TRATAMENTO DE EXCEÇÕES EM JAVA

Dra. Alana Morais

Roteiro da Aula

1. O que é uma exceção?
2. Try, catch e finally
3. Diferenciar entre o uso de throw e throws
4. Utilizar as classes de exceções existentes
5. Diferenciar entre exceções verificadas e não verificadas
6. Definir suas próprias classes de exceção;

O que é uma exceção?

- O tratamento de exceção é o mecanismo responsável pelo tratamento da ocorrência de **condições que alteram o fluxo normal da execução** de programas de computadores.

O que é uma exceção?

- O tratamento de exceção é o mecanismo responsável pelo tratamento da ocorrência de **condições que alteram o fluxo normal da execução** de programas de computadores.

- O estado do programa é gravado em um local pré-definido e a sua execução é direcionada para uma rotina de tratamento.

O que é uma exceção?

- Acontece quando encontra algo inesperado:

Problemas no *hardware*	Arrays fora de faixa	Valores de variáveis	Erro de entrada e saída (IO)	Erros da aplicação
Valores de variáveis	Saldo insuficiente	Divisão por zero	Parâmetros de métodos	Falha de Memória
Usuário não existe	Nota invalida			

O que é uma exceção?

- Dependendo da situação, a rotina de tratamento pode prosseguir a execução a partir do ponto que originou a exceção, utilizando a informação gravada para restaurar o estado.
  - Um exemplo de exceção que permite o prosseguimento da execução é aquela originada por uma operação aritmética.

O que é uma exceção?

- Para o desenvolvedor de uma rotina, lançar uma exceção é um modo útil de assinalar que a rotina não deve continuar a execução.

- Exemplos:
  - argumentos de entrada não são válidos (um denominador igual a zero em uma divisão, por exemplo)
  - um recurso no qual o programa depende não está disponível (um arquivo não encontrado ou um erro em um disco, por exemplo).

O que é uma exceção?

Eventos excepcionais

Erros que ocorrem durante a execução do programa

Causam distúrbios no fluxo normal do programa

Tratamento de Exceções

- Sempre que um método de alguma classe é passível de causar algum erro, então, podemos usar o método de tentativa - o *try*.

- Tudo que estiver dentro do bloco *try* será executado até que alguma exceção seja lançada, ou seja, até que algo dê errado.

Tratamento de Exceções

- Quando uma exceção é lançada, ela sempre deve ser capturada.
- O trabalho de captura da exceção é executado pelo bloco *catch*.
- Um bloco *try* pode possuir vários blocos de *catch*, dependendo do número de exceções que podem ser lançadas por uma classe ou método.

Tratamento de Exceções

- Em Java:

try, catch e finally

- Define um bloco de tratamento de exceção.

throws

- Declara que um método pode lançar uma exceção ou mais exceções.

throw

- Lança uma exceção.

Try, catch e finally

- Instruções try-catch
  - Sintaxe:

```
try {
    <código a ser monitorado para exceções>
} catch (<ClasseExceção> <nomeObj>) {
    <tratar se ClasseExceção1 ocorrer>
}
```

Try, catch e finally

- Exemplo:

```
package exemploexcecao1;
import java.util.Scanner;
public class ExemploExcecao1 {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Digite seu nome: ");
		String nome = sc.next();
		System.out.println("Digite sua idade: ");

		try{
			int idade = sc.nextInt();
		} catch( Exception e){
			System.out.println(“Você informou a idade errada!”);
		}
	}
}
```

Palavra-chave finally

- Finally é o trecho de código final. A função básica de finally é sempre executar seu bloco de dados mesmo que uma exceção seja lançada.

- É muito útil para liberar recursos do sistema quando utilizamos, por exemplo, conexões de banco de dados e abertura de buffer para leitura ou escrita de arquivos.

- Finally virá após os blocos de catch.

Palavra-chave finally

- Sintaxe:

```
try {
    <código a ser monitorado para exceções>
} catch (<ClasseExceção1> <nomeObj>) {
    <tratar se ClasseExceção1 ocorrer>
} ...
} finally {
    <código a ser executado antes do bloco try ser finalizado>
}
```

Palavra-chave finally

- Exemplo

```
try {
	objQualquer.WriteLine("Escrevendo algo...");
} catch (Exeption e){
	System.out.println("Erro ao digitar...");
} finally {
	objQualquer.Close();
}
```

Palavra-chave finally

- Bloco de código é executado em um desses quatro cenários:

1. Execução normal sem exceções;

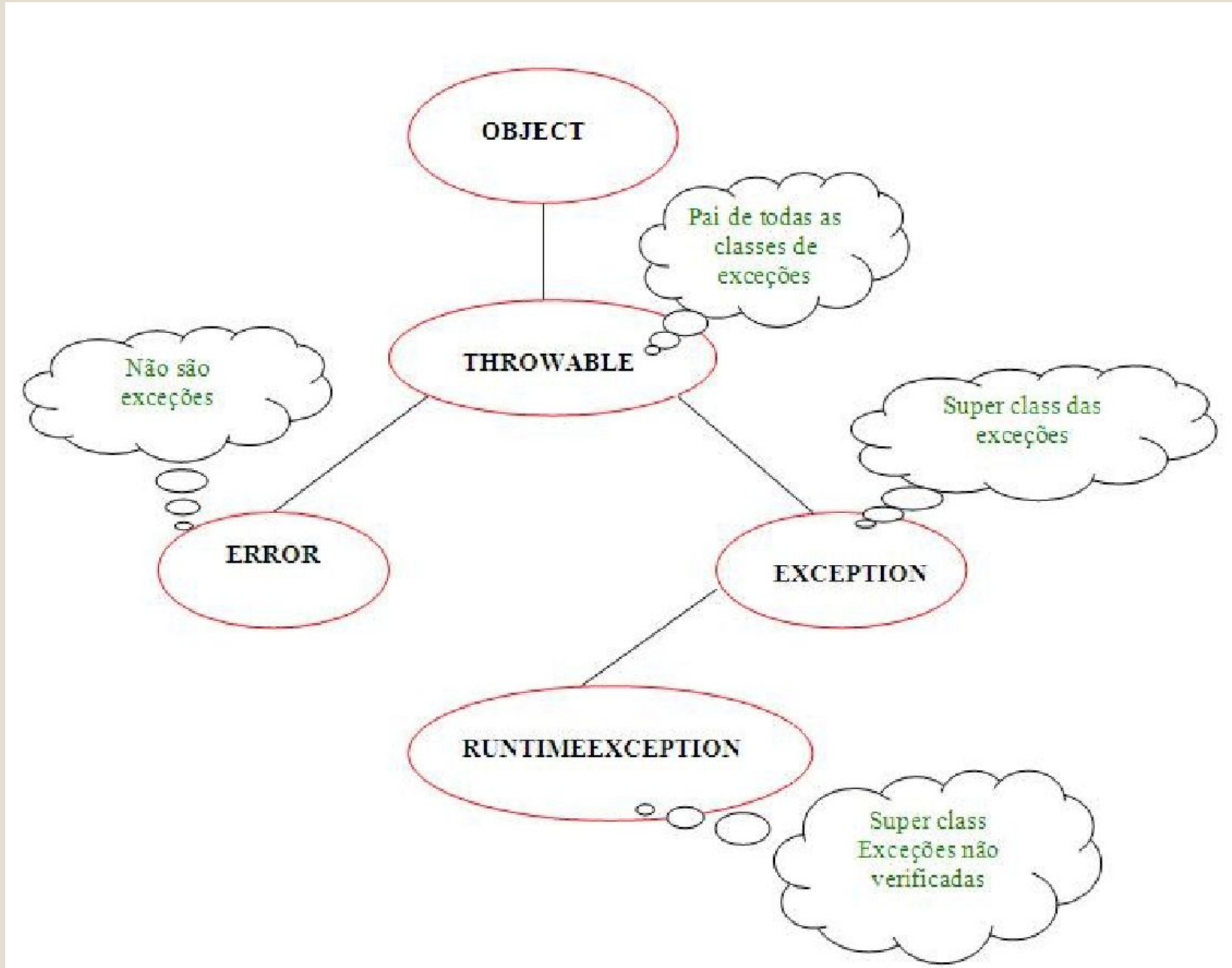
2. Um bloco catch captura e trata uma exceção lançada;

3. Exceção lançada não é capturada por nenhum bloco catch;

4. Saída forçada usando instruções return, continue ou break.

4. Saída forçada usando instruções return, continue ou break

```
Scanner sc = new Scanner(System.in);
try {
    System.out.println("Digite sua idade: ");
    int idade = sc.nextInt();
    return;
} catch (ArithmeticException e) {
   System.out.println("Você informou a idade errada!");
} finally {
   System.out.println("Executei!");
}
```


OBJECT
Pai de todas as classes de exceções
THROWABLE
Não são exceções
ERROR
Super class das exceções
EXCEPTION
RUNTIMEEXCEPTION
Super class Exceções não verificadas

Classe Throwable

Classe de origem de todas as classes de exceção

Sub-classe Error

- Usada pela JVM para manipular erros ocorridos no ambiente de execução
- Geralmente, esses erros encontram-se além do controle dos programas
- Exemplos:
  - Erro de falta de memória
  - Erro de acesso ao disco rígido

Sub-classe Exception

- Circunstâncias que os usuários podem tratar
- Resultam de falhas no código do programa
- Exemplos:
  - Erro em uma divisão por zero
  - Erro ao tentar acessar um índice inexistente um array

Hierarquia das classes de Exceções

Throwable	Error	LinkageError, …	
		VirtualMachineError, …	
	Exception	ClassNotFoundException	
		CloneNotSupportedException	
		IlegalAccessExcption	
		InstatiationException	
		InterruptedException	
		IOException	EOFException
			FileNotFoundException
		RuntimeException	ArithmeticException
			ArrayStoreException
			ClassCastException
			…

Ordem de captura

Exemplo não compila

```
try {

} catch (Exception e){

} catch (ArithmeticException e){

}
```

Exemplo que compila

```
try {

} catch (ArithmeticException e){

} catch (Exception e){

}
```

Lançando de exceções: throws

- É necessário um método para cada catch ou lista de exceções que podem ser lançadas
- Exceto para as Classes Error, RuntimeException e suas sub-classes
- Se um método causar uma exceção mas não capturá-la, então deve-se utilizar a palavra-chave throws
- Sintaxe:

```
<tipo> <nomeDoMétodo> (<argumento>*) throws <listaDeExceção>
{
    <Corpo do Método>
}
```

Exemplo 1

```
public static void funcao1(){
    try {
        funcao2();
    } catch (Exception e){
        ...
    }
}
public static void funcao2() throws Exception {
   ...
}
```

Exemplo 2

```
public static void main(String[] args) {
    try {
        funcao1();
    } catch (Exception e){
        ...
    }
}
public static void funcao1() throws Exception {
   funcao2();
}
public static void funcao2() throws Exception {
   ...
}
```

Exemplo 3

```
public static void main(String[] args) throws Exeption {
        funcao1();
}

public static void funcao1() throws Exception {
   funcao2();
}

public static void funcao2() throws Exception {
   ...
}
```

Lançando de exceções: throws

- Java nos permite lançar exceções:
  - throw <objetoExceção>;
- Exemplo: throw new ArithmeticException("testing...");

```
public static void funcao1(){
        try {
                funcao2();
        } catch (Exception e){
                ...
        }
}
public static void funcao2() throws Exception {
        throw new Exception("Deu erro!");
}
```

Tipos de Exceções

Exceções não-verificadas

- As exceções não-verificadas são aquelas que quando um método pode lança-la, não necessita-se mostrar em seu cabeçalho;
- Caso o lançamento conste no cabeçalho, esta exceção não precisa é obrigatória ser capturada com try/catch.
- Todas as exceções derivadas de RuntimeException são não verificadas.

Exceções verificadas

- As exceções verificadas (checked exceptions) são aquelas que se um método pode lançá-la, ela deve constar em seu cabeçalho.
- Todas as exceções que derivam de Exception de forma direta ou indireta, são exceções verificadas.

Tipos de Exceções Vantagens e Desvantagens

Exceções não-verificadas

- Vantagem: torna a escrita do código mais fácil de escrita e leitura;
- Desvantagem: ao acontecer uma exceção o programa é imediatamente finalizado.

Exceções verificadas

- Vantagem: fazem do código mais seguro já que não é permitido ignorar exceções conhecidas e dá oportunidade de recuperar de problemas;
- Desvantagem: obriga a inserir try/catch ao longo do código

Exemplo

- Implementar a exceção ExcecaoDivisaoPorZero.
- Esta exceção será lançada pelo método double calcula(double a, double b) da classe Divisao.
- Implementar também a aplicação AplicacaoDivisao que recebe dois números informados pelo usuário e faz a divisão do primeiro pelo segundo usando uma chamada ao método calcula da classe Divisao.

Exercício (1)

- Crie uma Classe CalculoMatematico
  - Nela, crie um método divisão, que recebe como parâmetros os valores a serem divididos.
  - O retorno é o resultado da divisão (todos os números devem ser do tipo inteiro)
  - Crie uma classe de teste para testar a CalculoMatematico
  - Nela crie um objeto CalculoMatematico e acesse o método divisao, tentando dividir 4 por 0.
  - Execute a classe e veja o que acontece

Exercício (2)

- Crie um bloco try...catch no método divisão para tratar a operação realizada
- No catch:
  - Informar o objeto do tipo ArithmeticException
- Imprimir uma mensagem informando que a operação não pode ser realizada
- Retorna zero

Exercício (3)

- Tire o bloco try...catch do método divisao
- Adicione throws ArithmeticException na assinatura do método
- E na primeira linha do bloco do método, faça uma verificação se o divisor é igual a 0
  - Se for, lance uma exceção
  - throw new ArithmeticException("Texto");
- Na classe de teste, crie um bloco try...catch, tentando executar o método divisão – Catch para ArithmeticException
- No bloco do Catch, imprima o método getMessage() do objeto criado do tipo ArithmeticException

Exercício (4)

- Crie uma nova Classe
  - DivisorZeroException
  - A classe deve herder a classe Exception
- Na Classe CalculoMatematico, troque
  - ArithmeticException por DivisorZeroException
  - throws DivisorZeroException – throw new DivisorZeroException();
- Na classe de teste, troque no Catch
  - ArithmeticException por DivisorZeroException

DÚVIDAS?

alanamm.prof@gmail.com