Compartilhar o seu link com: luciorocha@professores.utfpr.edu.br

Revisão de conteúdos. Exercícios propostos.

-Quizz (parte 1): https://quizizz.com/embed/quiz/625ebb3c7f80870020557253

-Quizz (parte 2): https://quizizz.com/admin/quiz/61000b97575f86001ceaa905

1) Implemente com Programação Orientada a Objetos:

```
public class Principal  {

public abstract class Conta {

       public abstract float getSaldo();

      //Todas as subclasses herdam da superclasse Conta
       public String toString(){
            return this.getClass().getSimpleName();
        }


}

public interface ICliente {

      public abstract String getCliente();

      public abstract String toString();

}


public class ContaCorrente extends Conta
                                  implements ICliente {
        private int a;
        public ContaCorrente(){

        }
        public ContaCorrente(String a ){
              this.a = Integer.parseInt( a );
        }
        public int getA() {
```

```
            return this.a;
        }
        public String getCliente(){
           return “NOME”;
        }
        public float getSaldo(){
            return 123.0f;
        }


}
public class ContaSalario extends Conta implements ICliente {
         private int b;
         public ContaSalario(){

         }
         public ContaSalario(String  b ){
              this.b = Integer.parseInt( b );
         }
         public int getB(){
               return this.b;
         }
        public String getCliente(){
           return “NOME”;
        }
       public float getSaldo(){
            return 123.0f;
        }


}
public class ContaPoupanca extends Conta implements ICliente {
      public int c;
      public ContaPoupanca(){

      }
      public ContaPoupanca( String  c ){

               this.c = Integer.parseInt( c );
      }
      public int getC(){
         return this.c;
      }
      public String getCliente(){
            return “NOME”;
      }
      public float getSaldo(){
            return 123.0f;
```

```
        }

}
```

```
        public Principal(){
             ArrayList<ICliente> lista = new ArrayList<>();
             ICliente cc = new ContaCorrente();
             lista.add( cc );

             cc = new ContaSalario();
             lista.add( cc );

             cc = new ContaPoupanca();
             lista.add( cc );

             for( ICliente  item : lista )
                 System.out.println(  item  );
        }

        public static void main(String [ ] args){
              Principal obj = new Principal();
       }

}
```

a) (1,0 ponto) Crie 3 (três) classes: ContaCorrente, ContaSalário e ContaPoupança. Crie 2 (dois) construtores sobrecarregados para cada classe.

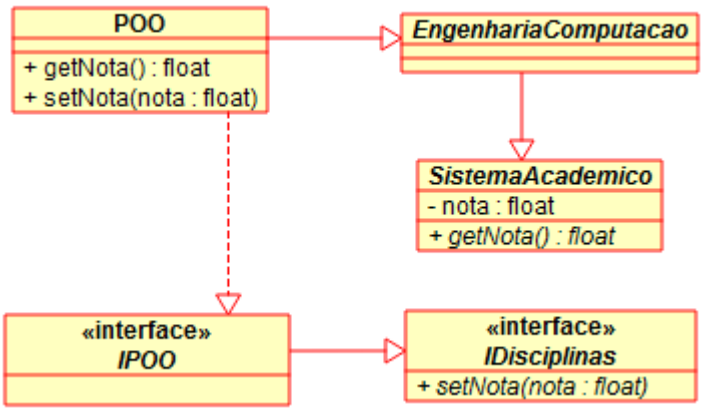
b) (1,0 ponto) Forneça para cada Classe 1 (um) atributo e 1 (um) comportamento únicos que não estão presentes nas outras classes. Inicialize os atributos em um dos construtores da Classe.

c) (1,0 ponto) Crie a Interface 'ICliente' com um método 'getCliente()' que retorna o nome do cliente. Cada Classe deve implementar a Interface 'ICliente'.

d) (1,0 ponto) Crie uma classe abstrata 'Conta' com um método abstrato 'getSaldo()' que retorna o saldo do cliente. A Classe Conta tem um relacionamento de Generalização com as Classes ContaCorrente, ContaSalário e ContaPoupança.

e) (1,0 ponto) Crie uma classe Principal com um vetor 'ICliente' que armazena objetos das Classes ContaCorrente, ContaSalario e ContaPoupanca. Itere polimorficamente o vetor imprimindo o conteúdo de cada objeto da seguinte forma: um objeto da Classe ContaCorrente imprime o nome da Classe e o valor do saque; um objeto da Classe ContaSalário imprime o nome da Classe, o valor e o CPF do cliente; um objeto da Classe ContaPoupança imprime o nome da Classe, o saldo e a data da consulta.

2) (5,0 pontos) No Diagrama UML da figura a seguir:



Nome	Tipo
SistemaAcademico	Classe Abstrata
EngenhariaComputacao	Classe Abstrata
POO	Classe Concreta

IPOO	Interface
IDisciplinas	Interface

a) (3,0 pontos) Implemente o Diagrama UML apresentado em linguagem de Programação Orientada a Objetos.

b) (2,0 pontos) Implemente a Classe Principal que instancia um objeto da Classe POO. A partir desse objeto da Classe POO, ilustre a invocação dos métodos acessores e mutadores para atribuir e recuperar a nota do Sistema Acadêmico, com tratamento de exceções.

Exercícios propostos (aulas anteriores):

1) Implemente o tratamento de exceções no trecho do cálculo no código a seguir:

```
import java.util.Scanner;

public class TratamentoExcecao1 {

	public static void main(String[] args) {
		Scanner obj = new Scanner(System.in);
		System.out.print("\nDigite o numerador: ");
		int numerador = obj.nextInt();
		System.out.print("\nDigite o denominador: ");
		int denominador = obj.nextInt();
		//Aritmetica de inteiros: nao eh permitida divisao por zero
		int resultado = numerador / denominador;
		//double resultado = (double) numerador / denominador;
		System.out.println("\nResultado: " + resultado);
	}
}
```

a) Crie um método de leitura de dados do usuário que capture a exceção não-verificada InputMismatchException. Corrija a entrada inválida.

```
//Filipe Augusto Parreira Almeida
import java.util.Scanner;

public class TratamentoExcecao1 {

	public static void main(String[] args) {
		Scanner obj = new Scanner(System.in);
		System.out.print("\nDigite o numerador: ");
		int numerador;
		try {
		      numerador = obj.nextInt();
		} catch( InputMismatchException e ){
		        e.printStackTrace();
		        numerador = 1;
		}

		System.out.print("\nDigite o denominador: ");
		int denominador = obj.nextInt();
		//Aritmetica de inteiros: nao eh permitida divisao por zero
		int resultado;
		try{
		    resultado = numerador / denominador;

		} catch (ArithmeticException e){
		    System.out.println(e.getMessage());
		}

		//double resultado = (double) numerador / denominador;
		System.out.println("\nResultado: " + resultado);
	}
}
```

b) Crie um método de leitura de dados do usuário que possa disparar uma exceção não-verificada ArithmeticException quando o denominador=0. Corrija a entrada inválida.

```
//Isabella Melo Almeida

import java.util.Scanner;

public class TratamentoExcecao1 {
	private int numerador;
	private int denominador;
	private float resultado;


	public void leitura() throws ArithmeticException {

		Scanner obj = new Scanner(System.in);
		System.out.print("\nDigite o numerador: ");

		try {
		      numerador = obj.nextInt();
		} catch( InputMismatchException e ){
		        e.printStackTrace();
		        numerador = 1;
		}

		System.out.print("\nDigite o denominador: ");
		denominador = obj.nextInt();

		if (denominador == 0)
		     throw new ArithmeticException();

		//Aritmetica de inteiros: nao eh permitida divisao por zero

		try{
		    resultado = numerador / denominador;

		} catch (ArithmeticException e){
		    System.out.println(e.getMessage());
	       }

		//double resultado = (double) numerador / denominador;
		System.out.println("\nResultado: " + resultado);
	}


	public TratamentoExcecao1(){
```

```
            try {
               leitura();
            } catch( ArithmeticException e ){
                    System.out.println("Passei por aqui");
                    denominador = 1;
            }

        }


         public static void main(String[] args) {
                  new TratamentoExcecao1();
         }
}
```

c) Crie um método de leitura de dados do usuário que possa disparar uma exceção não-verificada personalizada do tipo ArithmeticException quando o denominador=0. Corrija a entrada inválida.

```
//Joao Vitor
```

d) Crie um método de leitura de dados do usuário que possa disparar uma exceção verificada Exception quando o denominador=0. Corrija a entrada inválida.

```
//Lucas dos Reis
```

e) Crie um método de leitura de dados do usuário que possa disparar uma exceção verificada personalizada do tipo Exception quando o denominador=0. Corrija a entrada inválida.

```
//Matheus Hirata
import java.util.Scanner;
```

```
public class Excecao {

   int numerador = 0;
   int denominador = 0;
   float resultado = 0;

   public class MinhaExcecao extends Exception{

      public MinhaExcecao(){
         super();
      }

      public int corrigir(){
         return 1;
      }
   }

   public float leitura() throws MinhaExcecao{

      Scanner entrada = new Scanner(System.in);

      System.out.print("Numerador: ");
      numerador = entrada.nextInt();

      System.out.print("Denominador: ");
      denominador = entrada.nextInt();

      if( denominador == 0 )
         throw new MinhaExcecao();

      float resultado = numerador / denominador;

      entrada.close();

      return resultado;

   }

   public Excecao(){

      try{

         resultado = leitura();

      }catch(MinhaExcecao e){

         System.out.println("essessao");
```

```
            denominador = e.corrigir();
            resultado = numerador / denominador;
        }
    }

    public static void main(String[] args) {
        new Excecao();
    }
}
```

f) Ilustre um exemplo de captura seletiva das exceções: Exception, ArrayListOutOfBoundsException e FileNotFoundException.

```
//Rafael Kendy
```

2) Implemente o tratamento de exceções no trecho do cálculo no código a seguir:

```
import java.util.Scanner;

public class Principal {

  public void iniciar(){

      //Leitura de nome do usuario
      Scanner entrada = new Scanner( System.in );

           String nome = entrada.next();

      try {
          String CPF = entrada.next();
          //Suponha que CPF possua apenas numeros
          for( int i=0; i<CPF.length(); i++)
              if(  (int) CPF[i] != 0 && (int) CPF[i] != 1 &&
                   …
                   (int) CPF[i] != 9 )
                        throw new InputMismatchException();  //Disparou a excecao

      } catch( InputMismatchException  e ) { //Digitou um numero

      }

      //Leitura de CPF do usuario
```

```
      entrada.close();

  }
   public static void main(String [ ] args){
        Principal principal = new Principal();
        principal.iniciar();
  }
}
```

a) Crie um método de leitura de dados do usuário que capture a exceção não-verificada InputMismatchException. Corrija a entrada inválida.

b) Crie um método de leitura de dados do usuário que possa disparar uma exceção não-verificada ArithmeticException quando o CPF tiver mais do que 11 caracteres. Corrija a entrada inválida.

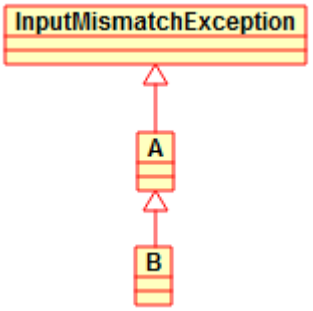
c) Crie um método de leitura de dados do usuário que possa disparar uma exceção não-verificada personalizada do tipo ArithmeticException quando o quando o CPF tiver mais do que 11 caracteres. Corrija a entrada inválida.

d) Crie um método de leitura de dados do usuário que possa disparar uma exceção verificada Exception quando o quando o CPF tiver mais do que 11 caracteres. Corrija a entrada inválida.

e) Crie um método de leitura de dados do usuário que possa disparar uma exceção verificada personalizada do tipo Exception quando o CPF tiver mais do que 11 caracteres. Corrija a entrada inválida.

f) Ilustre um exemplo de captura seletiva das exceções: Exception, InputMismatchException e NullPointerException.

3) Observe o diagrama UML a seguir:



a) Crie um método de leitura de dados do usuário que capture a exceção do tipo A.
b) Crie um método de leitura de dados do usuário que capture a exceção do tipo B.
c) Crie um método de leitura de dados do usuário que capture a exceção do tipo A. A seguir, dispare a exceção do tipo B e faça a captura.

4) Modifique o programa anterior para que a superclasse seja uma classe de Exceção verificada.