Tratamento de Exceções

Prof. Bruno Gomes

bruno.gomes@ifrn.edu.br

Programação Orientada a Objetos

Exceções

- Aplicações, durante a execução, podem incorrer em muitas espécies de erros de vários graus de severidade
- Quando métodos são invocados sobre um objeto:
  - Problemas de estado interno
  - Erros com objetos ou dados que eles manipulam
  - Ele pode estar violando seu contrato básico

Exceções

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

Exceções

- É desagradável encontrar erros
- O que deve ser feito:
  - Notificar o usuário de um erro;
  - Conseguir salvar todo o trabalho
  - Permitir que usuários saiam elegantemente do programa

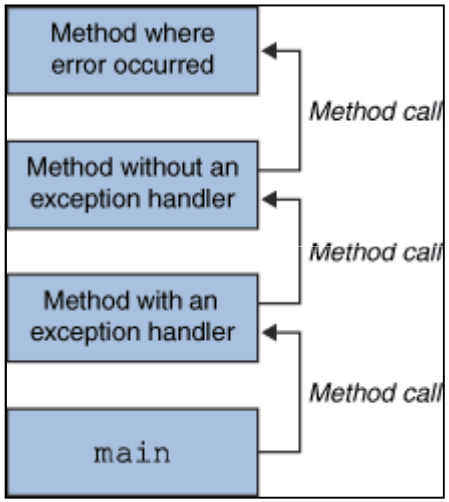
Exceções

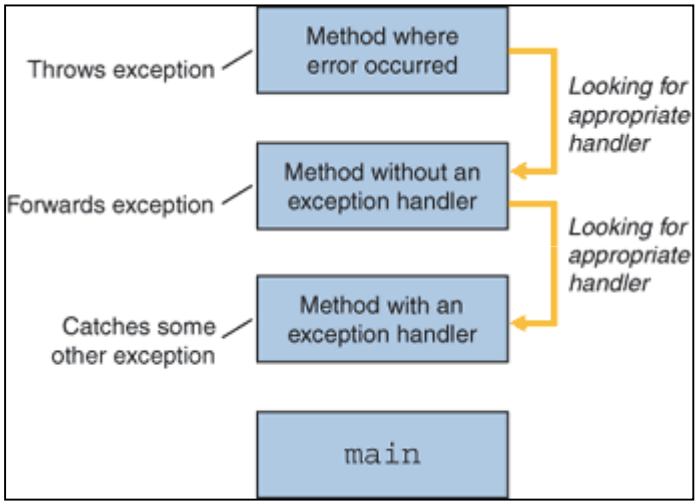
- Exceção é um desvio no fluxo de execução normal do programa
- Indica que houve problema na execução de um bloco do programa
- Se não for tratado, programa pode parar
- O uso correto de exceções torna o programa mais robusto e confiável

Exceções

- Quando um erro ocorre dentro de um método:
  - Um objeto é criado
  - O objeto contém informações sobre o erro, assim como o tipo e o estado do programa
  - A ação de capturar esse objeto chamamos de *throwing an exception*
  - O ambiente de execução (*runtime system*) tenta encontrar algum tratamento a exceção
  - A busca segue a *call stack* – lista de chamadas de métodos

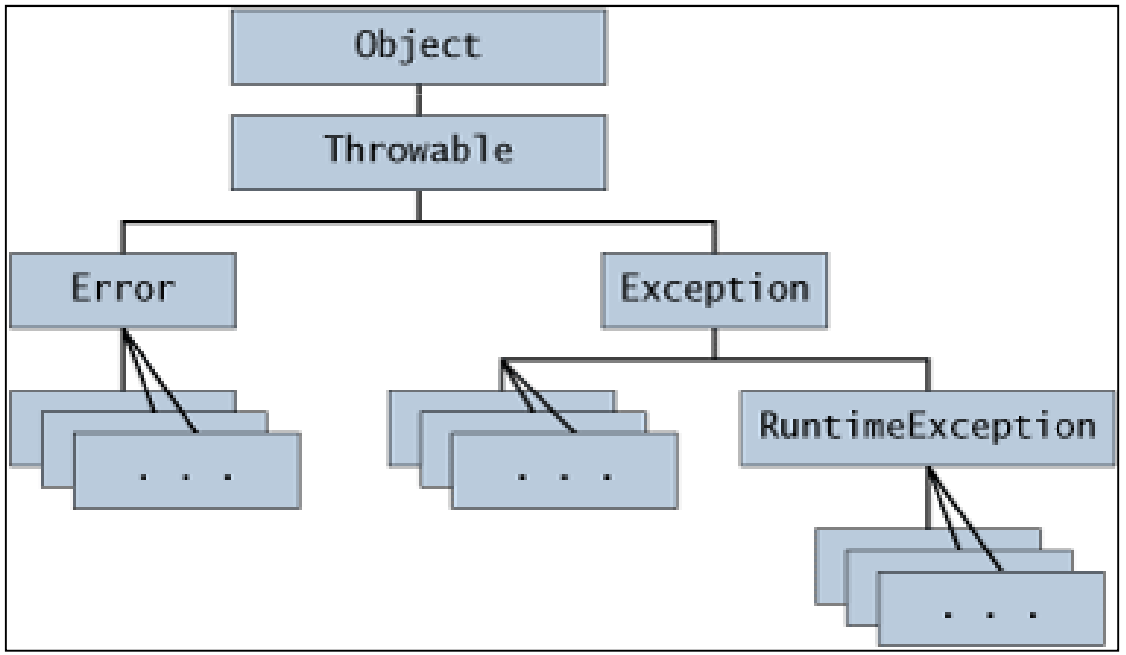
Exceções





Hierarquia de Exceções em Java

- Em Java, um objeto de exceção é sempre uma instância de uma classe derivada *Throwable*



Hierarquia de Exceções em Java

- *Error:*
  - Ocorrem devido a problemas de SO ou hardware
  - Não deve lançar um objeto desse tipo

- Ao se fazer um programa Java, deve focalizar na hierarquia *Exception*
  - Divide em dois ramos: As que derivam de *RuntimeException* e as que não derivam

Hierarquia de Exceções em Java

- *RuntimeException*:
  - Acontece porque se fez um erro de programação
  - Exemplos:
    - Acesso de array proibido; acesso de ponteiro nulo.
- Qualquer outra Exceção:
  - Ocorre porque algo ruim, como um erro de E/S, aconteceu com o programa bom em outros aspectos
  - Exemplos:
    - Tentar ler além do fim de um arquivo; tentar abrir um URL malformado.

Tipos de Exceções

- A *Java Language Specification* define 2 tipos de exceções:
- **Exceção Não Verificada (*unchecked*)**:
  - Deriva da classe *Error* ou da *RuntimeException*
  - Não precisam ser tratadas, mas podem ser
    - NullPointerException, NumberFormatException, ArrayIndexOutOfBoundsException
  - Se uma exceção não verificada ocorrer e não for tratada, o programa pode parar (console)
- **Exceção Verificada (*checked*):**
  - Exceções previsíveis
  - Devem ser tratadas pelo programa

Formas de Capturar

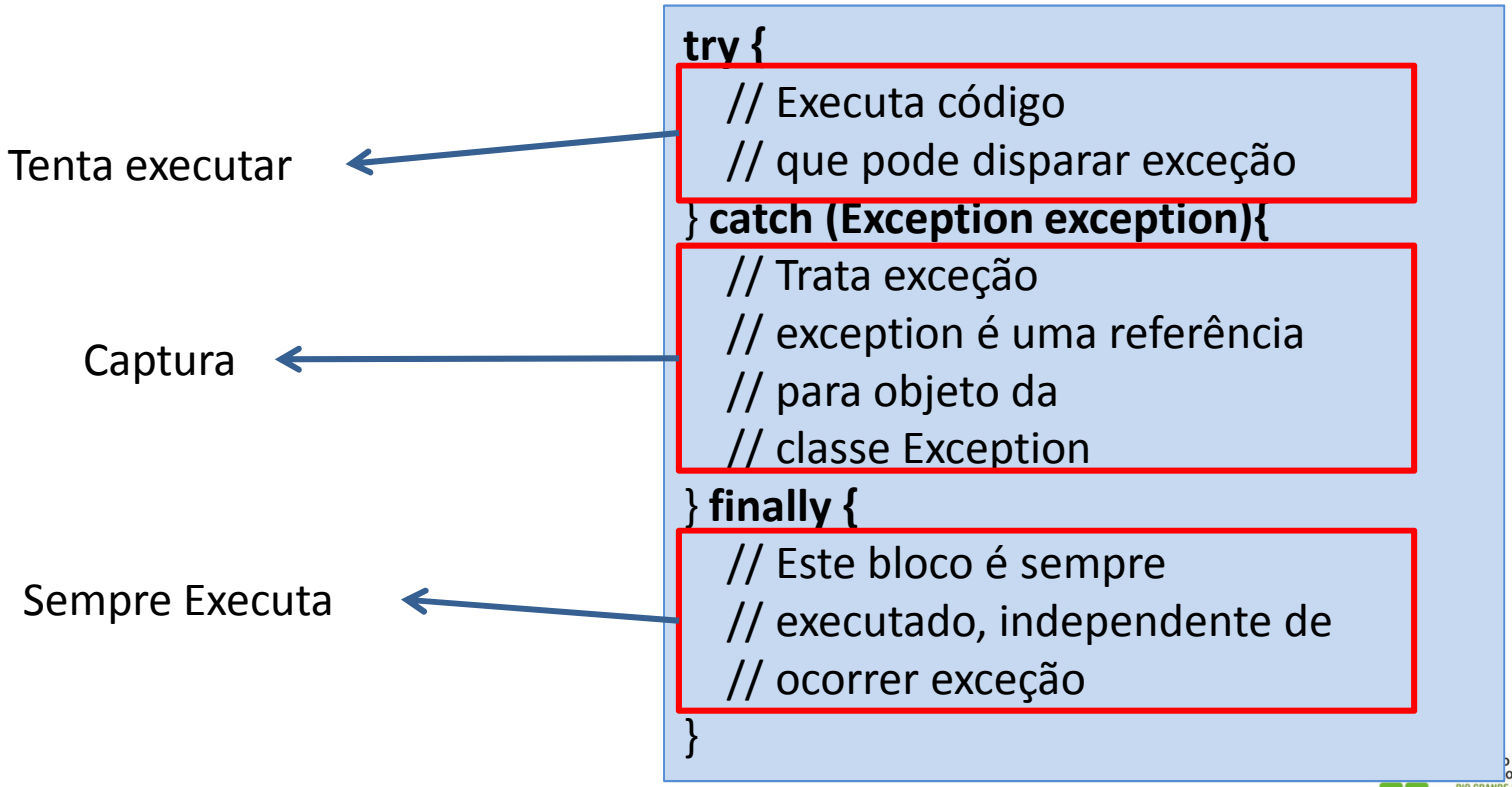
- Através de uma cláusula *throws* na assinatura do método
- Ou através de uma bloco *try*

Formas de Capturar

- Um método deve informar que exceções ele pode disparar (throw)
  - cláusula throws na definição do método
- Um bloco que tenta (try) chamar um método que pode disparar uma exceção deve tratá-la
  - Chamada normal de um método, mas que deve estar em um bloco try{...} catch {...}
- Uma exceção é um objeto que deve ser capturado (catch)
  - É nesse bloco que a exceção deve ser tratada
- Um trecho de código pode ser executado sempre
  - bloco finally

Tratando Exceções

- 1ª Forma: bloco try-catch-finally
  - Um bloco deve capturar uma exceção para tratá-la



Exemplo

- Método parseInt pode disparar exceção NumberFormatException (não verificada)
  - Se a exceção for dispara os comandos do bloco try não serão mais executados.
  - O fluxo de execução muda para a captura (bloco catch)
  - Ela pode não ser tratada.
    - Programa encerra se for disparada

```
System.out.print("Digite um número");
String numero = sc.nextLine();
int x;
   try{
      x = Integer.parseInt(numero);
      System.out.println("Numero digitado é válido");
   } catch (NumberFormatException exception){
      System.out.println("Digite um número válido");
   }
```

Catch

- Pode haver mais de um bloco catch
  - Cada bloco trata um tipo específico de exceção
  - O bloco try contém métodos que podem disparar todas as exceções
  - Um método pode disparar mais de uma exceção

```
try {
   // bloco de comandos
} catch (Excecao1 ex1) {
   // Trata exceção1
} catch (Excecao2 ex2) {
   // Trata exceção2
} catch (Excecao3 ex3){
   // Trata exceção3
}
```

Repassando a responsabilidade

- Um método pode repassar uma exceção
  - Ele chama um método que dispara uma exceção, mas não quer tratar
    - Ele pode repassar a exceção
- Basta colocar a cláusula throws na assinatura do método
- Para quem chama, é o método que dispara a exceção

Repassando a Responsabilidade

```
public void metodo() throws AlgumaException{
  // Corpo do método chama
  // método que dispara AlgumaException
  obj.metodoQueDisparaExcecao();
}
```

```
public void qualquer(){
  // alguma coisa
  try {
    metodo();
  } catch (AlgumaException e) {
    e.printStackTrace();
  }
    // mais coisa
}
```

Definindo Exceções

- **1º Passo:** Defina uma classe que herde de Exception
  - Ou *RuntimeException* se desejar fazer exceção não verificada

```
public class ContatoNaoEncontradoException extends Exception {
        public ContatoNaoEncontradoException()
        {
                super("Contato não encontrado");
        }
}
```

Definindo Exceções

- **2º Passo:** No método que dispara a exceção:
  - Coloque a cláusua *throws*
  - Crie o objeto da classe de exceção
  - Dispare (*throw*) a exceção
    - Onde a exceção será disparada depende de cada método

```
public Contato buscar(String nome)
  throws ContatoNaoEncontradoException{
    // Laco para procurar contato pelo nome
    for (int i = 0 ; i < quantidade ; i++)
        if (contatos[i].nome().equals(nome))
    return contatos[i];
    // Se sair do laço e não tiver retornado
    // o contato não esta cadastrado
    // deve disparar excecao
    throw new ContatoNaoEncontradoException();
}
```

Definindo Exceções

- **3º Passo:** Exceção é tratada na interface com usuário

```
private void buscarContato() {
    System.out.print("Digite o nome: ");
    String nome = sc.nextLine();
    try {
        Contato contato = agenda.buscar(nome);
        System.out.println(contato);
    } catch (ContatoNaoEncontradoException e) {
        System.out.println("ERRO!!!!");
        System.out.println(e.getMessage());
        System.out.println("\nDigite [ENTER] para continuar....");
        sc.nextLine();
}
}
```

Exercício

- Crie uma Classe **CalculoMatematico**
  - Nela, crie um método **divisao**, 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

- Crie um bloco **try...catch** no metodo divisao 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

- 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

- Crie uma nova Classe
  - DivisorZeroException
  - Implemente da mesma forma do slide 19
- Na Classe CalculoMatematico, troque **ArithmeticException** por **DivisorZeroException**
  - throws DivisorZeroException
  - throw new DivisorZeroException();
- Na classe de teste, troque no Catch **ArithmeticException** por **DivisorZeroException**