Programação em Java - Fundamentos

5 - Tratamento de exceções

Citeforma

Jose Aser Lorenzo, Pedro Nunes, Paulo Jorge Martins

jose.l.aser@sapo.pt, pedro.g.nunes@gmail.com, paulojsm@gmail.com

Março de 2012

Sumário

Tratamento de exceções

citeforma

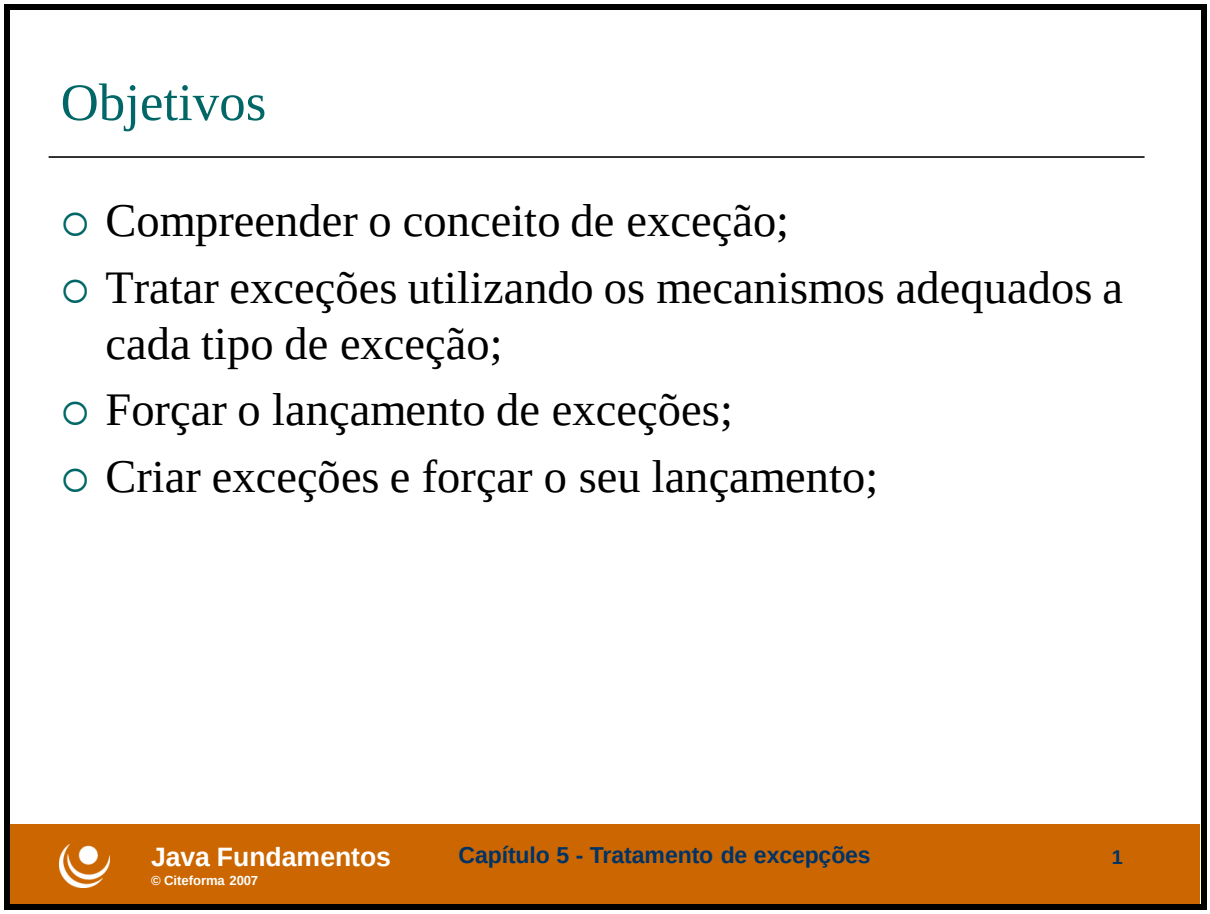
Programação em Java
Fundamentos

Capítulo 5 – Tratamento de exceções

José Aser Lorenzo
Pedro Nunes
Paulo Jorge Martins

© Citeforma 2007

Objetivos



No fim deste capítulo compreenderá o conceito de exceção e estará apto a utilizar os mecanismos adequados para as tratar, evitando a paragem do programa por erro de execução. Será também capaz de criar as suas próprias classes de exceção.

As classes desenvolvidas nos exercícios deste capítulo deverão ficar dentro do projeto **JavaFundamentosProjecto** e dentro do package **capitulo5**.

Exceção

Sumário

- Exceção;
- try-catch-finally;
- Escalar exceção não tratada;
- Forçar o lançamento de uma exceção;

© Citeforma 2007

O que é uma exceção?

O que é uma excepção?

- A execução de um programa pode parar repentinamente (inesperadamente) por causa de um **erro de runtime**;
- Um erro de runtime pode ter várias causas:
  - **Erro de lógica**: divisão por zero, acesso fora dos limites do array, leitura de número com formato inválido, …;
  - **Erro físico**: ficheiro removido, ficheiro sem permissões, erro de comunicações, …;
- Uma exceção é um objeto da classe **Exception** criado pelo JRE como resposta a um erro de runtime;

Java Fundamentos Capítulo 5 - Tratamento de excepções 3

© Citeforma 2007

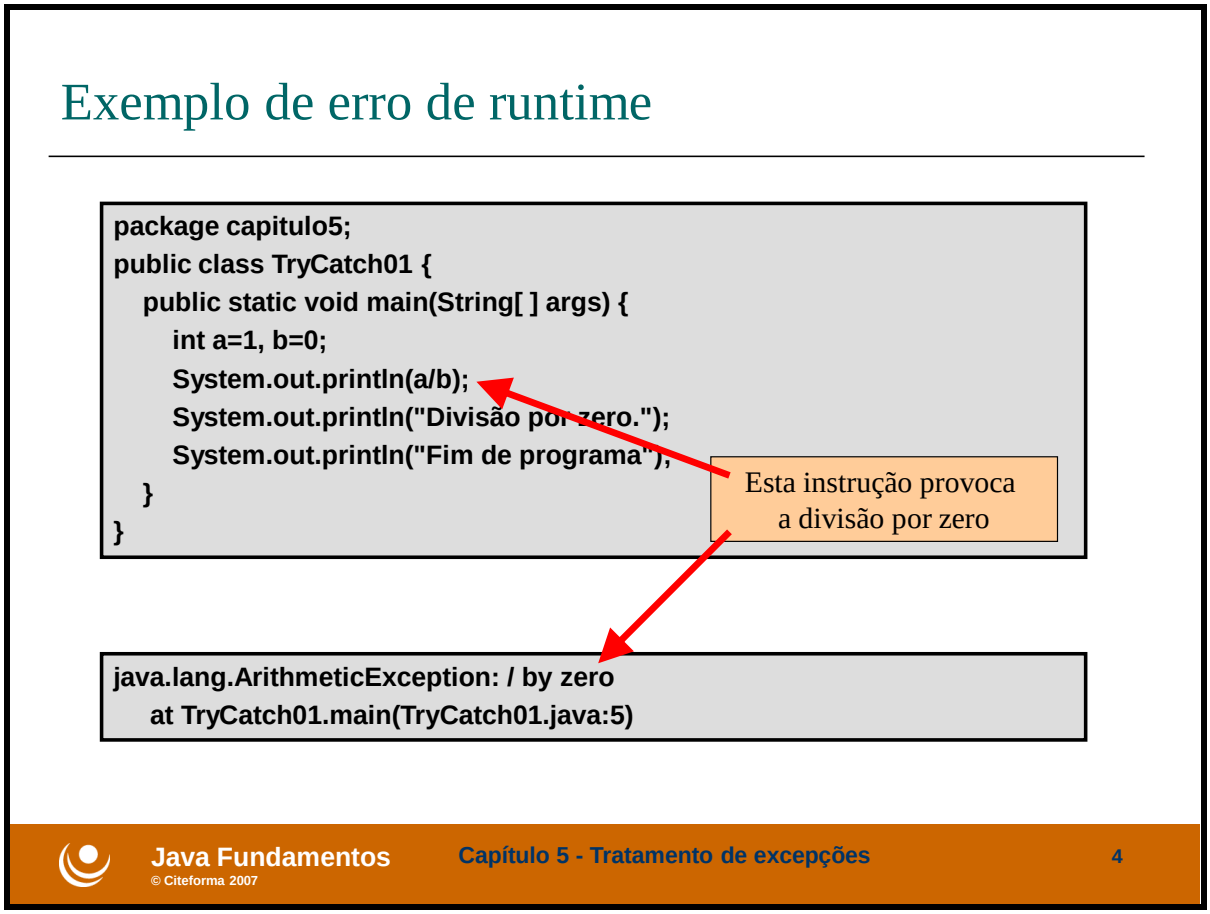
Durante a execução de um programa pode surgir um erro, normalmente designado por "*runtime-error*", que provoca a sua paragem abrupta. O programa não contém erros de sintaxe e pode nem ter erros de lógica, mas o erro ocorre durante a execução. As causas mais frequentes para estes erros são:

- Um erro de entrada/saída de dados, por exemplo quando o programa está a ler um **int** e o utilizador introduz um **float**;
- Um dispositivo não preparado (impressora, rede);
- A memória RAM encheu ("*memory overflow*");
- Um ficheiro não encontrado;
- Uma divisão por zero;
- Falha temporária da ligação à rede;

A paragem repentina de um programa deve ser evitada, pois não só interrompe o trabalho que o programa estava a fazer, como também o pode destruir ou deixár num estado inconsistente.

Suponha uma aplicação que regista transferências bancárias, onde cada transferência envolve o levantamento de uma quantia em dinheiro de uma conta e o seu depósito noutra. Uma transferência só está completa se ambas operações, levantamento e depósito, se confirmarem. Se a execução do programa parar abruptamente depois do levantamento e antes do depósito, em que estado fica a transferência?

Exemplo de erro de runtime



O programa apresentado no slide não contém erros de sintaxe mas provoca uma divisão por zero em tempo de execução.

Uma exceção é um objeto que pertence à classe **Exception** (ou suas derivadas) que é gerado (instanciado) pela máquina virtual quando ocorre um erro em tempo de execução. Este objeto contem um conjunto de propriedades que ajudam o programador a identificar a causa do problema.

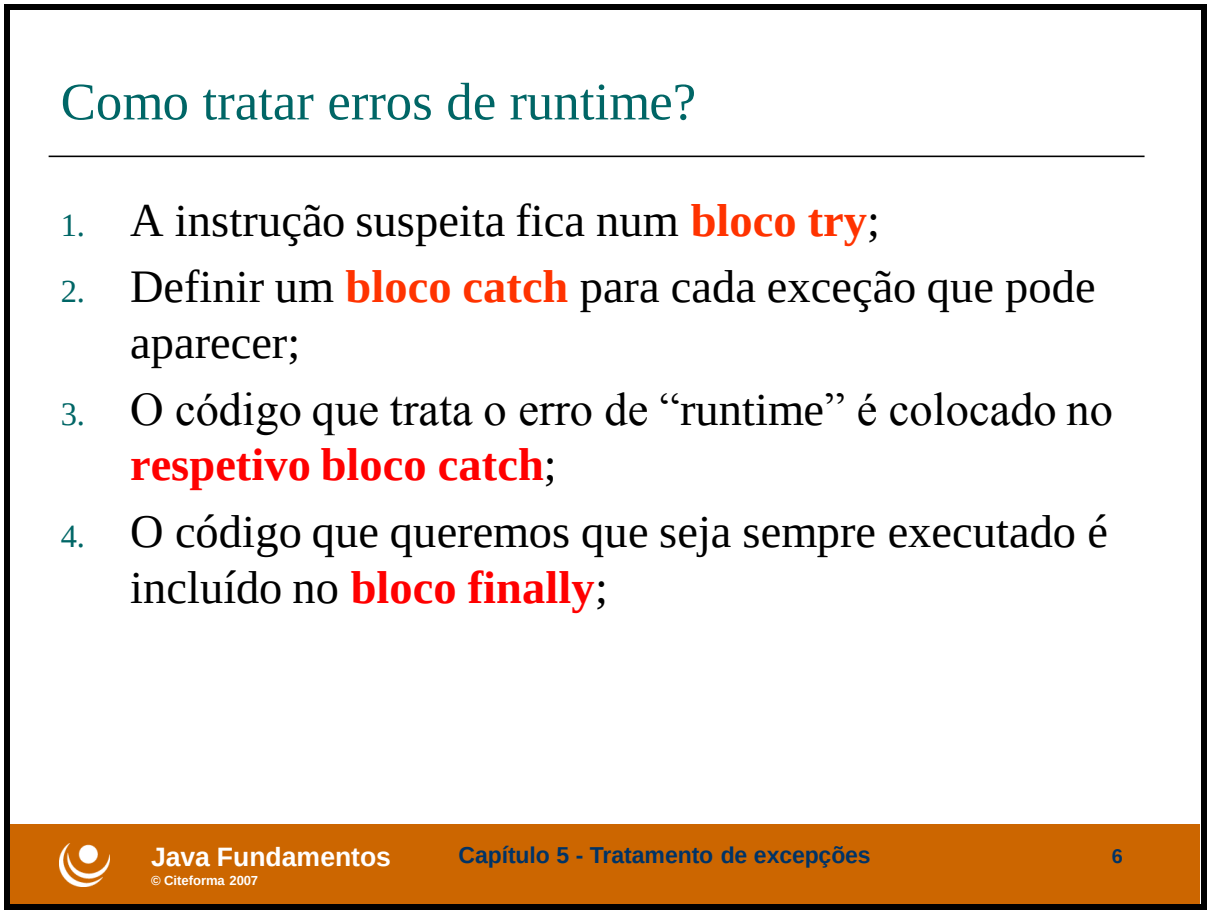
try-catch-finally

Sumário

- Exceção;
- try-catch-finally;
- Escalar exceção não tratada;
- Forçar o lançamento de uma exceção;

© Citeforma 2007

Como tratar erros de runtime?

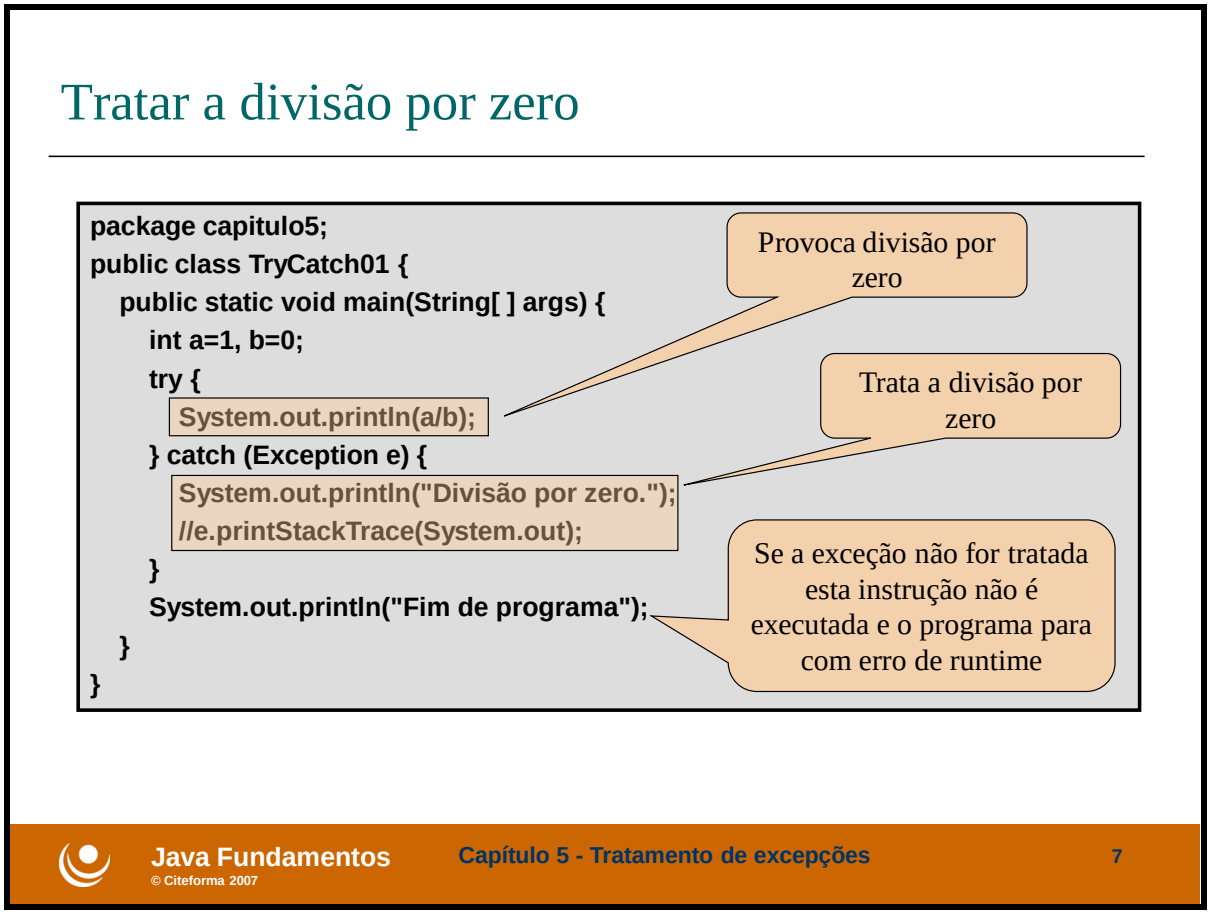


A linguagem Java possui um mecanismo para lidar com os erros em tempo de execução que inclui três partes:

- Na primeira é executada a instrução que poderá gerar o erro, incluída num bloco **try**;
- Na segunda são definidas as ações a tomar caso o erro ocorra, incluídas num bloco **catch**;
- Na terceira são definidas as ações que devem ser sempre executadas, quer o erro ocorra ou não, incluídas no bloco **finally**;

O erro do último exemplo ocorreu numa situação muito simples, mas poderia ocorrer no meio de cálculos complexos e demorados, o que interromperia a execução do programa, com passagem de controlo para o sistema operativo. O tratamento do erro permite ao programador retomar o controlo do programa, executando ações que minimizam o impacto do erro, o contornam ou ultrapassam. Se nada disto for possível, o programador tem pelo menos a oportunidade de registar as causas que provocaram o erro para ajudar na sua posterior correção ("debugging").

Tratar a divisão por zero



O exemplo acima mostra como tratar o erro da divisão por zero. A divisão é colocada no bloco **try** e o tratamento do erro no bloco **catch**.

A execução deste programa produz o seguinte resultado:

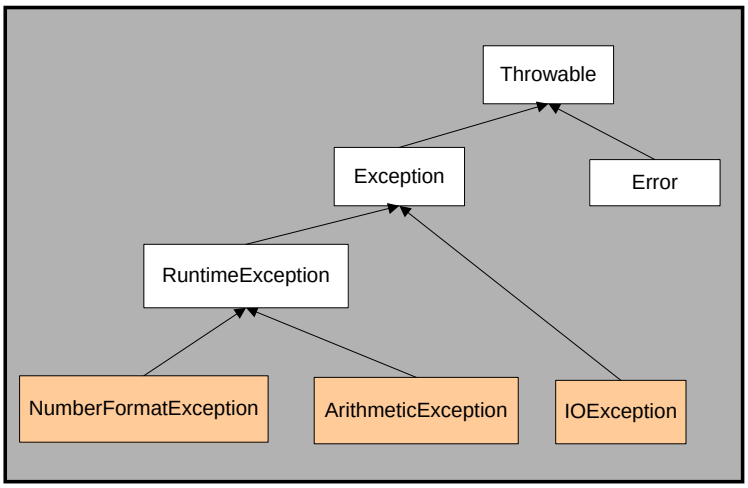
```
Divisao por zero !
Fim de programa
```

Com o tratamento do erro a divisão por zero não provocou a interrupção repentina da execução do programa, tendo este terminado normalmente, o que significa que devolveu ao sistema operativo um código de retorno com o valor zero.

Subclasses de Exception

Subclasses de Exception

- A classe **Exception** possui subclasses cujas propriedades se adaptam a erros específicos:



Java Fundamentos Capítulo 5 - Tratamento de excepções 8
© Citeforma 2007

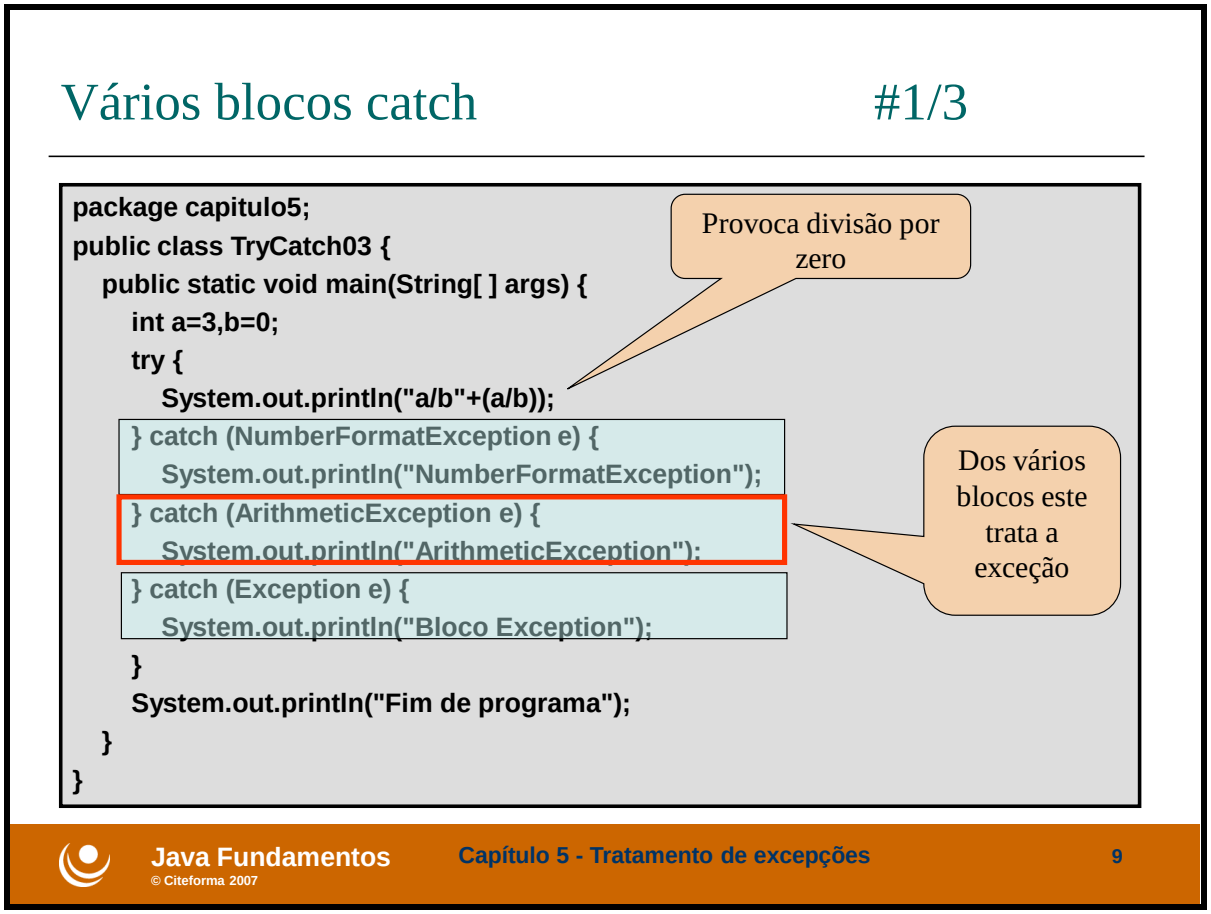
Quando ocorre um erro em tempo de execução a JVM cria um objeto da classe **Exception**, cujas propriedades caracterizam de forma genérica o tipo de erro gerado. A classe **Exception** possui várias subclasses pré definidas, podendo o utilizador definir outras. O diagrama do slide mostra as relações entre algumas destas classes.

As subclasses de **Exception** foram criadas para tratar erros específicos, pelo que contêm propriedades adequadas à descrição de erros comuns nas mais variadas áreas.

Para um bloco **try** podemos ter **vários blocos catch**. Em cada bloco **catch** trata-se um erro possível de ocorrer, utilizando uma das subclasses de **Exception**. Ao ocorrer um erro na execução do bloco **try**, os blocos **catch** são percorridos sequencialmente até encontrar a primeira classe cujas propriedades se ajustam ao erro ocorrido.

Se nenhum **catch** se ajustar ao erro ocorrido o programa interrompe a execução com erro. No entanto qualquer evento tem propriedades que se ajustam a **Exception**. Por este motivo, no último **catch** deve colocar-se um objeto da classe **Exception**, para que, caso as classes utilizadas nos primeiros blocos **catch** não se ajustem ao evento, então teremos garantias que o programa entrará no último **catch**.

Vários blocos catch



No exemplo do slide o erro resultante da divisão por zero é tratado num dos blocos catch.

A execução produz o seguinte resultado:

```
ArithmeticException
Fim de programa
```

Vários blocos catch #2/3

```
package capitulo5;
public class TryCatch04 {
   public static void main(String[ ] args) {
      int a;
      try {
         a=Integer.parseInt("123,x");
      } catch (NumberFormatException e) {
         System.out.println("NumberFormatException");
      } catch (ArithmeticException e) {
         System.out.println("ArithmeticException");
      } catch (Exception e) {
         System.out.println("Bloco Exception");
      }
      System.out.println("Fim de programa");
   }
}
```

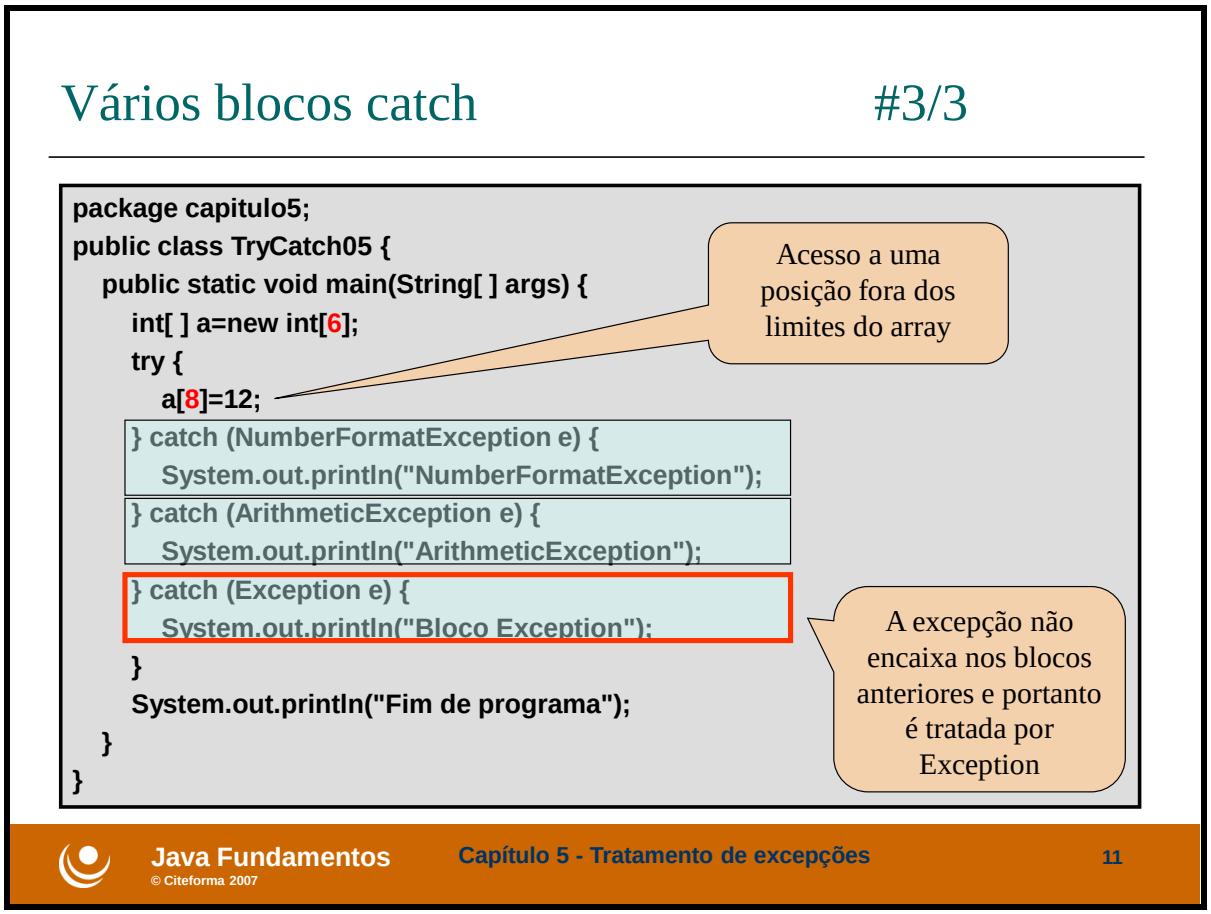
Formato de número inválido

Dos vários blocos este trata a exceção

Neste exemplo o erro é provocado por um formato numérico inválido. Usamos os mesmos blocos catch que no exemplo anterior e o erro será tratado logo pelo primeiro bloco.

A execução produz o seguinte resultado:

```
NumberFormatException
Fim de programa
```



Este slide mostra um novo tipo de erro provocado por um acesso fora dos limites do array. Como não foi declarada uma classe específica para fazer este tratamento, "dispara" o último bloco **Exception**.

A execução produz o seguinte resultado:

```
Bloco Exception
Fim de programa
```

Neste exemplo poderíamos ter usado a classe ***ArrayIndexOutOfBoundsException*** para tratar o erro. Existem outras exceções pré definidas das quais destacamos:

- IndexOutOfBoundsException;
- NegativeArraySizeException;
- ClassCastException;
- IllegalArgumentException;
- IllegalThreadStateException;
- IOException
- SQLException;

finally

finally

```
package capitulo5;
public class Finally01 {
   public static void main(String[ ] args) {
      int a=3,b=0;
      try {
         System.out.println("a/b"+(a/b));
      } catch (Exception e) {
         System.out.println("Bloco Catch");
         System.out.println("  Erro! Divisao por zero !!");
      } finally {
         System.out.println("Bloco Finally");
      }
      System.out.println("Fim de programa");
   }
}
```

Estas instruções são sempre executadas, quer ocorra ou não uma exceção

Java Fundamentos © Citeforma 2007 Capítulo 5 - Tratamento de excepções 12

A cláusula **finally** é utilizada para definir um grupo de instruções que serão sempre executadas depois do grupo **try**, independentemente de ocorrer ou não uma exceção.

A execução do programa acima produz o seguinte resultado:

```
Bloco Catch
  Erro! Divisao por zero !!
Bloco Finally
Fim de programa
```

Este exemplo não mostra claramente a necessidade de usar o bloco **finally**, pois na realidade a mensagem “Fim de programa” aparece quando o erro é tratado, o que nos leva a pensar que as instruções que desejamos que sejam sempre executadas (quer haja erro, quer não) não têm necessariamente que estar no bloco finally, podendo estar depois do **catch**.

No próximo exemplo é tratada a exceção com uma classe diferente daquela que é lançada quando ocorre o erro. Nestas situações o programa interrompe a sua execução, não sendo executadas as instruções finais (“Fim de programa”). No entanto, as instruções incluidas no finally são executadas:

```
package capitulo5;
public class Finally02 {
    public static void main(String[] args) {
        int a=3,b=0;
        try {
            System.out.println("a/b"+(a/b));
        } catch (NumberFormatException e) {
            System.out.println("Erro! Divisao por zero !!");
            System.out.println("Bloco Catch");
        } finally {
            System.out.println("Bloco Finally");
        }
        System.out.println("Fim de programa");
    }
}
```

A execução deste programa produz o seguinte resultado:

```
Bloco Finally
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at capitulo5.Finally02.main(Finally02.java:6)
Java Result: 1
```

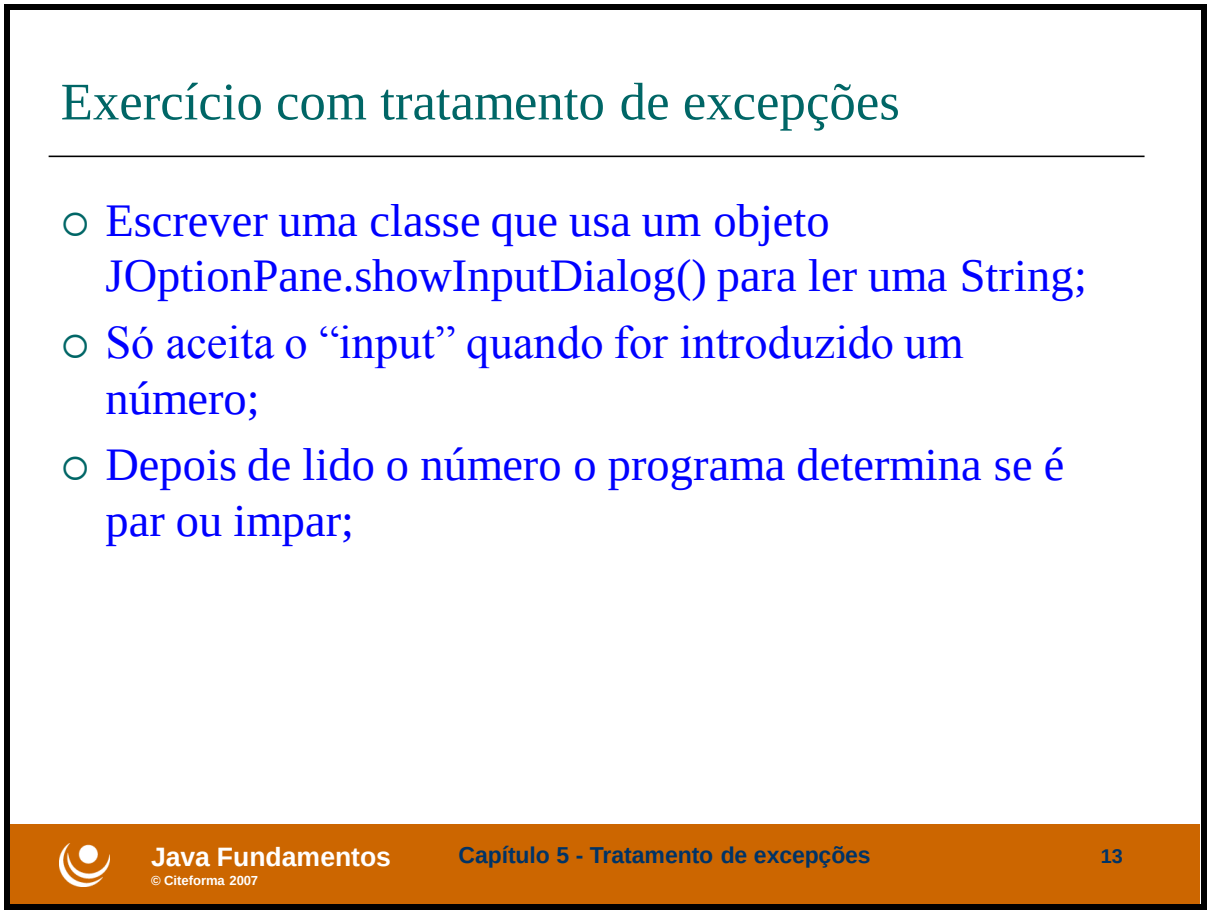
O próximo exemplo mostra uma situação em que o erro é tratado corretamente, mas nesse tratamento força-se a saída do método **main()**, usando a instrução **return**. Mais uma vez não são executadas as instruções finais do método (“Fim de programa"), mas é executado o bloco finally:

```
package capitulo5;
public class Finally03 {
    public static void main(String[] args) {
        int a=3,b=0;
        try {
            System.out.println("a/b"+(a/b));
        } catch (Exception e) {
            return;  //sair
        } finally {
            System.out.println("Bloco Finally");
        }
        System.out.println("Fim de programa");
    }
}
```

A execução deste programa produz o seguinte resultado:

```
Bloco Finally
```

Exercício com tratamento de exceções



O próximo exemplo utiliza o tratamento de exceções para fazer validação de entrada de dados. O programa pede ao utilizador para introduzir um número inteiro. Este fica armazenado numa String que é depois convertida para int. Caso a String não contenha um número válido ocorre uma exceção, cujo tratamento força a repetição da leitura.

```
/*
 * Lê um número e verifica se é par.
 * Usa o tratamento de excepções para validar o input
 */
package capitulo5;
public class TryCatch06 {
  public static void main (String[] args) {
    boolean repetir;
    int n=0;
    do{
        String s = javax.swing.JOptionPane.showInputDialog("Qual o numero?");
        try {
            n = Integer.parseInt(s);
            repetir = false;
        } catch (NumberFormatException e) {
            repetir = true;
        }
    } while (repetir);
    if (n%2==0) {
```

```
      System.out.println ("E' Par");
    } else {
      System.out.println ("E' Impar");
    }
  }
}
```

Este programa utiliza o método javax.swing.JOptionPane.showInputDialog() que serve para abrir uma caixa de diálogo, pedindo ao utilizador um dado através de uma pergunta. Este método pertence ao Swing e o seu funcionamento é visto no curso de Swing.

Escalar exceção não tratada

Sumário

- Exceção;
- try-catch-finally;
- Escalar exceção não tratada;
- Forçar o lançamento de uma exceção;

© Citeforma 2007

O que significa “escalar a exceção”?

O que significa “escalar a exceção”?

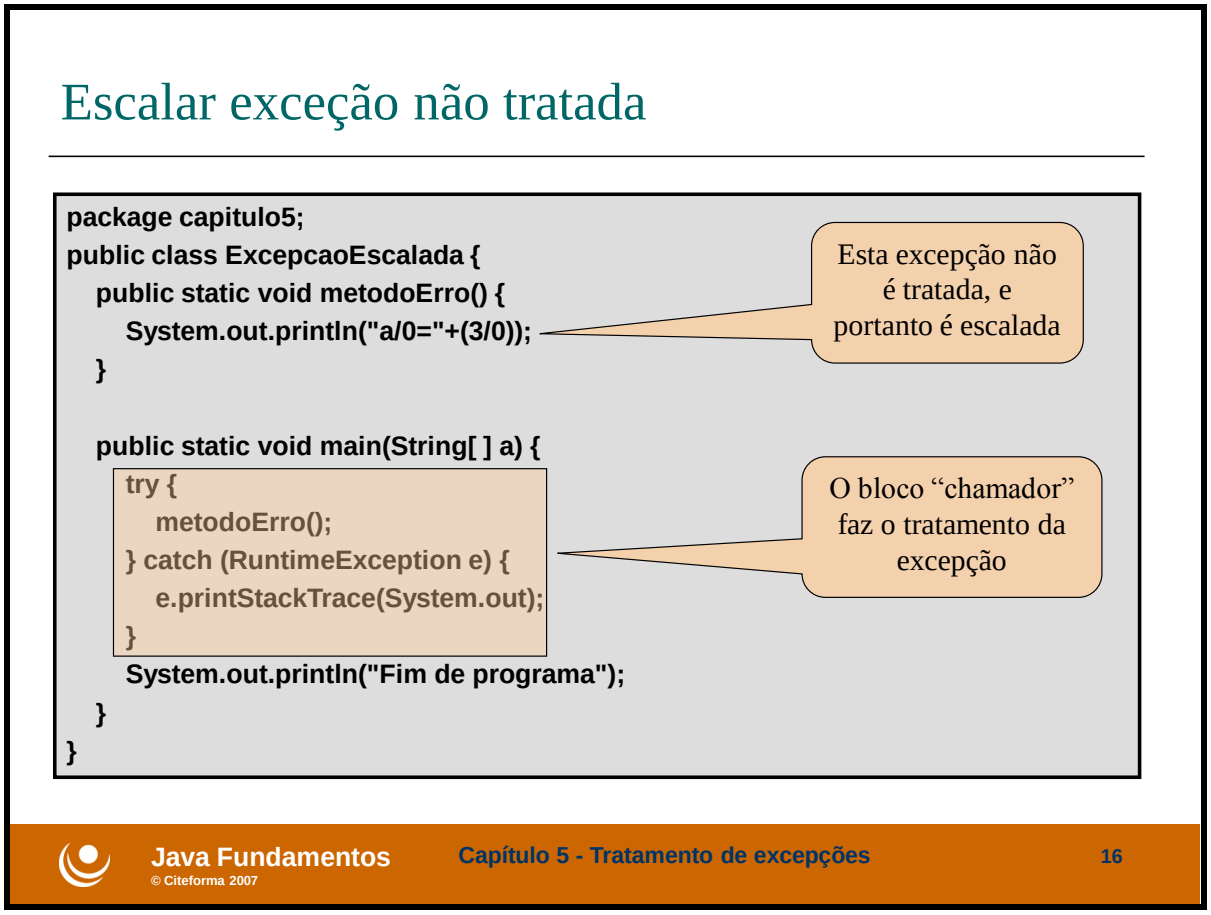
- Em Java as exceções não tratadas são passadas para o nível acima;
- Escalar significa passar a “batata quente para o chefe”;
- O nível acima é o bloco de código “chamador”;
- A exceção escala sucessivamente até ao bloco main().
- Se o bloco main() não trata a exceção então a execução do programa é interrompida;
  - Neste caso o JRE devolve um código de erro ao sistema operativo (normalmente um número diferente de zero);

Java Fundamentos Capítulo 5 - Tratamento de excepções 15

© Citeforma 2007

A linguagem Java usa um mecanismo que escala as exceções não tratadas para o “nível” acima. Quando ocorre uma exceção a primeira verificação é feita no bloco que está em execução. Se não houver tratamento a exceção é enviada para o método onde o bloco está incluído. Se de novo não houver tratamento a exceção é enviada para o método que chamou este, e assim sucessivamente até chegar à thread main(). Se esta também não trata a exceção, então a execução do programa é interrompida, sendo o utilizador informado da causa do erro.

Escalar exceção não tratada



No exemplo acima o método metodoErro() provoca uma divisão por zero, que gera uma exceção que não é tratada. Então a exceção é escalada para o método que invocou metodoErro(), neste caso main(). Este procede ao seu tratamento usando try-catch.

A execução deste programa produz o seguinte resultado:

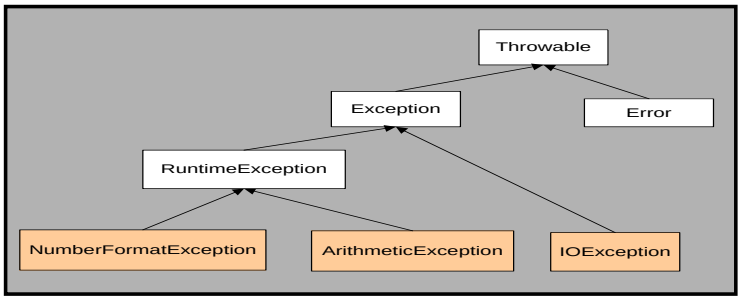
```
java.lang.ArithmeticException: / by zero
        at capitulo5.ExcepcaoEscalada.metodoErro(ExcepcaoEscalada.java:4)
        at capitulo5.ExcepcaoEscalada.main(ExcepcaoEscalada.java:9)
Fim de programa
```

No tratamento da exceção usamos o método printStackTrace() que imprime o conteúdo da pilha de execução, dando-nos informação sobre a sequência de invocação dos métodos. A mensagem informa que o erro ocorreu na linha 4 do método metodoErro(), que por sua vez foi chamado na linha 9 do método main(). O tratamento que fizemos ao erro foi imprimir a pilha de execução, tendo o programa terminado normalmente.

throws - tratar exceção escalando-a

throws - tratar exceção escalando-a #1/2

- Alguns métodos têm grande probabilidade de gerar exceções, pelo que o Java obriga ao seu tratamento;
- O tratamento pode ser feito por try-catch ou throws;
- A diretiva throws não trata a exceção. A sua função é escala-la (enviá-la para o bloco chamador);



© Citeforma 2007

Em Java existem dois tipos de exceções: ***unchecked*** *e* ***checked exception.***

As primeiras descendem da classe **RuntimeException** e o Java não obriga ao seu tratamento, pois são consideradas como tendo menor probabilidade de ocorrer. São exemplos **NumberFormatException** e **ArithmeticException**.

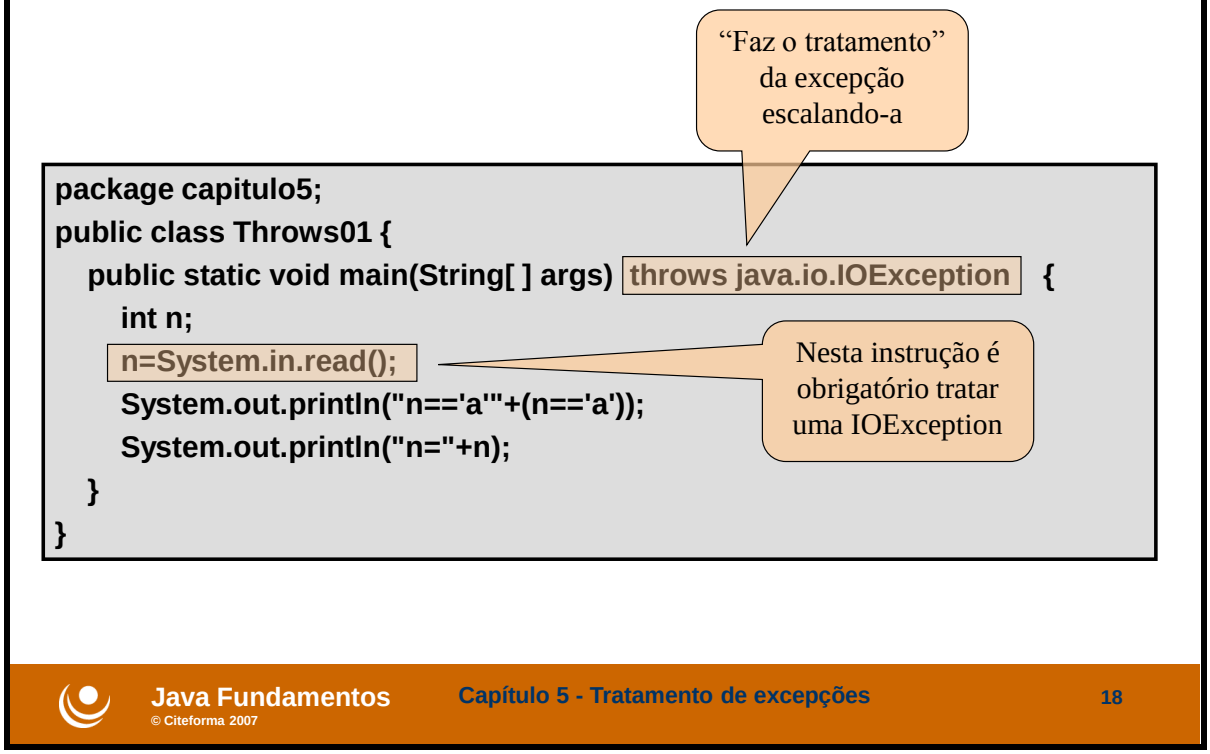
As ***checked exception*** são classes que descendem diretamente de **Exception** e a linguagem obriga ao tratamento destes erros, pois considera que existe elevada probabilidade de ocorrer durante a execução do programa.

O próximo exemplo usa o método **System.in.read()** para fazer a leitura de uma letra a partir do "*standard input*". Este é um desses métodos que tem grande probabilidade de gerar erros em tempo de execução, neste caso provocando uma **IOException**. Nestas situações a linguagem Java obriga ao tratamento da exceção, o que pode ser feito de duas formas: **try-cath** ou **throws**.

Com **try-cath** tentamos resolver o problema. Com **throws** limitamo-nos a escalar o problema, ou seja, enviamos a exceção para o bloco que chamou este. Utilizando **throws** "tranquilizamos" o compilador relativamente à necessidade de tratamento da exceção, sem de facto a termos tratado.

O nome da classe **RuntimeException** não nos parece bem escolhido, já que todas as exceções ocorrem em tempo de execução, mesmo aquelas que hierarquicamente não descendem de **RuntimeException**.

throws - tratar exceção escalando-a #2/2



No exemplo deste slide o método read() pode lançar a exceção **IOException**, que por ser uma ***checked exception*** o Java obriga ao seu tratamento. Para isso ou usamos um bloco try-catch ou escalamos a exceção usando a cláusula **throws.** Nesta segunda alternativa a cláusula é escrita na declaração do método que inclui a instrução perigosa.

Forçar o lançamento de uma exceção

Sumário

- Exceção;
- try-catch-finally;
- Escalar exceção não tratada;
- Forçar o lançamento de uma exceção;

Lançar uma exceção predefinida

Lançar uma exceção predefinida #1/2

- A instrução throw permite ao programador simular a ocorrência de um erro de runtime;
- O programador pode "provocar" erros de runtime em qualquer altura;
- Não confundir throw com throws:
  - O primeiro cria uma exceção;
  - O segundo trata uma exceção;

A instrução **throw** permite ao programador simular a ocorrência de um erro de execução sempre que ele o entender.

Não confundir throw com throws: o primeiro cria uma exceção, o segundo trata uma exceção

Lançar exceção predefinida #2/2

```
package capitulo5;
public class Throw01 {
   public static void main(String[ ] args) {
      int a=2, b=2;
      try {
         System.out.println(a/b);
         throw new ArithmeticException("Fui eu que lancei");
      } catch (Exception e) {
         e.printStackTrace(System.out);
      }
      System.out.println("Fim de programa");
   }
}
```

Força um erro de runtime quando não havia razão para ocorrer

Java Fundamentos © Citeforma 2007 Capítulo 5 - Tratamento de excepções 21

No exemplo acima forçamos uma ArithmeticException numa situação em que não havia razão para ela ocorrer. Como se trata de uma *unchecked exception* não temos que tratar o erro e por isso não indicamos a cláusula **throws** no método onde é lançada.

A execução deste programa produz o seguinte resultado:

```
1
Fim de programa
java.lang.ArithmeticException: Fui eu que lancei
        at capitulo5.Throw01.main(Throw01.java:7)
```

Criar uma exceção

Criar uma exceção

- O programador pode **criar** as suas **próprias** exceções:

Tem que estender a classe Exception

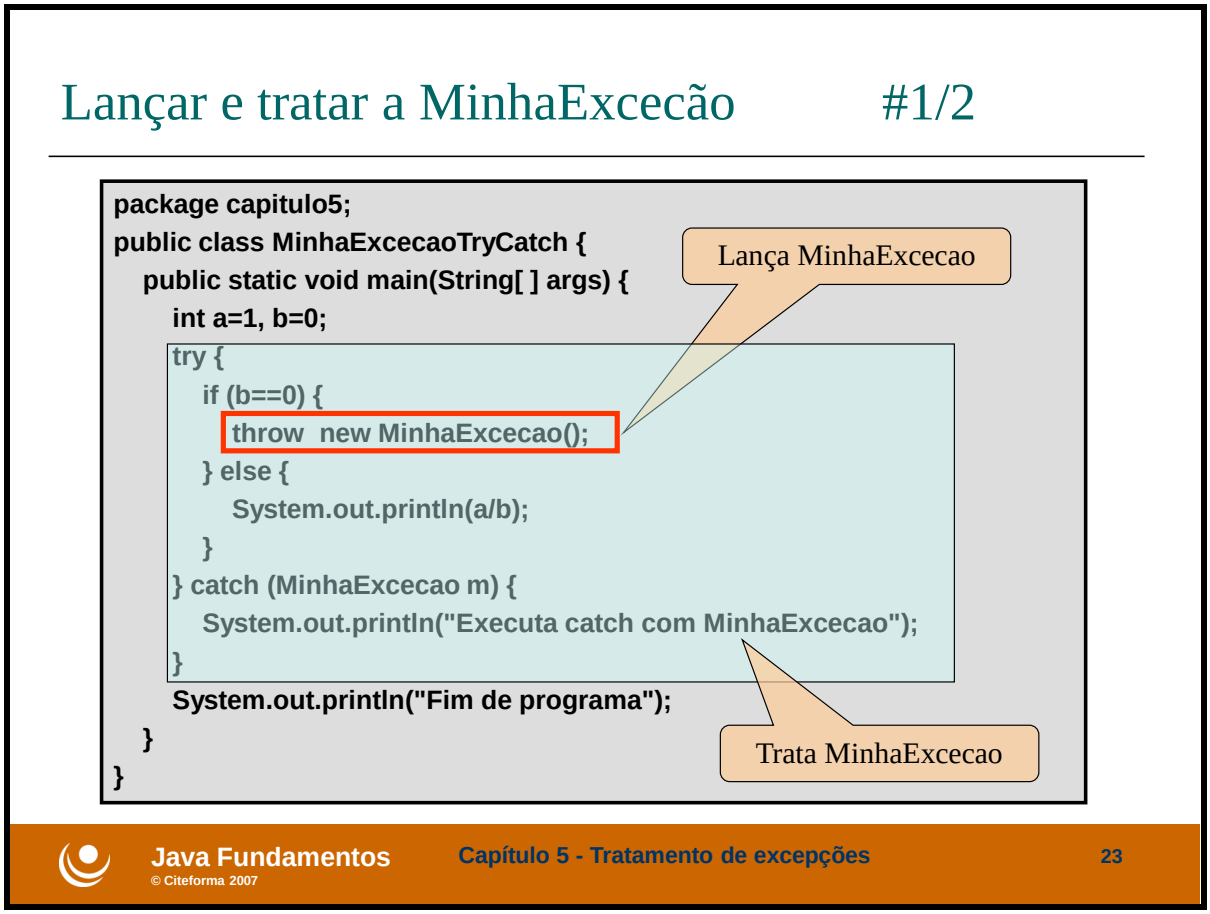
```
package capitulo5;
public class MinhaExcecao extends Exception {
   public MinhaExcecao() {
      super("Criar objeto de MinhaExcecao");
      System.out.println("Criado objeto de MinhaExcecao!");
   }
}
```


© Citeforma 2007

O exemplo do slide cria a minha própria exceção.

O programador pode criar exceções que se ajustam às suas necessidades específicas, por extensão da classe **Exception**. A instrução **throw** permite-lhe forçar o lançamento dessas exceções quando entender, que terão depois que ser tratadas pelos mecanismos tradicionais (try-catch ou throws).

Lançar e tratar a MinhaExcecao



O programa acima lança a MinhaExcepção quando a variável **b** recebe o valor zero. Essa exceção é tratada usando um mecanismo try-catch.

A execução deste programa produz o seguinte resultado:

```
Criado objecto de minha excepcao!
Executa catch com MinhaExcepcao
Fim de programa
```

Lançar e tratar a MinhaExcecão #2/2

Trata MinhaExcepção

```
package capitulo5;
public class MinhaExcecaoThrows {
   public static void main(String[] args) throws MinhaExcecao {
      int a=1, b=0;
      if (b==0) {
         throw new MinhaExcecao();
      } else {
         System.out.println(a/b);
      }
      System.out.println("Fim de programa");
   }
}
```

Lança MinhaExcepção

O exemplo acima lança MinhaExcepcao, mas em vez de a tratar, escala o problema com **throws**.

A execução desse programa produz o seguinte resultado:

```
Criado objecto de minha excepcao!
Exception in thread "main" capitulo5.MinhaExcepcao: Criar objecto de Minha Excepcao
        at capitulo5.MinhaExcepcaoThrows.main(MinhaExcepcaoThrows.java:6)
Java Result: 1
```

Exercício sobre criar uma exceção

Exercício sobre criar uma exceção

- Escrever as classes usadas no exemplo anterior e testar o seu funcionamento;

Java Fundamentos Capítulo 5 - Tratamento de excepções 25

© Citeforma 2007

O exemplo abaixo cria a classe MinhaExceção:

```
package capitulo5;
public class MinhaExcecao extends Exception {
    public MinhaExcecao() {
        super("Criar objeto de MinhaExcecao");
        System.out.println("Criado objeto de MinhaExcecao!");
    }
}
```

O programa seguinte lança a MinhaExceção quando a variável **b** recebe o valor zero. Essa exceção é tratada usando um mecanismo try-catch:

```
package capitulo5;
public class MinhaExcecaoTryCatch {
    public static void main(String[] args) {
        int a=1, b=0;
        try {
            if (b==0) {
                throw  new MinhaExcecao();
            } else {
                System.out.println(a/b);
```

```
            }
        } catch (MinhaExcecao m) {
            System.out.println("Executa catch com MinhaExcecao");
        }
        System.out.println("Fim de programa");
    }
}
```

A execução deste programa produz o seguinte resultado:

```
Criado objecto de minha excepcao!
Executa catch com MinhaExcepcao
Fim de programa
```

O exemplo abaixo lança MinhaExcecao mas em vez de a tratar, escala o problema com **throws**:

```
package capitulo5;
public class MinhaExcecaoThrows {
    public static void main(String[] args) throws MinhaExcecao {
        int a=1, b=0;
        if (b==0) {
            throw new MinhaExcecao();
        } else {
            System.out.println(a/b);
        }
        System.out.println("Fim de programa");
    }
}
```

A execução deste programa produz o seguinte resultado:

```
Criado objecto de minha excepcao!
Exception in thread "main" capitulo5.MinhaExcepcao: Criar objecto de Minha Excepcao
        at capitulo5.MinhaExcepcaoThrows.main(MinhaExcepcaoThrows.java:6)
Java Result: 1
```

Sumário

- Exceção;
- try-catch-finally;
- Escalar exceção não tratada;
- Forçar o lançamento de uma exceção;

© Citeforma 2007