

# TRATAMENTO DE EXCEÇÕES

## ERROS E EXCEÇÕES JAVA



21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.

1



## AGENDA

1. Introdução
2. Exceções em Java
3. Tipos de Exceções
4. Exceções Verificadas
5. Exceções Não-Verificadas
6. Erros
7. Projetando Exceções
8. Tratamento de Exceções
9. Recomendações

21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.

2



## INTRODUÇÃO

- Uma **exceção** é uma indicação de um problema que ocorre durante a execução de um programa.

21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.

3



## INTRODUÇÃO

- O tratamento de exceções permite aos programadores criar aplicativos que podem resolver exceções;
- As exceções são lançadas quando um método detecta um problema e é incapaz de tratá-lo;
- Com o tratamento de exceções, se o usuário cometer um erro, o programa **captura e trata** a exceção.

21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.

4



## EXEMPLOS

- Índice de uma lista (Array) fora do intervalo permitido (**IndexOutOfBoundsException**);
- Problemas em operações aritméticas, tais como "overflows" e divisões por zero (**ArithmeticException**);
- Argumentos inválidos em uma chamada a um método (**IllegalArgumentException**);
- Uso de uma referência que não aponta para nenhum objeto (**NullPointerException**);
- Falta de Memória (**OutOfMemoryError**).

21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.

5




## EXCEÇÕES EM JAVA

- Exceções são objetos, o que oferece uma maior flexibilidade para a inclusão de informações relativas ao contexto da ocorrência da exceção;
- **Throwable** é a classe base para todos os tipos de exceções e erros.

21/9/2007

Tratamento de Exceções  
Prof. Ademair Schmitz, M.Sc.


6



## EXCEÇÕES EM JAVA

- Diferença entre Erro e Exceção
  - A Sun menciona que **Erro** indica um problema sério que uma aplicação não tem como tratar. A maioria dos erros acontecem em circunstâncias anormais.
  - A Sun explica que uma **Exceção** indica uma condição anormal que deve ser cuidadosamente manipulada para prevenir a finalização da aplicação.


21/9/2007
Tratamento de Exceções  
Prof. Ademar Schmitz, M.Sc.
7



## EXCEÇÕES EM JAVA

- A classe **Throwable** é responsável em lançar (throw) qualquer tipo de exceção em Java, inclusive os erros.
- Objetos podem ser lançados para indicar que algo anormal aconteceu, e capturados para lidar com esta situação, devem estender a classe **Throwable** do pacote **java.lang**.


21/9/2007
Tratamento de Exceções  
Prof. Ademar Schmitz, M.Sc.
8



## EXCEÇÕES EM JAVA

- Os métodos mais úteis da classe **Throwable** são:
  - public String getMessage()**
    - Retorna a mensagem do erro.
  - public void printStackTrace()**
    - Imprime uma descrição da pilha no instante em que ocorreu o problema.

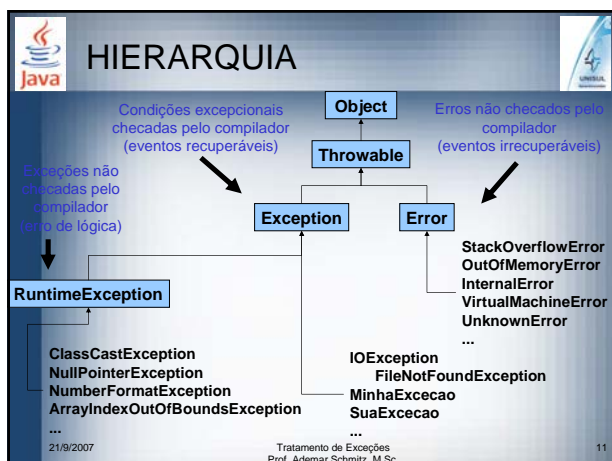
21/9/2007
Tratamento de Exceções  
Prof. Ademar Schmitz, M.Sc.
9




## EXCEÇÕES EM JAVA

- Java separa as exceções em dois tipos:
  - public class Exception extends Throwable**
  - public class Error extends Throwable**
- Classes derivadas de **Exception** teoricamente, podem ser recuperáveis.
- Classes derivadas de **Error**, teoricamente, não podem ser recuperáveis (finaliza a aplicação).

21/9/2007
Tratamento de Exceções  
Prof. Ademar Schmitz, M.Sc.
10






## TIPOS DE EXCEÇÕES

- As exceções em Java dividem-se em duas categorias:
  - Exceções verificadas;
  - Exceções não-verificadas;
- Lembrando que há também uma segunda categoria de erros internos que são informados disparando-se objetos do tipo *Error*.


21/9/2007
Tratamento de Exceções  
Prof. Ademar Schmitz, M.Sc.
12



## EXCEÇÕES VERIFICADAS

- Temos que dizer ao compilador o que iremos fazer a respeito da exceção;
- Herdadas diretamente da classe *Exception* ou qualquer uma de suas subclasses diretas ou indiretas, exceto *RuntimeException*;
- Se devem a circunstâncias externas que o programador não consegue prever.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 13



## EXEMPLOS: EXCEÇÕES VERIFICADAS

- *IOException*
- *FileNotFoundException*
- *ClassNotFoundException*

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 14




## EXEMPLO: EXCEÇÃO VERIFICADA

```
BufferedReader console
= new BufferedReader(new InputStreamReader(System.in));
String input = console.readLine()
//readLine() throws IOException

String filename = "C:/fonte/arquivo.txt";
BufferedReader console
= new BufferedReader(new FileReader(filename));
//FileReader() throws FileNotFoundException
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 15



## EXCEÇÕES NÃO VERIFICADAS

- O compilador não exige que as monitoremos;
- Estendem a classe *RuntimeException*;
- Se devem a erros cometidos pelo programador.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 16



## EXEMPLOS: EXCEÇÕES NÃO-VERIFICADAS

- *NumberFormatException*
- *IllegalArgumentException*
- *NullPointerException*

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 17

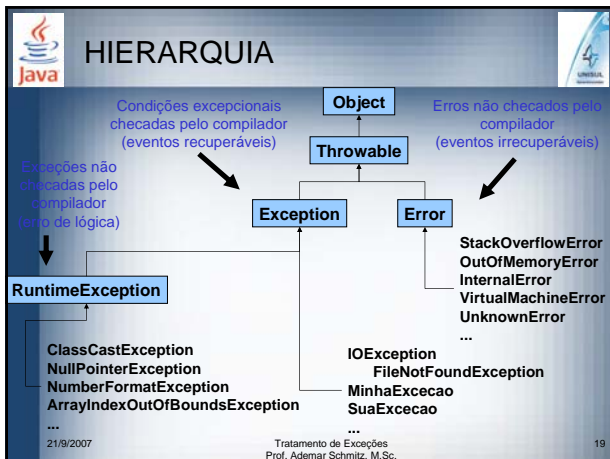


## EXEMPLO: EXCEÇÃO NÃO-VERIFICADA

```
int[] intArray = new int[10];
for (int i = 0; i <= 11; i++) {
    intArray[i] = i;
}
//ArrayIndexOutOfBoundsException

String nome = new String("Programação para Computação II");
System.out.println(nome.charAt(10));
nome = null;
System.out.println(nome.charAt(10));
//NullPointerException
```

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 18



- ## ERROS
- Há uma segunda categoria de erros internos que são informados disparando-se objetos do tipo *Error*;
  - Esses são erros fatais que raramente acontecem e estão fora do nosso controle;
  - Eles também são não-verificados.
- 21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 20

- ## EXEMPLOS: ERROS
- *OutOfMemoryError*
  - *StackOverflowError*
- 21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 21

## EXEMPLO: OutOfMemoryError

```
public class Erro1 {

    public static void main(String[] args) {
        Vector<String> strings = new Vector<String>();
        boolean teste = true;
        while (teste == true) {
            strings.add(new String ("Sequencial de Java"));
        }
    }

}
```

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 22

## EXEMPLO: StackOverflowError

```
public class Erro2 {

    public static void metodo1() {
        metodo2();
    }

    public static void metodo2() {
        metodo1();
    }


    public static void main(String[] args) {
        metodo1();
    }

}
```

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 23

- ## PROJETANDO EXCEÇÕES
- Às vezes, nenhum dos tipos de exceção-padrão descreve suficientemente bem nossa situação de erro específico;
  - Neste caso, podemos projetar nossa própria classe de exceção;
  - Podemos projetar nossos próprios tipos de exceção – subclasses de *Exception* ou *RunTimeException*;
- 21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 24






## EXEMPLO: EXCEÇÃO

```
public class ExcecaoVelocida extends Exception {

    public ExcecaoVelocida(String motivo) {
        super(motivo);
    }

}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 25



## DISPARANDO EXCEÇÕES

- O mecanismo de tratamento de exceções foi projetado para resolver dois tipos de problemas:
  - As exceções não podem ser desprezadas.
  - As exceções podem ser tratadas por um tratador competente.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 26



## DISPARANDO EXCEÇÕES

- Para sinalizar uma situação excepcional, use-se o comando *throw* para disparar uma objeto de exceção;
- Quando disparamos uma exceção, o método corrente termina imediatamente.

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 27




## DISPARANDO EXCEÇÕES

- Sintaxe:
 

```
throw ObjetoDeExceção;
```
- Exemplo:
 

```
throw new IllegalArgumentException();
```
- Objetivo:
 Disparar uma exceção e transferir o controle para um tratador desse tipo de exceção.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 28



## EXEMPLO: Carro

```
public void ligar() throws ExcecaoLigado {
    if (!ligado) {
        ligado = true;
    } else {
        throw new ExcecaoLigado("Carro já está ligado");
    }
}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 29



## ESPECIFICANDO EXCEÇÕES

- Acrescente um especificador *throws* a um método que pode disparar uma exceção verificada;
- A cláusula *throws* sinaliza ao chamador do método que ele pode encontrar uma exceção;
- O chamador precisa tomar a sua decisão – tratar a exceção ou dizer ao seu chamador que a exceção pode ser disparada;
- Se o método puder disparar múltiplas exceções verificadas, separamo-las por vírgulas.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 30



## ESPECIFICANDO EXCEÇÕES

- **Sintaxe**  
`especificadorDeAcesso tipoDeRetorno nomeDoMétodo  
(tipoDoParâmetro nomeDoParâmetro, ...)  
throws ClasseDeExceção, ...`
- **Exemplo**  
`public void read (BufferedReader in) throws  
IOException`
- **Objetivo**  
Indicar as exceções verificadas que esse método pode disparar.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 31



## EXEMPLO: Carro

```
public void ligar() throws ExcecaoLigado {
    if (!ligado) {
        ligado = true;
    } else {
        throw new ExcecaoLigado("Carro já ligado");
    }
}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 32



## CAPTURANDO EXCEÇÕES

- Toda exceção deve ser tratada em algum lugar em nosso programa;
- Se uma exceção não tem tratador, uma mensagem de erro é impressa, e seu programa, abortado.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 33



## CAPTURANDO EXCEÇÕES

- Instalamos um tratador de exceção com o comando *try*;
- Cada **bloco try** contém uma ou mais chamadas de métodos que podem causar uma exceção, e cláusulas *catch* para todos os tipos possíveis de exceção que o **bloco try** pode tratar.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 34



## CAPTURANDO EXCEÇÕES

- Se qualquer uma das exceções for disparada, então o resto das instruções do bloco *try* são puladas, e a cláusula *catch* adequada é executada imediatamente;
- Quando o **bloco catch** é executado, significa que algum método do **bloco try** falhou com uma exceção, e esse objeto é armazenado na variável;
- A cláusula *catch* pode analisar esse objeto para descobrir mais detalhes sobre a falha.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 35



## CAPTURANDO EXCEÇÕES

- Devemos colocar cláusulas *catch* apenas nos métodos em que podemos competentemente tratar a exceção;
- É melhor declarar que um método dispara uma exceção verificada do que tratar a exceção de maneira pouco precisa.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 36



## SINTAXE: BLOCO try

```
try {
    comando
    ...
} catch (classeDaExceção objetoDaExceção) {
    comando
    ...
} catch (classeDaExceção objetoDaExceção) {
    comando
    ...
} finally {
    comando
    ...
}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 37



## EXEMPLO: Motorista

```
public int acelerarCarro(Carro carro, int valor) {
    int vel = 0;
    try {
        vel = carro.acelerar(valor);
    } catch (ExcecaoLigado el) {
        System.out.println(el.getMessage());
        el.printStackTrace();
    } catch (ExcecaoVelocidade ev) {
        System.out.println(ev.getMessage());
        ev.printStackTrace();
    }
    return vel;
}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 38



## OBJETIVO: BLOCO try

- Executar uma ou mais instruções que podem gerar exceções;
- Se uma exceção de um determinado tipo ocorrer, então paramos de executar essas instruções e vamos para a cláusula *catch* correspondente;
- Se não ocorrer nenhuma exceção, pulamos as cláusulas *catch*;
- Em todos os casos, executamos a cláusula *finally* se houver alguma presente.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 39



## A CLÁUSULA finally

- Ocasionalmente, temos de tomar alguma atitude se uma exceção for disparada ou não;
- O comando *finally* é usado para tratar essa situação;
- Um vez que o programa entre em um bloco *try*, os comandos em um cláusula *finally* garantidamente serão executados, seja disparada uma exceção ou não.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 40



## A CLÁUSULA finally

- Utilizamos a cláusula *finally* sempre que precisamos fazer alguma limpeza, tal como fechar um arquivo, para garantir que a limpeza ocorra independente de como o método termine;
- Também é possível ter uma cláusula *finally* após uma ou mais cláusulas *catch*;


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 41



## CLÁUSUA finally

- O código da cláusula *finally* é executado sempre que o programa sair do **bloco try** por meio de qualquer uma das três maneiras seguintes:
  - Após completar o último comando do **bloco try**.
  - Quando uma exceção for disparada no **bloco try** que esta sendo passada para o chamador deste método.
  - Quando uma exceção foi disparada no **bloco try** que foi tratado por uma das cláusulas *catch*.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 42



## EXEMPLO: CLÁUSULA finally

```
try {
    in = new BufferedReader(new FileReader (filename));
    // faz alguma coisa
} finally {
    if (in != null) {
        in.close();
    }
}
```


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 43



## RECOMENDAÇÕES

- Use exceções apenas para condições excepcionais:
  - As exceções, como o próprio nome indica, devem ser usadas apenas em condições excepcionais, nunca para controle de fluxo.
- Use exceções verificadas para condições recuperáveis e exceções não verificadas para erros de programação:
  - Há três tipos de throwable na linguagem Java: exceções verificadas, exceções não-verificadas e erros.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 44



## RECOMENDAÇÕES

- Evite o uso desnecessário de exceções verificadas.
- Prefira o uso de exceções padrão:
  - O reuso de código é uma boa prática também no caso de exceções.
- Documente todas as exceções lançadas por cada método:
  - A descrição das exceções lançadas é uma parte importante da documentação necessária para utilizar um método corretamente.


21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 45



## RECOMENDAÇÕES

- Inclua informações sobre a falha em mensagens de detalhe:
  - O stack trace de uma exceção é sua representação em String, resultado do método String.
- Tente garantir atomicidade das falhas:
  - Um método cuja chamada não deu certo deve manter o objeto no estado em que ele estava antes da chamada ao método.

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 46




## RECOMENDAÇÕES

- Não ignore exceções

```
try {
    //código
} catch(Exception e {
    // tratamento
}
```

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 47



## RECOMENDAÇÕES

- Lance exceções apropriadas à abstração

```
try {
    //código
} catch(LowerLevelException e {
    throw new HigherLevelException(...);
}
```

21/9/2007 Tratamento de Exceções Prof. Ademar Schmitz, M.Sc. 48





## RECOMENDAÇÕES



- Havendo mais de uma cláusula catch para um mesmo bloco try, elas devem ser ordenadas da mais específica para a mais genérica (tratar a filha e propagar a pai).