ânima EDUCAÇÃO

UNISUL
UNIVERSIDADE DO SUL DE SANTA CATARINA

PROGRAMAÇÃO DE SOLUÇÕES COMPUTACIONAIS

Prof. Ricardo Ribeiro Assink

POO – *Exception (ERROS)*

As **Exceptions** são provenientes de erros.

Todos os métodos de alguma classe são passíveis de causar algum erro.

Alguns exemplos:

- Acessar elemento ou método NULO.
- Dividir um número por zero.
- Tentar acessar uma posição inexistente de um Array.
- Atribuir tipos incompatíveis de dados em uma variável.
- Tentar conectar em um banco de dados que não existe.

POO – *Exception (ERROS)*

Simulando alguns erros veja:

```
//  -----  Simulando NullPointerException


    // Aluno aluno1 = new Aluno("Sistemas", 7); // gera NullPointerException em aluno1.getNome() logo abaixo.
    Aluno aluno1 = new Aluno("Sistemas", 7, 1111, "Tiburcio", 95);  // este é o construtor completo.
    String nome = aluno1.getNome();
    nome = nome.toUpperCase();
    System.out.println(nome);
```

Note que, SE usarmos o construtor incompleto da classe Aluno, o método *getNome()* retornará *NULL* para o campo nome. E campo nulo não pode invocar métodos como *toUpperCase().*

Erro gerado caso tivéssemos usado o construtor incompleto:

Exception in thread "main" java.lang.NullPointerException
at Principal.Principal.main(Principal.java:13)

POO – *Exception (ERROS)*

Simulando alguns erros veja:

```
// ----- Simulando ArithmeticException
    // Linha abaixo gera ArithmeticException por conta da divisão por zero.
    int n = 10 / 0;
```

- **Divisão por Zero é impossível.**

Erro gerado :

Exception in thread "main" java.lang.ArithmeticException: / by zero
at Principal.Principal.main(Principal.java:18)

POO – *Exception (ERROS)*

Simulando alguns erros veja:

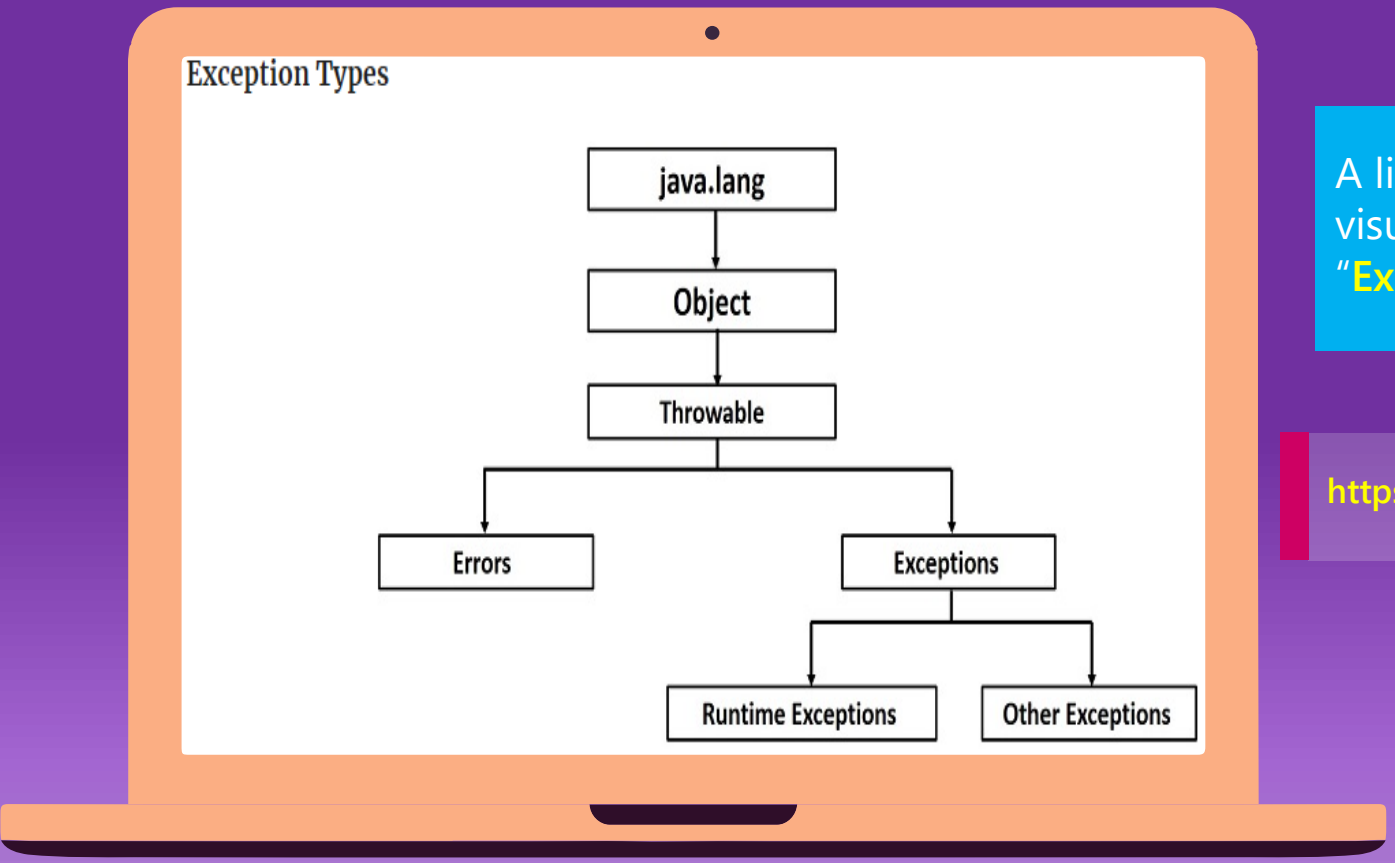
```
// ----- Simulando IndexOutOfBoundsException

    Aluno aluno2 = new Aluno("Massagem", 2, 2222, "Marilene", 21);
    Aluno mandarproDAO = new Aluno(); // instância vazia apenas para se comunicar com o DAO.
    mandarproDAO.InsertAlunoBD(aluno1);
    mandarproDAO.InsertAlunoBD(aluno2);

    System.out.println(mandarproDAO.getMinhaLista().get(0).toString());

    //  Linha abaixo gera IndexOutOfBoundsException, pois a posição 666 não existe.
    // System.out.println(mandarproDAO.getMinhaLista().get(666).toString());
```

Erro gerado :

Exception in thread "main" java.lang.IndexOutOfBoundsException: Index 666 out of bounds for length 2

POO – *Exception (ERROS)*

Apenas entendendo a estrutura, veja:



A lista completa das possibilidades do pacote ***java.lang*** pode ser visualizada na documentação do JAVA. Fique atento a lista "**Exception Summary**", que contem os exemplos da aula de hoje.

https://docs.oracle.com/javase/7/docs/api/java/lang/package-summary.html

Descrição de imagem:
Mostra a relação entre as Classes de Exceção dentro do pacote java.lang.

POO – *Try, Catch, Finally*

É possível implementar o tratamento dos códigos que podem lançar exceções (*Exception*). Para isso utilizamos os blocos *try, catch, finally*.

01 Usa-se o método de tentativa - o *try*.

02 Tudo que estiver dentro do bloco *try* será executado até que alguma exceção seja lançada.

03 No caso de alguma *Exception* lançada em *try*, o bloco *catch* é executado para tentar corrigir este erro ou apenas para capturá-lo.

04 Opcionalmente pode-se usar o bloco *finally*, que será executado **SEMPRE**, independente do resultado de *try*. Usado também para tratar o erro capturado.

POO – *Try, Catch, Finally*

Veja um exemplo de tratamento e funcionamento destes blocos:

```
//  -----  TRATANDO NullPointerException
      Aluno aluno1 = new Aluno("Sistemas", 7);  // gera NullPointerException em nome.toUpperCase() logo abaixo.
      // Aluno aluno1 = new Aluno("Sistemas", 7, 1111, "Tiburcio", 95); // NÃO GERARÁ ERRO, pois tem o campo nome.

      String nome = null;
      nome = aluno1.getNome(); // vai retornar null se construtor completo não for usado.

      try {
         // Aqui executamos as tentativas
         nome = nome.toUpperCase(); // objeto nulo não pode usar toUpperCase

      } catch (NullPointerException erro) {   // a Exception lançada NullPointerException será capturada na variável erro.
         // Se o try der errado, o catch é executado
         nome = "NULO";

      } finally {
         // executado, independente do resultado do try
         System.out.println(nome);
      }
```

POO – *throw e throws*

Algumas vezes **NÃO** queremos tratar os erros na mesma classe onde a ***Exception*** é lançada.
Nestes casos podemos **DESVIAR** o tratamento para outro lugar.

Para isso utilizamos os comandos ***throw*** e ***throws***.

Veja o exemplo a seguir.

POO – *throw e throws*

```
import Model.Aluno;
public class Principal3 {

    public static void maiusculas() throws Exception { // lança a exceção

        //  -----  TRATANDO NullPointerException deviando para o chamador


        //Aluno aluno1 = new Aluno("Sistemas", 7); // gera NullPointerException em nome.toUpperCase() logo abaixo.
        Aluno aluno1 = new Aluno("Sistemas", 7, 1111, "Tiburcio", 95); // NÃO GERARÁ ERRO, pois tem nome.

        String nome = null;
        nome = aluno1.getNome(); // vai retornar null se construtor completo não for usado.

        try {
            // Aqui executamos as tentativas
            nome = nome.toUpperCase(); // objeto nulo não pode usar toUpperCase

        } catch (NullPointerException erro) {
            // desvia o tratamento da exception para o try que chamou o método maiusculas();
            throw new Exception(erro);

        } finally {
            // executado, independente do resultado do try
            System.out.println(nome);
        }
    }
```

```
    public static void main(String[] args) {

        // o tratamento do erro em maiusculas() foi desviado para cá.
        try {
            maiusculas();
        } catch (Exception erro) {
            System.out.println("Estou tratando o erro aqui,
              caso em maiusculas() ocorra NullPointerException");
        }

    }
}
```

POO – CONCEITOS

Agora que já estudamos as principais estruturas básicas de controle dentro de um código em JAVA, já está na hora de conversarmos mais formalmente sobre os conceitos de **Encapsulamento** e **Polimorfismo**.

Com o conhecimento adquirido até aqui, já é possível relacionar a prática de programação com estes conceitos.

POO – ENCAPSULAMENTO

O **Encapsulamento** é um conceito da Programação Orientada a Objetos onde o estado de objetos (as variáveis da classe) e seus comportamentos (os métodos da classe) são agrupados em conjuntos segundo o seu grau de relação.

O **Encapsulamento** mantem atributos e métodos **SEGUROS** de interferências externas e de utilizações erradas por outros integrantes deste mesmo sistema.

O **Encapsulamento** está diretamente ligado ao conceito de *ocultação de dados*, tornando as informações privadas apenas a quem as possui.

Vamos conversar ?
Você consegue identificar as estruturas que você já utiliza, que estão ligados a este conceito ?
Conta aí para nós !!

POO – POLIMORFISMO

O **Polimorfismo** vem de MUITAS FORMAS, tanto para métodos como para objetos.

Polimorfismo é um princípio a partir do qual as classes derivadas de uma única classe base são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

Podemos dizer também que o **Polimorfismo** implementa a seleção correta dos métodos em tempo de execução. Exemplo *Override*.

Para utilizarmos Polimorfismo não é obrigatório o uso de herança, porém, os conceitos estão diretamente ligados.

Vamos conversar ?
Você consegue identificar as estruturas que você já utiliza, que estão ligados a este conceito ?
Conta aí para nós !!

EXERCÍCIO 37

Use o código abaixo e faça o tratamento de erro adequado para divisões por zero (try, catch, finally):

```
public class Exercicio37 {

    public static void main(String[] args) {
        int n1 = Integer.parseInt(JOptionPane.showInputDialog("Digite um número:"));
        int n2 = Integer.parseInt(JOptionPane.showInputDialog("Digite outro número:"));
        int n = 0;

        n = n1 / n2;       // se for uma divisão por zero vai dar erro.

    }
}
```

EXERCÍCIO 38

Use o código abaixo e faça o tratamento de erro adequado para acesso a índice inexistente (try, catch, finally):

```
public class Exercicio38 {

    public static void main(String[] args) {

        int numeros[] = new int[10];

        numeros[0] = 0;
        numeros[1] = 1;
        numeros[2] = 2;
        numeros[3] = 3;
        numeros[4] = 4;
        numeros[5] = 5;
        numeros[6] = 6;
        numeros[7] = 7;
        numeros[8] = 8;
        numeros[9] = 9;

        System.out.println("tentando mostrar índice 10:" + numeros[10]); //vai dar erro, não existe índice 10.

    }
}
```

EXERCÍCIO 39

Explique com suas próprias palavras o que entendeu sobre encapsulamento e polimorfismo.