

Asserções



$i \geq 0$



$i \leq \text{value.length}$



Asserções

Introdução



- Asserções são usadas para garantir que uma determinada condição é verdadeira em uma determinada parte do programa
 - É uma expressão que contém e verifica uma condição booleana
 - Servem para ajudar o programador a garantir a validade de algumas expressões durante o desenvolvimento do programa
 - E encontrar potenciais bugs ou erros de lógica em tempo de execução
- Asserções só devem ser usadas durante o desenvolvimento do sistema, e nunca em um sistema em produção (que está sendo usado pelo cliente)



- Existem duas formas de assertões
 - Assertão simples

```
// Sintaxe: assert <expressão1>;  
assert i >= 0;
```

- Assertão com mensagem

```
// Sintaxe: assert <expressão1> : <expressão2>;  
assert i >= 0 : "i deve ser positivo";
```

Assertões Exemplo



```
public class Livro {  
    private String titulo;  
    private int anoPublicacao;  
  
    // getTitulo, setTitulo ...  
  
    public int getAnoPublicacao() {  
        assert anoPublicacao > 0 : "ano de publicação é negativo";  
        return anoPublicacao;  
    }  
  
    public void setAnoPublicacao(int anoPublicacao) {  
        if (anoPublicacao > 0)  
            this.anoPublicacao = anoPublicacao;  
        else  
            this.anoPublicacao = 0;  
        assert anoPublicacao > 0;  
    }  
}
```

Pela lógica do método, essa assertão nunca seria verdadeira. Esse é um dos objetivos das assertões!

Assertões

Usando as Assertões



- Por padrão, as assertões são desabilitadas na execução
 - Para habilitá-las, usa-se a opção `-ea`

```
public class TesteAssercao {  
    public static void main(String args[]) {  
        int anoAtual = -2077;  
        assert anoAtual > 0 : "Ano atual negativo";  
        System.out.println("Ano Atual: " + anoAtual);  
    }  
}
```

```
$ javac TesteAssercao
```

```
$ java TesteAssercao  
Ano Atual: -2077
```

```
$ java -ea TesteAssercao  
Exception in thread "main" java.lang.AssertionError: Ano atual negativo  
    at TesteAssercao.main(TesteAssercao.java:8)
```



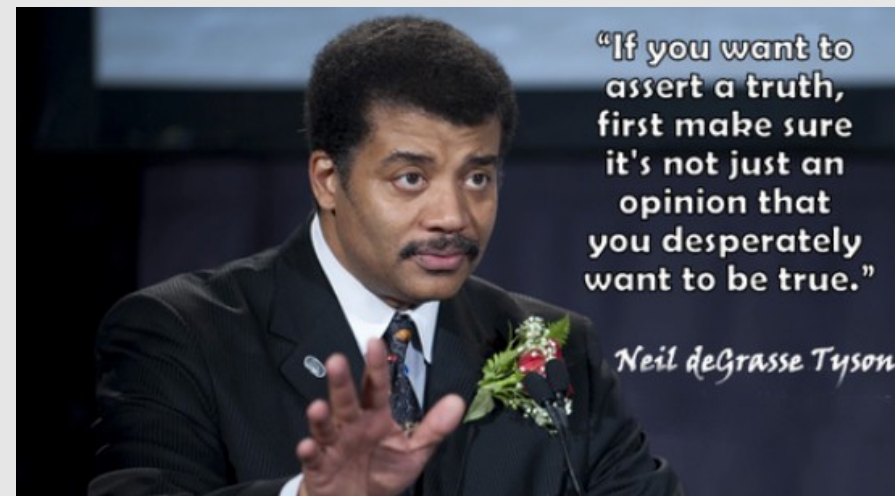
- Asserções são normalmente utilizadas em três casos
 - Pré-condições
 - *Usado no início dos métodos, para garantir que condições iniciais para a execução do método sejam satisfeitas*
 - Condições intermediárias
 - *Usado no meio dos métodos para garantir que até o momento o método está se comportando conforme esperado*
 - Pós-condições
 - *Usado no final dos métodos para garantir que as alterações realizadas pelo método são válidas*

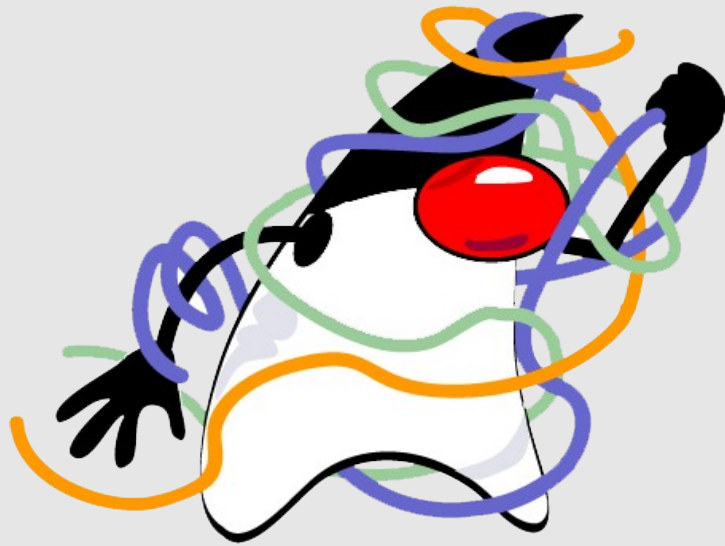
Assertões

Considerações Finais



- Normalmente, assertões são habilitadas durante o desenvolvimento e testes, e desabilitadas para distribuição, o que melhora a performance
- Como assertões podem ser desabilitadas (e -são-, por padrão), os programas não podem assumir que as verificações serão executadas.
 - A execução da assertão não deve alterar o estado do programa
 - Por isso, assertões não são normalmente utilizadas para verificar a validade de argumentos passados para os métodos, uma vez que essa validade deve ser garantida sempre



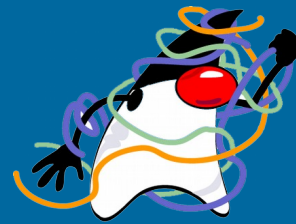


Tratamento de Exceções



Tratamento de Exceções

Introdução



- Uma *exceção* é um sinal de erro que o Java emite quando um programa viola restrições de semântica da linguagem
 - Ao contrário dos erros de sintaxe, que acontecem em tempo de compilação
 - Os erros de semântica (exceções) acontecem em tempo de execução
- Nomenclatura
 - Quando uma exceção ocorre, dizemos que uma exceção foi “*disparada*”
 - Em seguida, essa exceção pode ser “*capturada*”
 - Uma vez capturada, ela pode ser “*tratada*”

Tratamento de Exceções

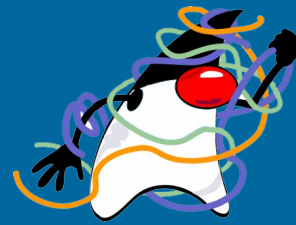
Introdução



- Um método seu pode também disparar uma exceção
 - Isso permite uma forma alternativa ao antigo método de retornar valores inválidos em funções para indicar erro. Exemplos:
 - *Uma função que retorna um inteiro retornar um número negativo quando algum erro foi encontrado na função*
 - *Uma função que retorna uma estrutura retornar null quando algum erro acontece*
- Exemplos:
 - Divisão por zero
 - Acesso a um elemento fora dos limites de um vetor
 - *Nestes casos, muitas linguagens simplesmente fecham o programa*
 - *Java, por outro lado, indica o acontecimento de uma exceção e permite o tratamento da mesma*

Tratamento de Exceções

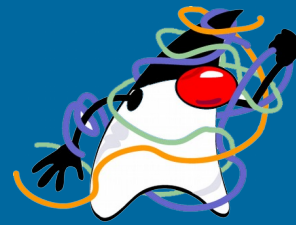
Exceções no Java



- Exceções são implementadas como uma classe
 - Quando uma exceção é disparada, um objeto daquela classe é criado
 - Essas classes são conhecidas como *classes de exceção*
 - *Elas são uma subclasse da classe Throwable*
- O Java possui uma série de exceções da própria linguagem
 - `StringIndexOutOfBoundsException`
 - `ArrayIndexOutOfBoundsException`
 - `NullPointerException`
 - `ArithmeticException`, etc
- Outras exceções podem ser criadas pelo programador
 - Isso normalmente é feito criando uma classe que herda a classe `Exception`

Tratamento de Exceções

Exemplo de Exceção I



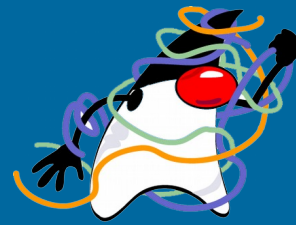
- `StringIndexOutOfBoundsException`
 - Disparado quando tentamos acessar um caractere de uma string que está além dos limites dela

```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        String teste = "War. War never changes.";  
  
        char c30 = teste.charAt(30);  
        System.out.println(c30);  
  
    }  
}
```

```
$ java TesteExcecao  
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:  
String index out of range: 30  
    at java.lang.String.charAt(String.java:658)  
    at TesteExcecao.main(TesteExcecao.java:10)
```

Tratamento de Exceções

Exemplo de Exceção II



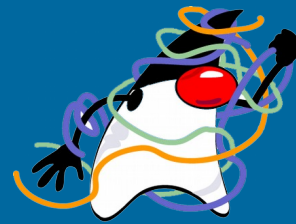
- `ArrayIndexOutOfBoundsException`
 - Disparado quando tentamos acessar um elemento de um vetor fora dos limites

```
public class TesteExcecao {  
    public static void main(String args[]) {  
        int vetor[] = new int[3];  
        vetor[3] = 204863;  
    }  
}
```

```
$ java TesteExcecao  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
    at TesteExcecao.main(TesteExcecao.java:6)
```

Tratamento de Exceções

Exemplo de Exceção III



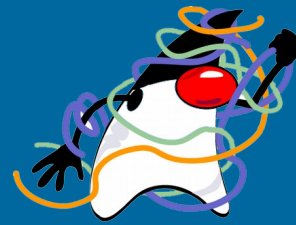
- `ArithmeticException`
 - Disparado quando há um erro aritmético (e.g., divisão por zero, número além dos limites do tipo)

```
public class TesteExcecao {  
    public static void main(String args[]) {  
        float i = 3 / 0;  
    }  
}
```

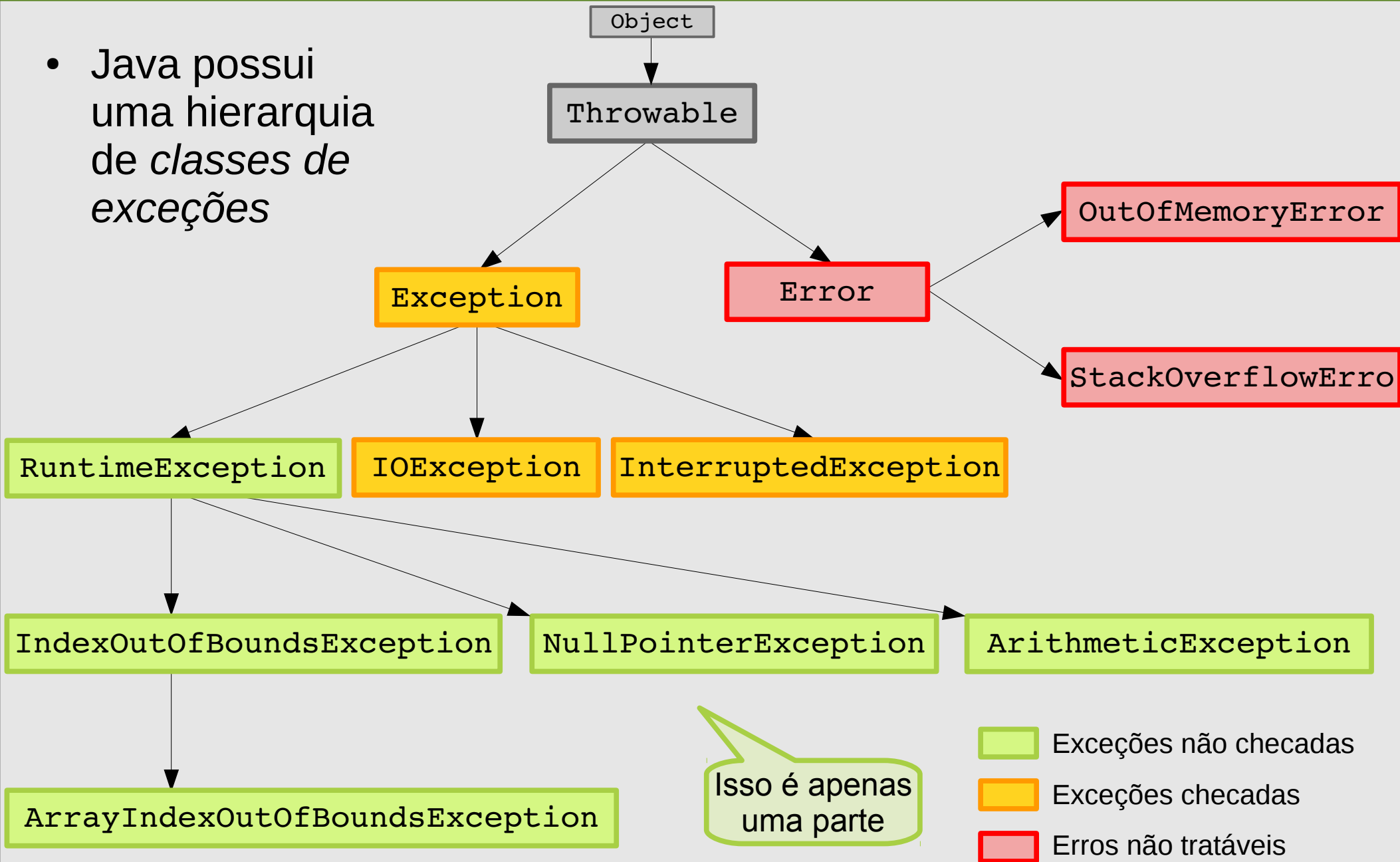
```
$ java TesteExcecao  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at TesteExcecao.main(TesteExcecao.java:5)
```

Tratamento de Exceções

Hierarquia de Exceções



- Java possui uma hierarquia de *classes de exceções*



Tratamento de Exceções

Tipos de Exceções



- Java possui três tipos de exceções

- Exceções não checadas

- *São exceções que podem ocorrer no programa, mas que não precisam ser obrigatoriamente tratadas*
 - *São todas que herdam a classe `RuntimeException`*

- Exceções checadas

- *São exceções que podem ocorrer e que obrigatoriamente precisam ser checadas pelo seu código*
 - *Um erro de compilação será gerado caso você não as trate*
 - *São todas que herdam a classe `Exception` (mas não a `RuntimeException`)*

- Erros

- *São exceções tão graves que não devem nem ser tratadas (e.g., falta de memória, stack overflow)*

Tratamento de Exceções

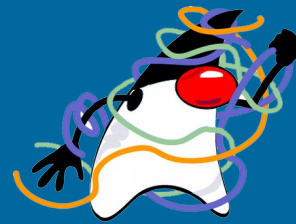
Tratamento de Exceções



- Tratar uma exceção é:
 - indicar ao java uma parte do código que pode gerar uma exceção; e
 - implementar um bloco que trate essa exceção, caso ela ocorra
- No caso das exceções checadas, o tratamento é obrigatório
 - No caso das exceções não checadas e dos erros, o tratamento é opcional

Tratamento de Exceções

Sintaxe



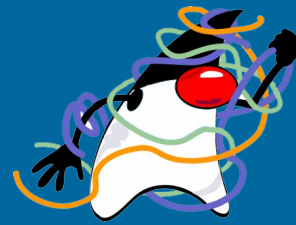
- Sintaxe

```
try {  
    // Bloco de instruções que podem gerar exceções  
}  
catch (UmTipoDeExcecao e1) {  
    // Bloco de instruções para tratar a  
    // exceção da classe UmTipoDeExcecao  
}  
catch (OutroTipoDeExcecao e2) {  
    // Bloco de instruções para tratar a  
    // exceção da classe OutroTipoDeExcecao  
}  
finally {  
    // Bloco de instruções que será  
    // executado sempre  
}
```



Tratamento de Exceções

Exemplo de Execução



- Execução normal (sem ocorrência de exceções)

A execução começa
no início do bloco

Como não houve
exceção, todos os
catches são
pulados e a
execução vai
para o finally

Após o finally,
a execução do
código continua
normalmente

```
try {  
    // Bloco de instruções que podem gerar exceções  
}  
catch (UmTipoDeExcecao e1) {  
    // Bloco de instruções para tratar a  
    // exceção da classe UmTipoDeExcecao  
}  
catch (OutroTipoDeExcecao e2) {  
    // Bloco de instruções para tratar a  
    // exceção da classe OutroTipoDeExcecao  
}  
finally {  
    // Bloco de instruções que será executado sempre  
}
```

Tratamento de Exceções

Exemplo de Execução



- Execução com exceção (da classe OutroTipoDeExcecao)

A execução começa no início do bloco

Como houve uma exceção, a execução pula para o catch correspondente

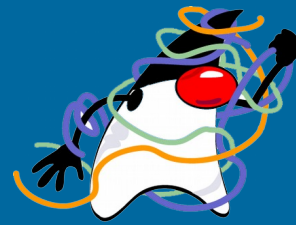
A exceção foi tratada, o controle vai agora para o finally

Após o finally, a execução do código continua normalmente

```
try {  
    // Bloco de instruções que podem gerar exceções  
}  
catch (UmTipoDeExcecao e1) {  
    // Bloco de instruções para tratar a  
    // exceção da classe UmTipoDeExcecao  
}  
catch (OutroTipoDeExcecao e2) {  
    // Bloco de instruções para tratar a  
    // exceção da classe OutroTipoDeExcecao  
}  
finally {  
    // Bloco de instruções que será executado sempre  
}
```

Tratamento de Exceções

Detalhes



- A palavra reservada `try`, marca o início do tratamento de exceção
 - Um `try` deve ser seguido de zero ou mais `catch`
 - O `finally` é opcional desde que se tenha pelo menos um `catch`
- O `finally` é sempre executado
 - Ocorrendo ou não uma exceção
 - A ideia é poder liberar recursos que foram alocados no `try`
 - *Exemplo: fechar arquivo*

Tratamento de Exceções

Exemplo / Quiz I

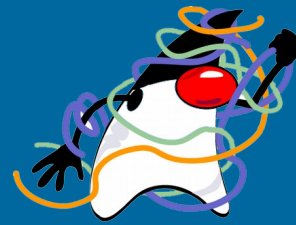


```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Erro no vetor!");  
        }  
        catch (Exception e) {  
            System.out.println("Houve um erro geral!");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Erro no vetor!  
Fim do programa.
```

Tratamento de Exceções

Exemplo / Quiz II



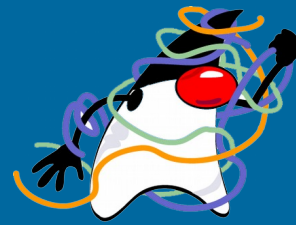
```
public class TesteExcecao {  
    public static void main(String args[]) {  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (Exception e) {  
            System.out.println("Houve um erro geral!");  
        }  
        catch (ArrayIndexOutOfBoundsException e) {  
            System.out.println("Erro no vetor!");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```



```
$ java TesteExcecao  
(...) Unreachable catch block for ArrayIndexOutOfBoundsException.  
It is already handled by the catch block for Exception  
at TesteExcecao.main(TesteExcecao.java:12)
```

Tratamento de Exceções

Exemplo / Quiz III



```
public class TesteExcecao {  
    public static void main(String args[]) {  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        finally {  
            System.out.println("Entrada garantida ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Entrada garantida ..  
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 3  
    at TesteExcecao.main(TesteExcecao.java:7)
```


Tratamento de Exceções

Exemplo / Quiz IV

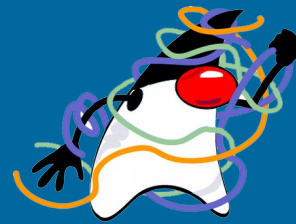


```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (Exception e) {  
            System.out.println("Houve um erro geral! Tratando ..");  
        }  
        finally {  
            System.out.println("Finalmente! Liberando recursos ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Houve um erro geral! Tratando ..  
Finalmente! Liberando recursos ..  
Fim do programa.
```

Tratamento de Exceções

Exemplo / Quiz V

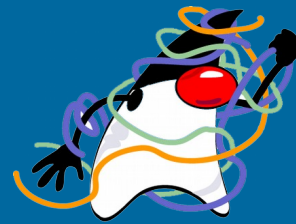


```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (Exception e) {  
            System.out.println("Houve um erro geral! Tratando ..");  
        }  
        finally {  
            System.out.println("Finalmente! Liberando recursos ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Houve um erro geral! Tratando ..  
Finalmente! Liberando recursos ..  
Fim do programa.
```

Tratamento de Exceções

Exemplo / Quiz VI



```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (Exception e) {  
            System.out.println("Houve um erro geral! Tratando ..");  
            return;  
        }  
        finally {  
            System.out.println("Finalmente! Liberando recursos ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Houve um erro geral! Tratando ..  
Finalmente! Liberando recursos ..
```

Tratamento de Exceções

Exemplo / Quiz VII



```
public class TesteExcecao {  
    public static void main(String args[]) {  
  
        int[] vet = new int[3];  
  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        }  
        catch (ArithmeticException e) {  
            System.out.println("Erro aritmetico! Tratando ..");  
        }  
        finally {  
            System.out.println("Finalmente! Liberando recursos ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```

```
$ java TesteExcecao  
Finalmente! Liberando recursos .. Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException: 3  
    at TesteExcecao.main(TesteExcecao.java:7)
```

Tratamento de Exceções

Disparando uma Exceção Existente



- Conforme mencionado, um método pode disparar uma exceção, indicando que algo inesperado aconteceu
- Para um método disparar uma exceção:
 1. ele precisa indicar quais exceções ele pode disparar
 - *Isso é feito através da palavra reservada `throws`*
 - *Opcional para as exceções não checadas*
 2. quando o inesperado acontecer, ele precisa disparar a exceção criando um novo objeto da classe de exceção apropriada
 - *Isso é feito através da palavra reservada `throw`*

Tratamento de Exceções

Disparando uma Exceção Existente



```
public class ArquivoTexto {  
    File arquivo;  
    public void abrir(String nome) throws NullPointerException, IOException {  
        if (nome == null) throw new NullPointerException();  
        this.arquivo = new File(nome);  
        if (!arquivo.exists()) throw new IOException();  
    }  
}
```

Indicando que o método pode disparar exceções

```
public class Principal {  
    public static void main(String args[]) {  
        ArquivoTexto arq = new ArquivoTexto();  
        arq.abrir("/nome/de/arquivo/errado.txt");  
    }  
}
```

Disparando exceções

```
$ javac Principal  
Principal.java:8: error: unreported exception IOException; must be caught  
or declared to be thrown  
    arq.abrir("/nome/de/arquivo/errado.txt");
```

Como o método abrir pode disparar IOException, que é uma exceção checada, ela precisa ser tratada

Tratamento de Exceções

Disparando uma Exceção Existente



```
public class ArquivoTexto {
    File arquivo;

    public void abrir(String nome) throws NullPointerException, IOException {
        if (nome == null) throw new NullPointerException();
        this.arquivo = new File(nome);
        if (!arquivo.exists()) throw new IOException();
    }
}
```

```
public class Principal {
    public static void main(String args[]) {
        ArquivoTexto arq = new ArquivoTexto();
        try {
            arq.abrir("/nome/de/arquivo/errado.txt");
        }
        catch (Exception IOException) {
            System.out.println("Arquivo não encontrado!");
        }
    }
}
```

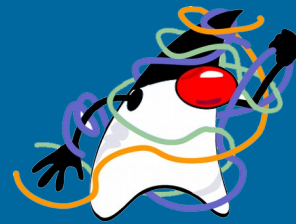
Tratar a exceção `NullPointerException` é opcional, uma vez que ela é uma exceção não checada

Mas tratar a `IOException` é obrigatória, uma vez que ela é exceção checada

```
$ java Principal
Arquivo não encontrado!
```

Tratamento de Exceções

Criando uma Nova Exceção



- Criar uma nova exceção é criar uma nova classe de exceção
 - De preferência, essa classe deve herdar uma classe de exceção já existente e que esteja mais relacionada com ela
 - Caso não tenha, ela deve herdar a classe `Exception`
 - Uma vez criada, ela pode ser disparada como qualquer outra exceção

```
import java.io.FileNotFoundException;

public class ArquivoTextoNaoEncontradoException
                                extends FileNotFoundException {
    private static final long serialVersionUID = 1L;

    public ArquivoTextoNaoEncontradoException() {
        this("Arquivo texto não encontrado");
    }

    public ArquivoTextoNaoEncontradoException(String s) {
        super(s);
    }
}
```


Tratamento de Exceções

Laboratório 6

- <http://tinyurl.com/slides-tp>
 - Laboratórios
 - TP – 6oLaboratorio.pdf
- Entrega por E-Mail
 - Para: horacio.fernandes@gmail.com
 - Cópia: moyses.lima@icomp.ufam.edu.br
 - Assunto: TP: 6o Lab
- Data Limite:
 - Hoje, às 12hs
 - E-Mails recebidos após 12hs não serão considerados

