



# INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho

[ivan@inf.puc-rio.br](mailto:ivan@inf.puc-rio.br)


## Programa – Capítulo 9




- Tratamento de Exceções
- Assertivas

Laboratório de Engenharia de Software

## Programa – Capítulo 9




- **Tratamento de Exceções**
- Assertivas




© LES/PUC-Rio

Laboratório de Engenharia de Software

## Exceções




- Exceções são condições anormais, possivelmente erros, que devem ser tratadas pelo programa de aplicação;
- Linguagens como Java, Python e C++ fornecem suporte ao tratamento sistemático de exceções;
- Tais mecanismos permitem separar o código relativo ao tratamento de exceções do resto do código da aplicação.



© LES/PUC-Rio

Laboratório de Engenharia de Software

## Exemplos de situações anormais




- Erros de lógica de programação:
  - Devem ser corrigidos pelo programador;
  - Exemplos: (a) limite do vetor ultrapassado; (b) divisão por zero;
- Erros devido a condições do ambiente de execução:
  - Fogem ao controle do programador, mas podem ser contornados;
  - Exemplos: (a) arquivo não encontrado; (b) conexão não estabelecida;
- Erros graves, em que não há recuperação:
  - Fogem ao controle do programador e não podem ser contornados;
  - Exemplos: (a) falta de memória; (b) erro interno da JVM.

© LES/PUC-Rio

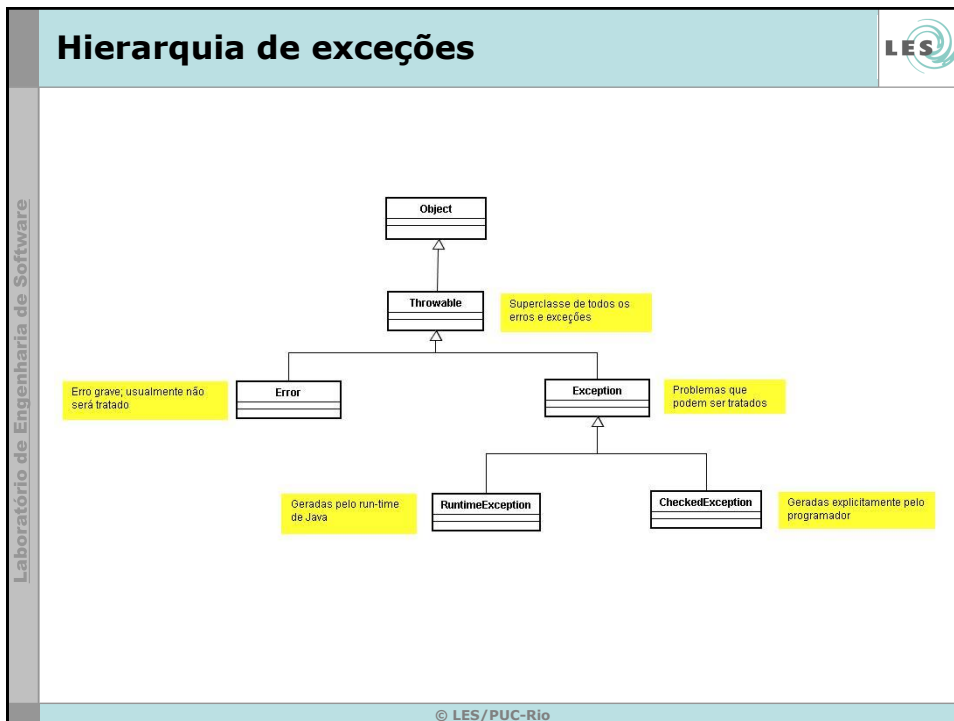
Laboratório de Engenharia de Software

## Tipos de exceção de Java



- De um modo geral, as exceções podem ser classificadas segundo o seguinte critério:
  - As geradas pelo run-time de Java;
  - As geradas explicitamente pelo programador (Checked Exceptions).
- Em ambos os casos, um objeto contendo informações sobre a exceção será instanciado;
- Tal objeto irá pertencer a uma das classes da hierarquia mostrada a seguir:

© LES/PUC-Rio




## Tratamento de exceções em Java

- A linguagem Java permite tratar sistematicamente exceções por meio do comando `try`;
- Um comando `try` deve envolver um bloco de código que pode levantar uma exceção;
- Ele deve ser seguido por:
  - Um ou mais blocos `catch(<TipoDeExcecao> e);`
  - Um bloco `finally` (opcional).

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Exemplo – try...catch




```
try {
    // código da aplicação
    // pode levantar uma exceção
}
catch (TipoExcecao1 ex) {
    // código para o tratamento da exceção
}
catch (TipoExcecao2 ex) {
    // código para o tratamento da exceção
}
catch (Exception ex) {
    // código para o tratamento da exceção
    // captura qualquer exceção
}

// código da aplicação - continuação
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Dinâmica do mecanismo try...catch




- O código contido em um bloco `try` levanta uma exceção;
- Um objeto contendo informações sobre a exceção é instanciado;
- A execução dos comandos contidos no bloco `try` é interrompida;
- O fluxo de execução desce pelos blocos `catch` até encontrar um que declare tratar a exceção;
- O bloco `catch` é executado;
- O bloco `finally` é executado, caso exista um;
- O código que segue o mecanismo `try...catch` é executado, caso não exista um `return` no bloco que tratou a exceção.

© LES/PUC-Rio

Laboratório de Engenharia de Software

**try...catch – Observações (1)**




- O bloco `catch` executado será aquele cujo parâmetro for da mesma classe ou de uma classe ancestral da exceção levantada;
- Apenas um dos blocos `catch` será executado;
- Não pode existir código algum entre o bloco `try` e o primeiro bloco `catch`;
- Não pode existir código algum entre dois blocos `catch`;

© LES/PUC-Rio

Laboratório de Engenharia de Software

**try...catch – Observações (2)**




- O bloco `try` deve ser seguido por pelo menos um bloco `catch`, ou por um bloco `finally`;
- Caso um `catch` mais genérico (ex: `catch(Exception e)`) apareça antes de um mais específico, o bloco mais específico jamais será executado;
- O compilador detecta a situação acima e sinaliza o erro;
- Para tratar uma exceção qualquer construa um `catch` que capture uma exceção do tipo `Exception`;
- Esta técnica, deve ser usada como um último recurso, mas não para substituir os tratamentos das exceções específicas.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## O bloco finally



- O bloco `finally` contém instruções que devem ser executadas independentemente da ocorrência ou não de exceções.


```
try {
    // instruções executadas até linha onde ocorrer exceção
}
catch (TipoExcecao1 ex) {
    // executado somente se ocorrer TipoExcecao1
}
catch (TipoExcecao2 ex) {
    // executado somente se ocorrer TipoExcecao2
}
finally {
    // executado caso seja levantada, ou não, uma exceção
}

// os comando posicionados após os blocos de
// try...catch...finally são executados caso uma exceção
// seja tratada ou caso não ocorra exceção alguma
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Mensagem da exceção



- Dentro de um bloco `catch` pode-se ter acesso à mensagem explicativa do motivo da exceção;
- Isso é feito por meio do método `getMessage()`:

```
try {
    ...
    throw new NovaExcecao("Ocorreu uma excecao");
}
catch (NovaExcecao e) {
    System.out.println(e.getMessage());
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Sequência de eventos (1)

- Quando uma exceção é levantada, o fluxo normal do programa é interrompido e os passos seguintes são executados:
  - O fluxo do programa segue a exceção;
  - Caso ela não seja tratada no método onde ela foi levantada, ela será propagada para o método que chamou o método gerador da exceção, e assim por diante.
- Caso nenhum método trate a exceção levantada a execução do programa será encerrada;
- A figura a seguir ilustra o que foi dito acima:

LES

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Sequência de eventos (2)

Cadeia de chamadas em tempo de execução

```

main()
  op1();
    op2();
  
```

Código fonte

```

void main()
{
  op1();
}
void op1()
{
  op2();
}
void op2()
{
  int i, j=0;
  i=i/j;
}
  
```

7

6

5

4

3

2

1

O sistema de tempo de execução procura por um try...catch no método main()

O sistema de tempo de execução procura por um try...catch no método op1()

O sistema de tempo de execução procura por um try...catch no método op2()

A divisão por zero levanta uma exceção;

Caso a exceção não seja tratada na função main() a execução do programa será encerrada e uma mensagem de erro será exibida na console.


LES

© LES/PUC-Rio



Laboratório de Engenharia de Software

## Checked Exceptions (1)



- Para gerar uma checked exception é necessário, primeiro, estender `java.lang.Exception`:

```
class NovaExcecao extends Exception {}
```


- Pode-se acrescentar variáveis de instância e métodos à classe de exceção, como em qualquer outra classe;
- Deve-se fornecer dois construtores para uma classe de exceção:

```
public class NovaExcecao extends Exception {  
    public NovaExcecao() {}  
    public NovaExcecao (String mensagem) {  
        super(mensagem);  
    }  
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Checked Exceptions (2)



- O passo seguinte é criar (`new`) e depois lançar (`throw`) a exceção:

```
NovaExcecao e = new NovaExcecao("Ocorreu uma excecao");  
  
throw e; // exceção lançada
```


- A referência é desnecessária. A sintaxe abaixo é mais usual:

```
throw new NovaExcecao("Ocorreu uma excecao");
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Exceções e métodos (1)




- Caso uma exceção seja explicitamente levantada no interior de um método, uma das medidas a seguir tem de ser tomada:
  - (a) tratar a exceção no próprio método;

```
try {  
    ...  
    throw new NovaExcecao();  
}  
catch (NovaExcecao ex) {  
    ...  
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Exceções e métodos (2)



- (b) não tratar a exceção localmente e declarar que o método pode levantar uma exceção do tipo declarado (ou de qualquer subtipo).


```
public void m1() throws NovaExcecao, Excecao2 {  
    ...  
}
```

- A palavra reservada `throws` declara que o método `m1()` pode levantar uma das exceções declaradas.

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Relançamento de uma exceção



- Às vezes, após tratar uma exceção, é desejável relançá-la para que outros métodos lidem com ela;
- Isto pode ser feito da seguinte forma:

```
public void metodo() throws ExcecaoSimples {  
    try {  
        // instruções  
    }  
    catch (ExcecaoSimples ex) {  
        // faz alguma coisa para lidar com a exceção  
        throw ex; // relança exceção  
    }  
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

## Programa – Capítulo 9



- Exceções
- **Assertivas**

© LES/PUC-Rio

## Assertivas



- As assertivas foram introduzidas no JDK 1.4 com o objetivo de auxiliar os programadores a encontrar erros em programas (*debugging*);
- Um assertiva é uma declaração que pode ser verdadeira ou falsa;
- Pode-se espalhar assertivas em determinados pontos de um programa para testar se certas condições são verdadeiras ou não;
- Se a declaração for falsa a assertiva levanta uma condição de erro (`java.lang.Error`).

## O comando `assert`



- O comando `assert` possui duas formas:  

```
assert <expressaoBooleana>;  
assert < expressaoBooleana> : <expressao>;
```
- Caso a expressão booleana seja falsa uma condição de erro do tipo `java.lang.AssertionError` será levantada;
- Na segunda forma, uma mensagem pode ser passada para o construtor da classe `AssertionError`;
- Deve-se usar assertivas apenas para detectar erros fatais – aqueles não podem ser tratados e que irão encerrar a aplicação.

## O comando assert – Exemplo



```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner s=new Scanner(System.in);

        System.out.println("Informe um inteiro\n");
        int x=s.nextInt();
        System.out.println("Informe um inteiro\n");
        int y=s.nextInt();

        assert y!=0.0 : "Denominador igual a zero";

        System.out.println("x/y="+ (x/y));
    }
}
```

## Execução da aplicação



- Após a compilação da classe `Ex`, executa-se a aplicação, como é mostrado na figura abaixo:

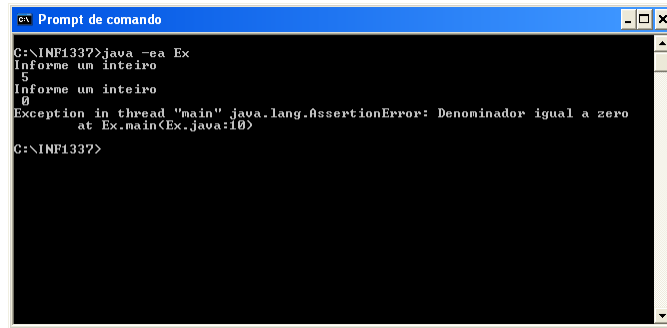
```
C:\INF1337>java Ex
Informe um inteiro
5
Informe um inteiro
0
x/y=Infinity
C:\INF1337>_
```

- Veja que o comando `assert` não teve nenhuma influência no resultado da execução.

## Comando java -ea



- É necessário habilitar as assertivas para que elas tenham efeito (opção `-ea` no comando `java`):



```
C:\INF1337>java -ea Ex
Informe um inteiro
5
Informe um inteiro
0
Exception in thread "main" java.lang.AssertionError: Denominador igual a zero
    at Ex.main(Ex.java:10)
C:\INF1337>
```