Profº Josivaldo de França

Exceções

- **Uma exceção é qualquer condição de erro ou comportamento inesperado encontrado por um programa em execução**

As exceções ocorrem quando algo imprevisto acontece, elas podem ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis.

Exceções Por motivos externos...

Tentar abrir um arquivo que não existe.
Tentar fazer consulta a um banco de dados que não está disponível.
Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita.
Tentar conectar em servidor inexistente.

Exceções erros de lógica...

Tentar manipular um objeto que está com o valor nulo.
Dividir um número por zero.
Tentar manipular um tipo de dado como se fosse outro.
Tentar utilizar um método ou classe não existentes.

Exceções exemplo

*Teste.java

```
1  package teste;
2
3  import java.util.Scanner;
4
5  public class Teste {
6
7      public static void main(String[] args) {
8          // TODO Auto-generated method stub
9          System.out.print("Digite um número : ");
10         Scanner sc = new Scanner(System.in);
11         int x = sc.nextInt();
12         System.out.print("Digite seu divisor: ");
13         int y = sc.nextInt();
14         System.out.print("O resultado é: "+x/y );
15
16         sc.close();
17
18     }
19
20 }
```

Exceções exemplo

Nulo.java

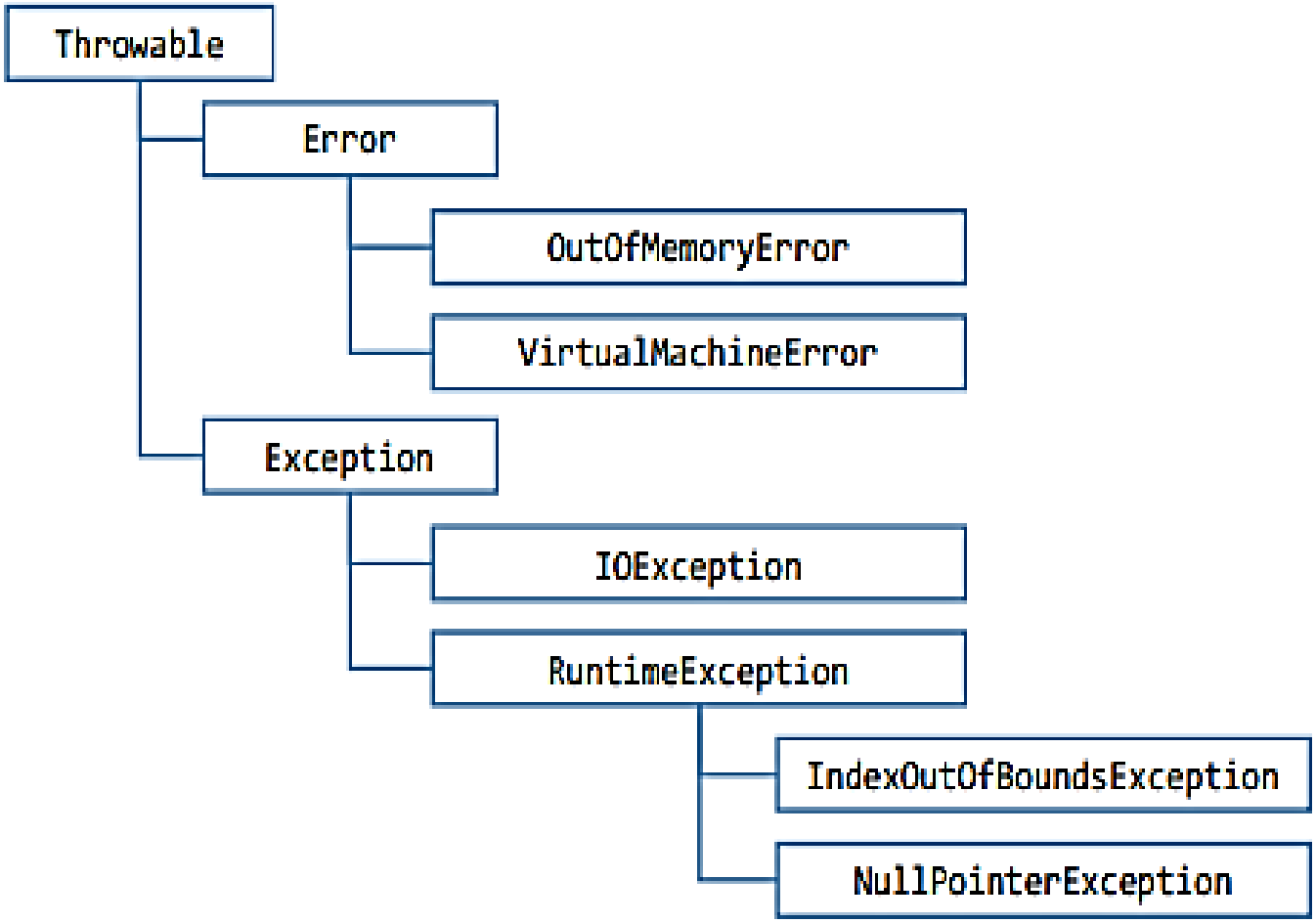
```
1  package teste;
2
3  public class Nulo {
4
5      public static void main(String args[])
6        {
7          String frase = null;
8          String novaFrase = null;
9          novaFrase = frase.toUpperCase();
10         System.out.println("Frase antiga: "+frase);
11         System.out.println("Frase nova: "+novaFrase);
12       }
13     }
14
```

Exceções

Quando lançada, uma exceção é propagada na pilha de chamadas de métodos em execução, até que seja capturada (tratada) ou o programa seja encerrado

Hierarquia de exceções do Java

https://docs.oracle.com/javase/10/docs/api/java/lang/package-tree.html



Por que tratar exceções?

- O modelo de tratamento de exceções permite que erros sejam tratados de forma consistente e flexível, usando boas práticas

• Vantagens:

- Delega a lógica do erro para a classe responsável por conhecer as regras que podem ocasionar o erro;
- Trata de forma organizada (inclusive hierárquica) exceções de tipos diferentes;
- A exceção pode carregar dados quaisquer.

Estrutura try-catch

- **Bloco try**
- Contém o código que representa a execução normal do trecho de código que pode acarretar em uma exceção

- **Bloco catch**
- Contém o código a ser executado caso uma exceção ocorra
- Deve ser especificado o tipo da exceção a ser tratada (upcasting é permitido)

try-catch - Exemplo

Sintaxe

```
try {
}
catch (ExceptionType e) {
}
catch (ExceptionType e) {
}
catch (ExceptionType e) {
}
```

try-catch - Exemplo

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.print("Digite um número : ");
    Scanner sc = new Scanner(System.in);
    int x = sc.nextInt();
    System.out.print("Digite seu divisor: ");
    int y = sc.nextInt();
    try {
        System.out.print("O resultado é: " + x/y );
    }
    catch (ArithmeticException e) {
        System.out.println("Jumento,não existe divisão por zero.");
    }
    sc.close();

}
```

try-catch - Exemplo

*Teste.java

```
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class Teste {
7
8      public static void main(String[] args) {
9          Scanner sc = new Scanner(System.in);
10
11         int x = 0;
12         int y = 0;
13         boolean valoresValidos = false;
14
15         while (!valoresValidos) {
16             try {
17                 System.out.print("Digite um número: ");
18                 x = sc.nextInt();
19                 System.out.print("Digite o divisor: ");
20                 y = sc.nextInt();
21
22                 valoresValidos = true;
23
24             } catch (InputMismatchException e) {
25                 System.out.println("Valor inválido. Tente novamente.");
26                 sc.nextLine(); // limpa o buffer de entrada do Scanner
27             }
28         }
29
30         try {
31             System.out.println("O resultado é: " + x / y);
32         } catch (ArithmeticException e) {
33             System.out.println("Erro: divisão por zero.");
34         }
35
36         sc.close();
```

try-catch

É usado em ações que sempre precisam ser executadas independente se gerar erro. Um exemplo é o fechamento da conexão de um banco de dados.

Praticamente, o uso dos blocos try/catch se dá em métodos que envolvem alguma manipulação de dados, bem como:

- **CRUD no banco de dados;**
- **Índices fora do intervalo de array;**
- **Cálculos matemáticos;**
- **I/O de dados;**
- **Erros de rede;**
- **Anulação de objetos;**
- **Entre outros;**

Bloco Finally

O bloco finally é utilizado para garantir que um código seja executado após um try, mesmo que uma exceção tenha sido gerada. Mesmo que tenha um return no try ou no catch, o bloco finally é sempre executado.

Bloco Finally

O bloco **finally**, geralmente, é utilizado para fechar conexões, arquivos e liberar recursos utilizados dentro do bloco **try/catch**. Como ele é sempre executado, nós conseguimos garantir que sempre após um recurso ser utilizado dentro do **try/catch** ele poderá ser fechado/liberado no **finally**.

```
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.print("Digite um número : ");
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        System.out.print("Digite seu divisor: ");
        int y = sc.nextInt();
        try {
            System.out.print("O resultado é: " + x/y );
        }
        catch (ArithmeticException e) {
            System.out.println("Jumento,não existe divisão por zero.");
        }
        finally {
            System.out.println("Finalizando sistema");
        }
        sc.close();

    }

}
```

Hora de codar

1-O B.uni, banco da Uninassau quer que você faça um programa para ler os dados de uma conta bancária e depois realizar um saque nesta conta bancária, mostrando o novo saldo.

Um saque não pode ocorrer se não houver saldo na conta, ou se o valor do saque for superior ao limite de saque da conta.

2-Crie uma classe Carro com os atributos marca, modelo e ano, e um método getInformacoes() que retorne uma string com as informações do carro. Crie um objeto dessa classe e chame o método getInformacoes().

3-Crie uma classe Banco com os atributos nome e cnpj, e um método abrirConta(String nomeTitular, double saldoInicial) que cria uma conta bancária com o titular e o saldo inicial especificados, e retorna o número da conta. Caso o saldo inicial seja negativo, lance uma exceção SaldoInicialInvalidoException. Crie um objeto dessa classe e teste o método abrirConta().

4-Crie uma classe Animal com os atributos nome e especie, e um método emitirSom() que exibe o som que o animal faz. Crie duas subclasses de Animal: Cachorro e Gato, cada uma com seu próprio som. Crie um método brincar(Animal animal) que exibe uma mensagem de brincadeira entre o animal que chama o método e o animal passado como parâmetro. Crie um objeto de cada uma das subclasses e chame o método brincar().

5-Crie uma classe Retangulo com os atributos base e altura, e os métodos getBase() e getAltura() que retornem os respectivos atributos. Adicione um método calcularArea() que calcula e retorna a área do retângulo. No construtor da classe, lance uma exceção DimensaoInvalidaException caso a base ou a altura seja negativa. Crie um objeto dessa classe e teste o método calcularArea().


Language
Pascal
ALGOL
Assembly
Object
Not
E
Lisp
language
shell
Enviar para Josivaldo@gmx.com
PEARL
Oxygene
Script
PHP
Ruby
Programming
REXX
Common
Python
Java
Prolog
Java


Language
Pascal
ALGOL
Assembly
C
shell
Not
Object
E
language
Lisp
VB
JavaScript
C++
ABC
Fancy
Cayenne
Go
Delphi
DASL
Z
Prolog
PEARL
Oxygene
Programming
Java
Ruby
Script
PHP
REXX
Common
Python
Strand
Eiffel
Coral

Java


Language
Pascal
ALGOL
Assembly
C
shell
Not
Object
E
language
Lisp
VB
JavaScript
C++
ABC
Fancy
Cayenne
Go
Delphi
DASL
Z
Prolog
PEARL
Oxygene
Programming
Java
Ruby
Script
PHP
REXX
Common
Python
Strand
Eiffel
Coral

Java


Language
Pascal
ALGOL
Assembly
C
shell
Not
Object
E
language
Lisp
VB
JavaScript
C++
ABC
Fancy
Cayenne
Go
Delphi
DASL
Z
Prolog
PEARL
Oxygene
Programming
Java
Ruby
Script
PHP
REXX
Common
Python
Strand
Eiffel
Coral

Java

Bibliografia

https://www.udemy.com/course/java-curso-completo/learn/lecture/10793838#overview