Lista de Exercícios 12 – Exceções

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

Herculano De Biasi
herculano.debiasi@unoesc.edu.br

Lista de Exercícios 11

1. Seja a classe `Calculadora` abaixo om o método `dividir()`

modulo2 - Calculadora.java

```java
public class Calculadora {
    public static int dividir(int numero1, int numero2) {
        int resultado = numero1 / numero2;

        return resultado;
    }
}
```

   a) Verifique, antes da divisão, se o denominador é zero, e em caso afirmativo, lance uma exceção `ArithmeticException` com `throw`

   b) No programa principal chame o método `dividir(10, 0)`, o programa irá abortar

   c) Trate agora essa exceção no código do programa principal e imprima a mensagem com o método `getMessage()`, note que a mensagem `null` irá aparecer

   d) Para evitar isso, passe a mensagem `"Erro de divisão por zero!"` ao lançar a exceção

   e) Apesar de não ser exigido pelo Java, acrescente a declaração `throws` na linha 2 indicando que essa exceção pode ser lançada neste método

UNOESC

Apoiadores:

Lista de Exercícios 11

2. Agora, em vez de usar a classe de exceção do Java `ArithmeticException`, crie a própria exceção chamada `ErroDivisaoPorZeroException`

   a) Faça que ela herde (*extends*) de `ArithmeticException`

   b) Essa classe terá um atributo chamado `numerador` do tipo `int`

   c) Faça o método `getNumerador()` para este atributo

   d) Crie nela um construtor que receba uma mensagem (*string*) e um número (*int*) – a mensagem deve ser repassada para a superclasse com o método `super()` e o número deverá ser atribuído ao atributo `numerador`

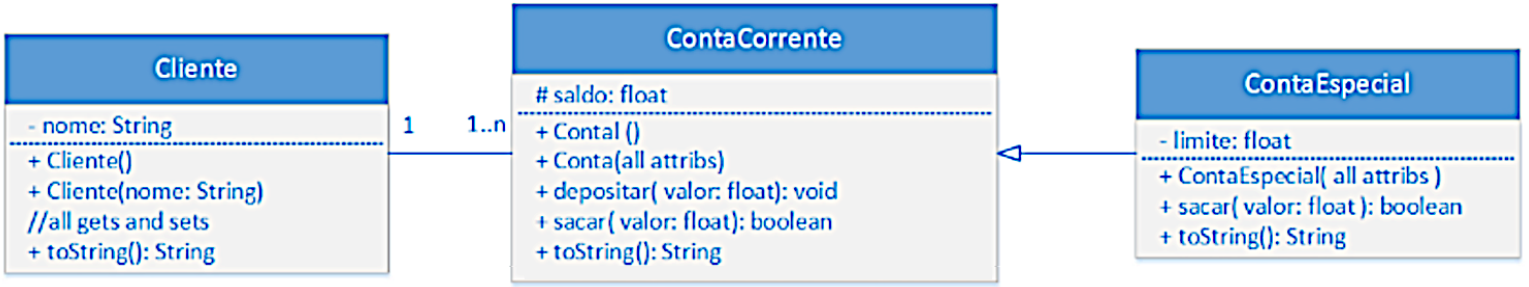
   e) Modifique agora a classe `Calculadora` para ela lançar a exceção personalizada criada no exercício, passando para ela a mensagem `"Erro de divisão por zero!"` e passando também o numerador `numero1` no construtor

   f) No tratamento de exceções do programa principal, use o método `getNumerador()` para ajudar a montar a frase abaixo, indicando exatamente qual divisão deu problema

```
Divisão 10 / 0 falhou
```

LISTA DE EXERCÍCIOS 11

3. Crie classes de forma a representar o diagrama a seguir



a) Clientes podem ter uma ou mais contas, implemente isso através de uma lista

b) Clientes de `ContaCorrente` não podem ter saldo negativo, o método `sacar` deve lançar uma exceção personalizada chamada `SaldoInsuficienteContaCorrenteException` caso o cliente tente sacar um valor superior ao saldo atual

c) Ao lançar a exceção deve-se passar para ela, como argumento, o valor que se tentou sacar

d) O construtor da classe `SaldoInsuficienteContaCorrenteException` deverá receber, portanto, como parâmetro, o valor que se tentou sacar `(double valor)`

e) Esse mesmo construtor deverá passar para a superclasse (usando o método `super()`), a mensagem `"Saldo insuficiente para sacar o valor de: " + valor`

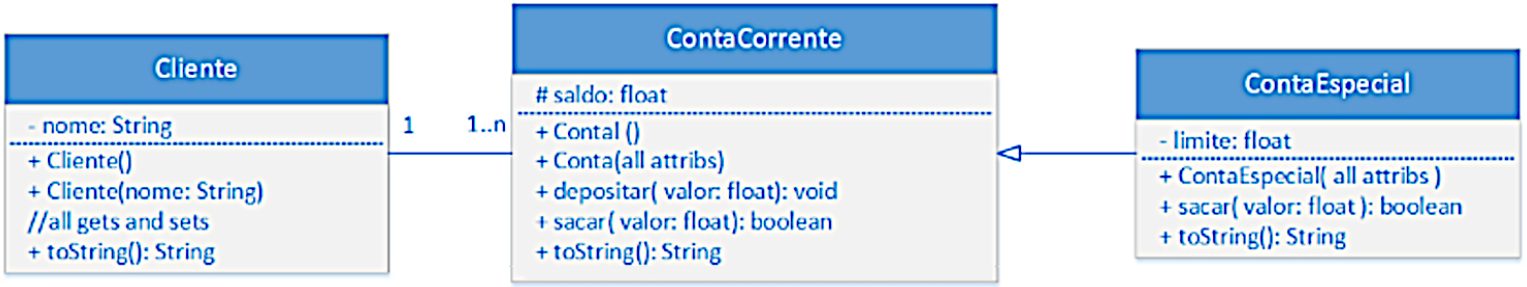
f) No programa principal crie um cliente que tenha uma conta corrente e tente sacar um valor maior do que o saldo – faça o tratamento da exceção com o bloco `try...catch`

UNOESC

Apoiadores: UNOESC CITEOSC INOVALE CENTRO DE INOVAÇÃO VIDEIRA DEV-TI SC TEC fapesc SANTA CATARINA

LISTA DE EXERCÍCIOS 11

3. Crie classes de forma a representar o diagrama a seguir



g) A classe `ContaEspecial` herda da classe `ContaCorrente` – clientes que possuem conta especial possuem um limite de crédito; dessa forma, podem fazer saques até esse valor limite, mesmo que não possuam saldo suficiente na conta

h) O construtor da classe `ContaEspecial` deve receber como parâmetro, além dos parâmetros da superclasse, o limite que o banco disponibiliza para o cliente

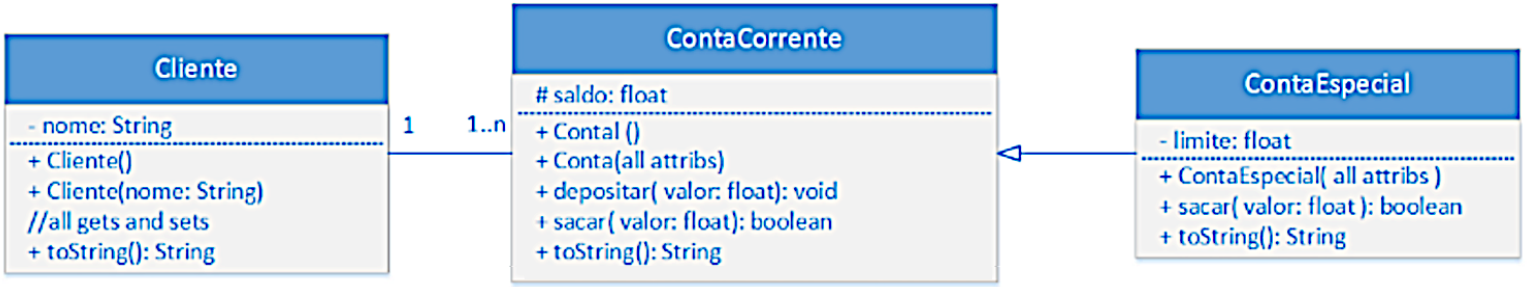
i) Sobrescreva o método `sacar` na classe `ContaEspecial`, de modo que o cliente possa ficar com saldo negativo até o valor de seu limite - note que o atributo saldo da classe `ContaCorrente` deve ser do tipo `protected` para que possa ser modificado na subclasse

j) Se o cliente especial tentar sacar além do limite, o programa deverá lançar uma exceção personalizada chamada `SaldoInsuficienteContaEspecialException`

UNOESC

Lista de Exercícios 11

3. Crie classes de forma a representar o diagrama a seguir



k) O construtor de `SaldoInsuficienteContaEspecialException` deve apresentar uma funcionalidade semelhante à da classe `SaldoInsuficienteContaCorrenteException`, ou seja, deverá receber, como parâmetro, o valor que se tentou sacar

l) Esse construtor deverá passar para a superclasse (usando o método `super()`), a mensagem `"Saldo além do limite para sacar o valor de: " + valor`

m) No programa principal acrescente um cliente que tenha uma conta especial e tente sacar um valor maior do que o limite – adicione este tratamento da exceção em mais um bloco `try...catch` ou use o recurso de *multicatch*

UNOESC