Algoritmos e Programação II

Prof.: Paulo César Melo

paulomelo@inf.ufg.br

Aula 6 – Tratamento de Exceções

Objetivos

- Controlar erros e tomar decisões baseadas nos mesmos
- Criar novos tipos de erros para melhorar o tratamento deles
- Assegurar que um método funcione

Roteiro

- Introdução
- Tratar exceções pelo uso de try, catch e Finally
- Diferenciar throw e throws
- Usar classes de exceções existentes
- Exceções verificadas e não verificadas
- Exemplos

O que é uma **Exceção**?

Exceção

- **É um evento que interrompe o fluxo normal de processamento de uma classe;**

- É um erro ou exceção de algum tipo
  - Eventos excepcionais
  - Erros que ocorrem durante a execução do programa;

- Causa o término inesperado da classe
  - Distúrbios no fluxo normal do programa

Exceção

- **Em geral, na ocorrência de uma exceção, o estado do programa é gravado em um local pré-definido e a sua execução é direcionada para uma rotina de tratamento**

Exceção

Motivação

- O que aconteceria ao tentar chamar o método sacar com um valor fora do limite?

Mostrará uma mensagem de erro...

Problema:

Quem chamou o método sacar não saberá que isso aconteceu!

Como avisar quem chamou o método que ele não conseguiu realizar a operação?

2 implementações

Tratar no método

Tratar em quem chamou

Exceção

Exemplo

- Criar uma conta
- Depositar R$ 200,00
- Definir limite como R$ 200,00
- Sacar R$ 1000,00

Há alguma inconsistência nesse exemplo?

Agora se pensarmos no método transferir... Quais os possíveis erros que precisaríamos tratar?

Saldo insuficiente

Conta inválida

Como implementar esse retorno?

Retornar um inteiro como código de erro.

Para compreender melhor

```java
public class TesteErro {

    public static void main(String[] args){
        System.out.println("inicio do main");
        metodo1();
        System.out.println("fim do main");
    }

    public static void metodo1(){
        System.out.println("inicio do metodo1");
        metodo2();
        System.out.println("Fim do metodo1");
    }

    public static void metodo2(){
        System.out.println("inicio do metodo2");
        int[] vetor new = int[10];
        for(int i=0; i<=15; i++){
            vetor[i]=i;
            System.out.println(i);
        }
        System.out.println("fim do metodo2");
    }
```

Exceções precisam ser tratadas

1. Tentar fazer algo potencialmente perigoso;
2. Se der certo, continuar;
3. Senão, tratar o erro ou repassá-lo a quem me chamou.

Roteiro

- Introdução
- Tratar exceções pelo uso de try, catch e Finally
- Diferenciar throw e throws
- Usar classes de exceções existentes
- Exceções verificadas e não verificadas
- Exemplos

Try...catch...finally

Try...catch...finally

Try

- Tentar fazer um conjunto de ações

Catch

- Captura um tipo de exceção específica e toma as medidas cabíveis;
- Um catch para cada possível exceção;
- Deve-se respeitar a hierarquia de exceções;

Finally

- Realizado após o try-catch, finalizando a tarefa
- Dando exceção ou não, vai ser chamado

Try...catch...finally

Sintaxe

```
try{
    //escreva as instruções passíveis de gerar
    uma exceção neste bloco
}catch(<exceptionType> <varName>){
    //escreva a ação que o seu programa fará caso
    ocorra uma exceção de um determinado tipo
}finally{
    //escreva a ação que o seu programa executará
    caso ocorra ou não um erro ou exceção
}
```

Try...catch...finally

Sintaxe

```
try{
    //escreva as instruções passíveis de gerar
    uma exceção neste bloco
}catch(<exceptionType1> <varName1>){
    //escreva a ação que o seu programa fará caso
    ocorra uma exceção de um determinado tipo
}catch(<exceptionType2> <varName2>){
    //escreva a ação que o seu programa fará caso
    ocorra uma exceção de um determinado tipo
}finally{
    //escreva a ação que o seu programa executará
    caso ocorra ou não um erro ou exceção
}
```

Try...catch...finally

Exemplo 1

```
public static void main(String[] args){
    String entrada = JOptionPane.showInputDialog(
    "Entre com um número");

    try{
        int num = Integer.parseInt(entrada);
    }catch(NumberFormatException e){
        Sytem.out.println("Você não entrou com o
        número certo");
    }
}
```

Try...catch...finally

Exemplo 2

```
...
        boolean erro;
        do {
            try {
                System.out.println("Digite sua idade: ");
                int idade = sc.nextInt();
                erro = false;
            } catch (Exception e){
                System.out.println("Você informou a idade
errada!");
                erro = true;
                sc.nextLine();
            }
        } while (erro);
...
```

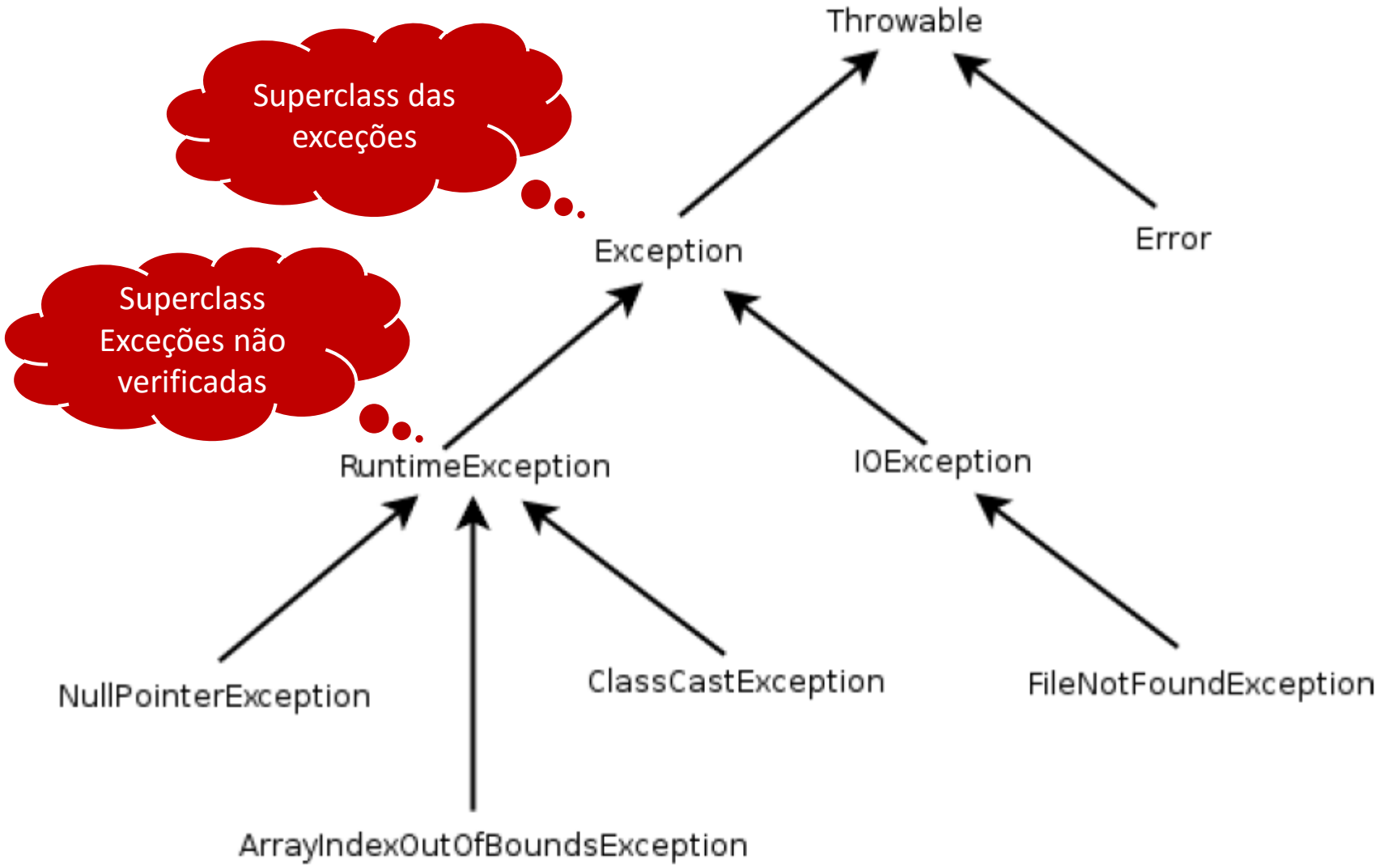
Try...catch...finally

Exemplo 2

Retomando o código do Slide 12

1. Adicione um try/catch em volta do for, pegando *ArrayIndexOutOfBoundsException.*
2. Em vez de fazer o try em torno do for inteiro, adicione somente com o bloco de dentro do for.
3. Coloque-o em volta da chamada do metodo2

Classe Throwable


Throwable
Superclass das exceções
Exception
Error
Superclass Exceções não verificadas
RuntimeException
IOException
NullPointerException
ClassCastException
FileNotFoundException
ArrayIndexOutOfBoundsException

Métodos

- toString():
  - Converte os dados da exceção para String para visualização
- printStackTrace():
  - Imprime na saída de erro padrão todos os pontos de erros
    - Útil para depuração

Métodos

- getCause():
  - Retorna a causa da Exceção, ou null se a causa for desconhecida ou não existir
- getMessage():
  - Retorna uma String com o erro. É uma forma simples e elegante de mostrar a exceção causada.

Ordem de Captura

```
try {

} catch (Exception e){

} catch (ArithmeticException e){

}
```

Ordem de Captura

```
try {

} catch (ArithmeticException e){

} catch (Exception e){


}
```

Exeções

Verificadas x Não verificadas

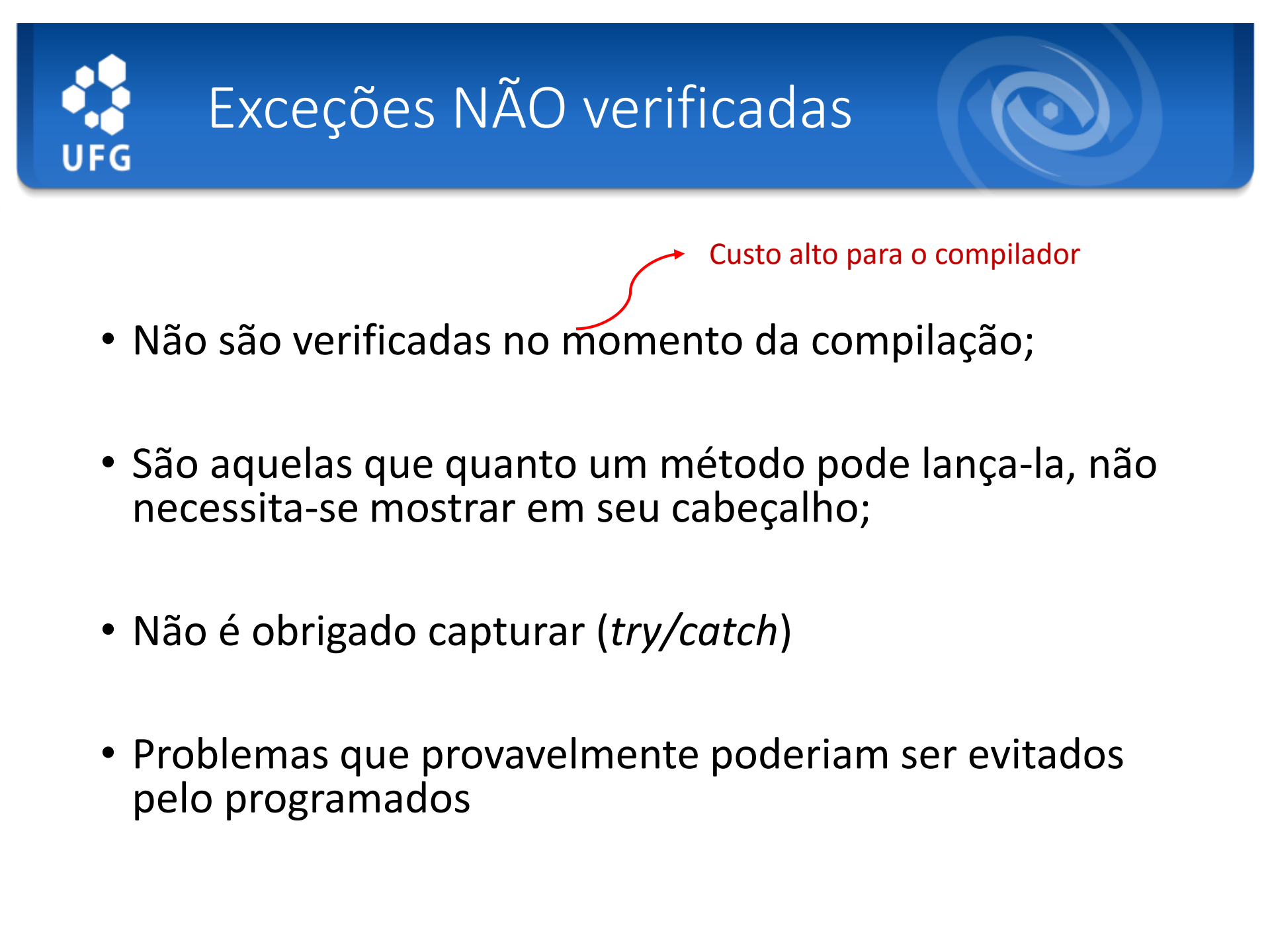
Exceções verificadas

- São aquelas que se um método pode lança-la, ela deve constar em seu cabeçalho.
- O compilador Java verifica se cada catch ou lista de exceções encontram-se dentro da cláusula *throws*
- Senão, ocorrerá um erro de compilação

Exceções verificadas

- Todas as classes que herdam da classe *Exception* mas não da classe *RuntimeException*
- Devem ser capturadas ou especificadas
- Diz-se que exceções são lançadas (*thrown*) dentro de um método. Isso inclui suas exceções e todas as outras possíveis de serem lançadas pelos métodos que ele chama

Exceções NÃO verificadas



- Não são verificadas no momento da compilação;
- São aquelas que quanto um método pode lança-la, não necessita-se mostrar em seu cabeçalho;
- Não é obrigado capturar (*try/catch*)
- Problemas que provavelmente poderiam ser evitados pelo programados

Exceções NÃO verificadas

- Classes de exceção não-verificadas são as exceções:
  - Error
  - RuntimeException e suas sub-classes.

Exemplo

```
public static void funcao1(){

      funcao2();

}

public static void funcao2()  {

    throw new RuntimeException("Deu erro!");

}
```

Exemplo

```
public static void funcao1(){

      funcao2();

}
public static void funcao2()
  throws RuntimeException {

    throw new RuntimeException("Deu erro!");

}
```

Vantagens e Desvantagens

- Exceções verificadas:
  - Vantagem:
    - Fazem do código mais seguro já que não é permitido ignorar exceções conhecidas
  - Desvantagem
    - Obriga a inserir try/catch
- Exceções não-verificadas
  - Vantagem
    - Torna a escrita do código mais fácil
  - Desvantagem:
    - Ao acontecer uma exceção o programa é imediatamente finalizado

Exceções mais comuns

```
public class TestandoADivisao {

    public static void main(String args[]) {
        int i = 5571;
        i = i / 0;
        System.out.println("O resultado  " + i);
    }
}
```

ArithmeticException

```java
public class TestandoReferenciaNula {
    public static void main(String args[]) {
        Conta c = null;
        System.out.println("Saldo atual " + c.getSaldo());
    }
}
```

NullPointerException

Lançando exceções

Lançando exceções

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

Lançando exceções

Sintaxe

```
<tipo> <nomeMetodo> (<argumento>*) throws <listaExceção> {
        ...
}
```

ou

```
metodo(){
        if( ){
                throw new Exception();
        }
}
```

Exemplo: sacar
Exceção: IllegalArgumentException

O que colocar dentro do try?

Isso influencia em que?

Criando sua própria Exceção

```
public class SemLetraAException extends Exception {
    public String toString(){
      return "Não existe letra A em sua frase";
    }
}
public class ExemploExcecao{
    public void imprima(String frase)
      throws SemLetraAException{

        if(!frase.contains("a") || !frase.contains("A"))
          throw new SemLetraAException();
        }
        System.out.println(frase);
    }
}
```

```
public class Exemplo{
    public static void main(String args){
        ExemploExcecao obj = new ExemploExcecao();
        try{
            obj.imprima("Tem erro!");
        } catch (SemLetraAException e){
            System.out.println(e);
        }
    }
}
```