Tratamento de Exceções

Exemplos de exceções

Alguns possíveis motivos externos para ocorrer uma exceção são:

- Tentar abrir um arquivo que não existe.
- Tentar fazer consulta a um banco de dados que não está disponível.
- Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita.
- Tentar conectar em servidor inexistente.

Alguns possíveis erros de lógica para ocorrer uma exceção são:

- Tentar manipular um objeto que está com o valor nulo.
- Dividir um número por zero.
- Tentar manipular um tipo de dado como se fosse outro.
- Tentar utilizar um método ou classe não existentes

Utilizamos um código diferente em Java para tratar aquilo que chamamos de exceções: os casos onde acontece algo que, normalmente, não iria acontecer.

Exceção

Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.

Um dos motivos de o programador Java saber tratamento de exceções é que os métodos de classes definidas na linguagem podem gerar exceções e na maioria das vezes o compilador nos obriga escrever comandos tratadores (blocos try {} catch{}) para chamadas destes métodos.

TRY / CATCH

```
try
{
        //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_exceçao_1 e)
{
        //ação a ser tomada
}
catch(tipo_exceçao_2 e)
{
        //ação a ser tomada
}
catch(tipo_exceçao_n e)
{
```

```
        //ação a ser tomada
}
```

Onde:

- try{ … } - Neste bloco são introduzidas todas as linhas de código que podem vir a lançar uma exceção.
- catch(tipo_excessao e) { … } - Neste bloco é descrita a ação que ocorrerá quando a exceção for capturada.

Porém, apenas para entender o controle de fluxo de uma Exception, vamos colocar o código que vai tentar (try) executar o bloco perigoso e, caso o problema seja do tipo ArrayIndexOutOfBoundsException, ele será pego (caught). Repare que é interessante que cada exceção no Java tenha um tipo... ela pode ter atributos e métodos.

Exemplo:

```
System.out.println("inicio");
int[] array = new int[10];
for (int i = 0; i <= 15; i++) {
  array[i] = i;
  System.out.println(i);
}
System.out.println("fim");
```

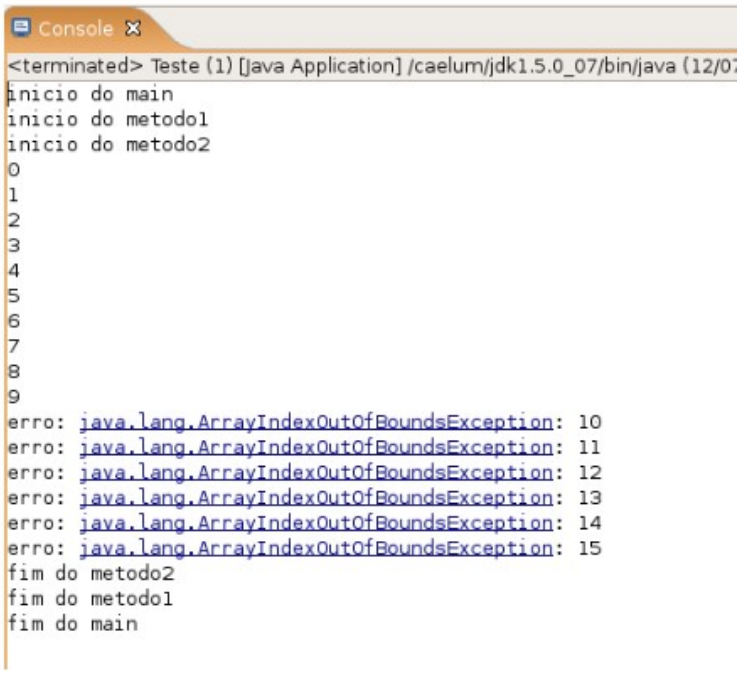
Adicione um try/catch em volta do for, pegando ArrayIndexOutOfBoundsException. O que o código imprime?

```
try {
  for (int i = 0; i <= 15; i++) {
    array[i] = i;
    System.out.println(i);
  }
} catch (ArrayIndexOutOfBoundsException e) {
  System.out.println("erro: " + e);
}
```

Agora, em vez de fazer o try em torno do for inteiro, tente apenas com o bloco de dentro do for:

```
for (int i = 0; i <= 15; i++) {
  try {
    array[i] = i;
    System.out.println(i);
  } catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("erro: " + e);
  }
}
```

Qual é a diferença?



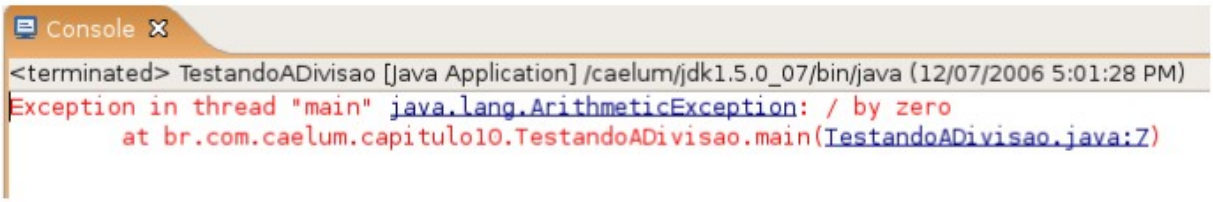
Repare que, a partir do momento que uma exception foi catched (pega, tratada, handled), a execução volta ao normal a partir daquele ponto.

Exceções de Runtime mais comuns

Que tal tentar dividir um número por zero? Será que a JVM consegue fazer aquilo que nós definimos que não existe?

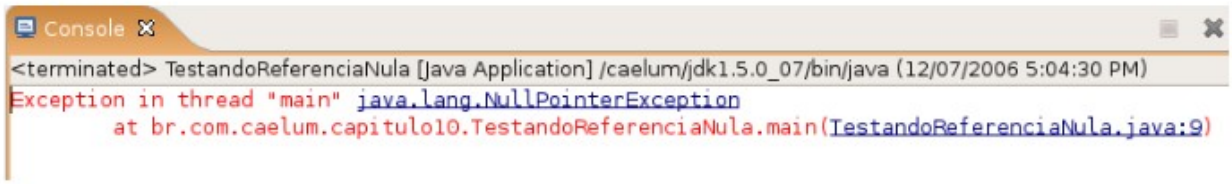
```
public class TestandoADivisao {

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

Tente executar o programa acima. O que acontece?




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

Tente executar este programa. O que acontece?



Repare que um ArrayIndexOutOfBoundsException ou um NullPointerException poderia ser facilmente evitado com o for corretamente escrito ou com ifs que checariam os limites da array.

Outro caso em que também ocorre tal tipo de exceção é quando um cast errado é feito (veremos mais pra frente). Em todos os casos, tais problemas provavelmente poderiam ser evitados pelo programador. É por esse motivo que o java não te obriga a dar o try/catch nessas exceptions e chamamos essas exceções de **unchecked. Em outras palavras, o compilador não checa se você está tratando essas exceções.**

Erros
Os erros em Java são um tipo de exceção que também podem ser tratados. Eles representam problemas na máquina virtual e não devem ser tratados em 99% dos casos, já que provavelmente o melhor a se fazer é deixar a JVM encerrar (ou apenas a Thread em questão).

Outro tipo de exceção: Checked Exceptions

Fica claro, com os exemplos de código acima, que não é necessário declarar que você está tentando fazer algo onde um erro possa ocorrer. Os dois exemplos, com ou sem o try/catch, compilaram e rodaram. Em um, o erro terminou o programa e, no outro, foi possível tratá-lo.

Mas não é só esse tipo de exceção que existe em Java. Um outro tipo, obriga a quem chama o método ou construtor a tratar essa exceção. **Chamamos esse tipo de exceção de checked, pois o compilador checará se ela está sendo devidamente tratada, diferente das anteriores, conhecidas como unchecked.**

Um exemplo interessante é o de abrir um arquivo para leitura, onde pode ocorrer o erro do arquivo não existir:

```
class Teste {
  public static void metodo() {
    new java.io.FileInputStream("arquivo.txt");
  }
}
```

O código acima não compila e o compilador avisa que é necessário tratar o FileNotFoundException que pode ocorrer:

```
Teste.java:3: unreported exception java.io.FileNotFoundException; must be caught
 or declared to be thrown
                new java.io.FileReader("arquivo.txt");
                ^
1 error
```

Para compilar e fazer o programa funcionar, temos duas maneiras que podemos tratar o problema. O primeiro, é tratá-lo com o try e catch do mesmo jeito que usamos no exemplo anterior, com uma array:

```
public static void metodo() {
  try {
    new java.io.FileInputStream("arquivo.txt");
  } catch (java.io.FileNotFoundException e) {
    System.out.println("Nao foi possível abrir o arquivo para leitura");
  }
}
```

A segunda forma de tratar esse erro, é delegar ele para quem chamou o nosso método, isto é, passar para a frente.

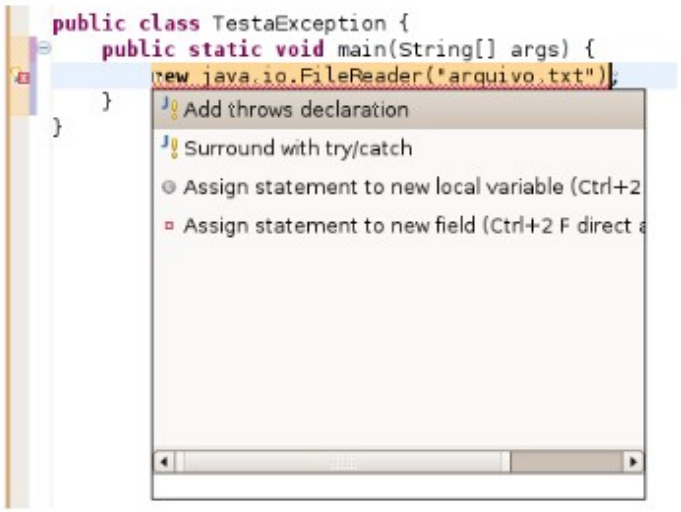
```
public static void metodo() throws java.io.FileNotFoundException {

  new java.io.FileInputStream("arquivo.txt");

}
```

No NetBeans é bem simples fazer tanto um try/catch como um throws:

Tente digitar esse código no NetBeans:

```
public class TestaException {
  public static void main(String[] args) {
    new java.io.FileInputStream("arquivo.txt");
  }
}
```

O NetBeans vai reclamar :



E você tem duas opções:

1. Adicionar a declaração throws, que vai gerar:

```
public class TestaException {
  public static void main(String[] args) throws FileNotFoundException {
    new java.io.FileInputStream("arquivo.txt");
  }
}
```

2. Cercar o código com o with try/catch, que vai gerar:

```
public class TestaException2 {
  public static void main(String[] args) {
    try {
      new java.io.FileInputStream("arquivo.txt");
    } catch (FileNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
}
```

No início, existe uma grande tentação de sempre passar o problema pra frente para outros o tratarem. Pode ser que faça sentido, dependendo do caso, mas não até o main, por exemplo. Acontece que quem tenta abrir um arquivo sabe como lidar com um problema na leitura. Quem chamou um método no começo do programa pode não saber ou, pior ainda, tentar abrir cinco arquivos diferentes e não saber qual deles teve um problema!

Não há uma regra para decidir em que momento do seu programa você vai tratar determinada exceção. Isso vai depender de em que ponto você tem condições de tomar uma decisão em relação àquele erro. Enquanto não for o momento, você provavelmente vai preferir delegar a responsabilidade para o método que te invocou.

Parte 1 do EXERCÍCIO DA SEMANA 9 – ESSE OS ALUNOS DEVERÃO FAZER E ENVIAR NO MOODLE

Crie um projeto no NetBeans chamado TesteExcecao e crie uma aplicação Java e no método método principal **main** que tem como seu único objetivo alterar todas as letras de uma frase para maiúsculas utilizando o método **toUpperCase()** da classe String, caso a frase esteja nula e se tente usar o método toUpperCase() na mesma será lançada uma exceção de NullPointerException.

Primeiro vamos ver como ficaria a tal classe sem a utilização do **try/catch**.

Teste primeiro o Exemplo de código abaixo sem o comando try-catch

```
public class aumentaFrase {
        public static void main(String args[])
        {
                String frase = null;
                String novaFrase = null;
                novaFrase = frase.toUpperCase();
                System.out.println("Frase antiga: "+frase);
                System.out.println("Frase nova: "+novaFrase);
        }
}
```

Quando este código for executado, o mesmo lançará uma **NullPointerException**, como poder ser visto na saída do console quando executamos tal programa.

Saída gerada pelo programa sem try-cach

```
Exception in thread "main" java.lang.NullPointerException
at aumentaFrase.main(aumentaFrase.java:15)
```

Ou seja, o mesmo tentou acessar um atributo de um objeto que estava nulo. Para ajudar a melhorar a situação, deve-se usar o try/catch.

Modifique o programa acima para tratar a exceção usando o comando TRY/CATCH (de acordo com o código abaixo)

```
public static void main(String args[])
{
        String frase = null;
        String novaFrase = null;
        try
        {
                novaFrase = frase.toUpperCase();
        }
        catch(NullPointerException e) //CAPTURA DA POSSÍVEL exceção.
        {
                //TRATAMENTO DA exceção
                System.out.println("O frase inicial está nula, para solucionar o
problema, foi lhe atribuito um valor default.");
                frase = "Frase vazia";
                novaFrase = frase.toUpperCase();
        }
        System.out.println("Frase antiga: "+frase);
        System.out.println("Frase nova: "+novaFrase);
}
```

Quando este código for executado, o mesmo lançará uma NullPointerException, porém esta exceção será tratada desta vez, sendo a mesma capturada pelo **catch{}** e dentro deste bloco as devidas providências são tomadas. Neste caso é atribuído um valor default à variável **frase**.

Até vai a primeira parte do exercício

Comando finally

Imagine a seguinte situação: foi aberta uma conexão com o banco de dados para realizar determinada ação, e no meio deste processo seja lançada alguma exceção, como por exemplo, NullPointerException ao tentar manipular um determinado atributo de um objeto. Neste caso seria necessário que mesmo sendo lançada uma exceção no meio do processo a conexão fosse fechada. Um outro exemplo bom seria a abertura de determinado arquivo para escrita no mesmo, e no meio deste processo é lançada uma exceção por algum motivo, o arquivo não seria fechado, o que resultaria em deixar o arquivo aberto.

Quando uma exceção é lançada e é necessário que determinada ação seja tomada mesmo após a sua captura, utilizamos a palavra reservada finally.

Sintaxe de uso do bloco finally

```
try
{
        //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_exceçao_1 e)
{
        //ação a ser tomada
}
catch(tipo_exceçao_2 e)
{
        //ação a ser tomada
}
catch(tipo_exceçao _n e)
{
        //ação a ser tomada
}
finally
{
        //ação a ser tomada
}
```

Exemplo de Programa aumetaFrase com bloco finally:

```
public class aumentaFrase {
        public static void main(String args[])
        {
                String frase = null;
                String novaFrase = null;
                try
                {
                        novaFrase = frase.toUpperCase();
                }
                catch(NullPointerException e)
                {
                        System.out.println("O frase inicial está nula, para
solucional tal o problema, foi lhe atribuito um valor default.");
                        frase = "Frase vazia";
                }
                finally
                {
                        novaFrase = frase.toUpperCase();
                }
                System.out.println("Frase antiga: "+frase);
                System.out.println("Frase nova: "+novaFrase);
        }
}
```

Quando este código fosse executado, o mesmo lançaria uma NullPointerException, porém esta exceção será tratada desta vez, sendo a mesma capturada pelo catch{} e dentro deste bloco as devidas providências são tomadas. Neste caso é atribuído um valor default à variável frase. Neste exemplo, mesmo o código lançando uma exceção durante a sua execução e a mesma sendo capturada pelo catch, uma determinada ação será tomada no bloco finally, neste caso tanto com a exceção ou não será executada a linha “ novaFrase = frase.toUpperCase();”, tornando todas letras da frase maiúsculas.

Comandos throw e throws

Imagine uma situação em que não é desejado que uma exceção seja tratada na própria classe ou método, mas sim em outro que venha lhe chamar. Para solucionar tal situação utilizamos o comando throws na assinatura do método com a possível exceção que o mesmo poderá a vir lançar.

Sintaxe de declação de método com definição de exceções:

```
tipo_retorno nome_metodo() throws tipo_exceção_1, tipo_exceção_2, tipo_exceção_n
{

…

}
```

Onde:

- tipo_retorno – Tipo de retorno do método.
- nome_metodo() - Nome do método que será utilizado.
- tipo_exceção_1 a tipo_exceção_n – Tipo de exceções separadas por virgula que o seu método pode vir a lançar.

Parte 2 do EXERCÍCIO DA SEMANA 9 – ESSE OS ALUNOS DEVERÃO FAZER E ENVIAR NO MOODLE

Vamos fazer outra solução de tratamento da mesma exceção do exercício parte 1 feito anteriormente, agora não trataremos Listagem 10:Definição de exceções que um método pode gerar

```
public class TesteString {
        private static void aumentarLetras() throws NullPointerException
//lançando excessão
        {
                String frase = null;
                String novaFrase = null;
                novaFrase = frase.toUpperCase();
                System.out.println("Frase antiga: "+frase);
                System.out.println("Frase nova: "+novaFrase);
        }


        public static void main(String args[])
        {
                try
                {
                        aumentarLetras();
                }
                catch(NullPointerException e)
                {
                        System.out.println("Ocorreu um NullPointerException ao
executar o método aumentarLetras() "+e);
                }
        }
}
```