depositar(double valor);

	public boolean sacar(double valor) {
		if(this.saldo >= valor) {
			this.saldo -= valor;
			return true;
		} else {
			return false; }
	}

	public boolean transferir(Conta destino, double valor) {
		if(this.sacar(valor)) {
			destino.depositar(valor);
			return true;
		} else {
			return false;
		  }
	}

	// parte final do código foi omitida //

}
```

Uma sequência de IF-ELSE poderia ser usada na lógica da aplicação para considerar outras condições de negativa de saque além de saldo insuficiente, como por exemplo, limite de valor de saque individual; ou limite de saques a múltiplos de R$20 (caixa eletrônico com apenas cédulas de R$20 disponíveis).

Vejam um exemplo da Classe Conta com o método sacar () que que testa 3 condições antes de permitir o saque:

1) O saldo em conta é maior ou igual ao valor a ser sacado?
2) O valor que se quer sacar é menor que o limite máximo por saque de R$300?
3) O valor que se quer sacar é múltiplo de R$20?

```
public abstract class Conta {

	private int numeroAgencia;
	private int numeroConta;
	private String tipoConta;
	protected double saldo;
	private String nomeTitular;
	private static int quantidadeContas;

	public Conta(int agencia, int conta) {
		Conta.quantidadeContas++;
		System.out.println("O total de contas abertas é " +
Conta.quantidadeContas);
		this.numeroAgencia = agencia;
		this.numeroConta = conta;
		System.out.println("Número da conta criada é " +
this.numeroConta);
	}

	public abstract void depositar(double valor);

	public boolean sacar(double valor) {

	  if(this.saldo >= valor & valor <= 300 & valor % 20 == 0){
			this.saldo -= valor;
			return true;
	   } else {
			return false;
	   }
	}

	public boolean transferir(Conta destino, double valor) {
		if(this.sacar(valor)) {
			destino.depositar(valor);
			return true;
		} else {
			return false;
		  }
	}

	// restante do código omitido //

}
```

Atente-se para uma limitação desse código que retorna false caso uma das condições testadas seja false, porém não especifica qual das condições foi falsa.
Dessa forma, veja na Classe Teste Conta a impossibilidade de imprimirmos uma mensagem de erro específica:

```
public class TesteConta {

	public static void main(String[] args) {
		Conta cc1 = new ContaCorrente(111,222);
		cc1.depositar(100.00);

		Conta cp1 = new ContaPoupanca(222,222);
		cp1.depositar(200.00);

		if (cc1.transferir(cp1, 10.00)) {
			System.out.println("CC1: " + cc1.getSaldo());
			System.out.println("CP1: " + cp1.getSaldo());
		} else {
			System.out.println("Saque não permitido!");
		}

		if (cc1.transferir(cp1, 90)) {
			System.out.println("CC1: " + cc1.getSaldo());
			System.out.println("CP1: " + cp1.getSaldo());
		} else {
			System.out.println("Saque não permitido!");
		}

		if(cc1.transferir(cp1, 10)) {
			System.out.println("CC: " + cc1.getSaldo());
			System.out.println("CP: " + cp1.getSaldo());
		} else {
			System.out.println("Saque não permitido!");
		}
}
```

Para melhorar esse código, poderia ser feito a quebra do teste da condição tripla - if(this.saldo >= valor & valor <= 300 & valor % 20 == 0) – em 3 IF-ELSEs encadeados com retorno de um INT, por exemplo, assim:

0 – sucesso no saque;
-1 – saldo insuficiente;
-2 – valor de saque superior ao limite máximo;
-3 – valor do saque não pode ser composto por cédulas de R$20.

No entanto, essa alternativa é considerada “code smells” uma vez que as boas práticas não recomendam longos encadeamentos de IF-ELSE que dificultam a legibilidade e a manutenção do código.

Uma possibilidade de escrever um código muito melhor seria com o uso de Java Exception, como no exemplo a seguir. Percebam que há uma inversão na lógica, ou seja, quando a condição do IF for atendida, é uma situação de falha para a realização do saque. E não há else, pois a exceção interrompe a execução do código.

Agora, é com vocês!!!
Alterem o código que vocês fizeram na Parte 3 do Projeto N3 de forma a testar na classe sacar() as seguintes condições para saque:

1) O saldo em conta é maior ou igual ao valor a ser sacado?
2) O valor que se quer sacar é menor que o limite máximo por saque de R$300?
3) O valor que se quer sacar é múltiplo de R$20?

Após a realização do saque e/ou da transferência, de acordo com o resultado, imprima mensagens específicas que devem ser no mínimo tão específica quanto:

– sucesso no saque;
– saldo insuficiente;
– valor de saque superior ao limite máximo;
– valor do saque não pode ser composto por cédulas de R$20.

Use em seu código Java Exception, para dessa forma, evitar o encadeamento de IF-ELSE. Veja no exemplo acima, a criação da exceção SaldoInsuficienteException, do tipo Exception.
Atenção à criação das demais exceções e aos ajustes necessários nas classes Conta, ConaCorrente.
Reescreva a classe TesteConta de forma a testar todas as exceções além de saque e transferência funcionando.

Poste no TEAMS todos os arquivos.JAVA individualmente, de forma que sua visualização seja imediata ao abrir a atividade no TEAMS.

Poste também no TEAMS um “print” da tela inteira do eclipse com o resultado da execução da Classe de Teste.

Atenção: NÃO poste os arquivos em pastas ou em arquivos compactados ZIP ou RAR para não inviabilizar sua abertura no Teams.