

INSTITUTO MAUÁ DE TECNOLOGIA



Linguagens I

Convenções e organização

Profº. Tiago Sanches da Silva
Profº. Murilo Zanini de Carvalho

Organização

Quando um programador utiliza as classes feitas por outro, surge um problema clássico: como escrever duas classes com o mesmo nome?

Pode ser que a minha classe de **Data** funcione de um certo jeito, e a classe **Data** de um colega, de outro jeito. Pode ser que a classe de **Data** de uma biblioteca do **java** funcione ainda de uma terceira maneira diferente.

Como permitir que tudo isso realmente funcione? Como controlar quem quer usar qual classe de **Data**?



Organização

- Crie um novo projeto: OrganizandoJava
- Crie uma classe chamada Cliente
- Crie uma segunda classe chamada Cliente também

O que aconteceu? Por que?

Pacotes

Pacotes

O que é um pacote?

Um pacote é uma coleção de classes relacionadas que provê acesso protegido e gerenciamento de espaço de nomes (**Namespace**).

Em um **namespace** não pode existir classes, interfaces e arquivos com mesmo nome, ou seja, para criar duas classes Cliente é necessário colocá-las em um **namespace** diferente, em outras palavras, em pacotes diferentes.

Organização - Discussão

- Crie um novo pacote no projeto OrganizandoJava
- Crie finalmente a segunda classe chamada Cliente

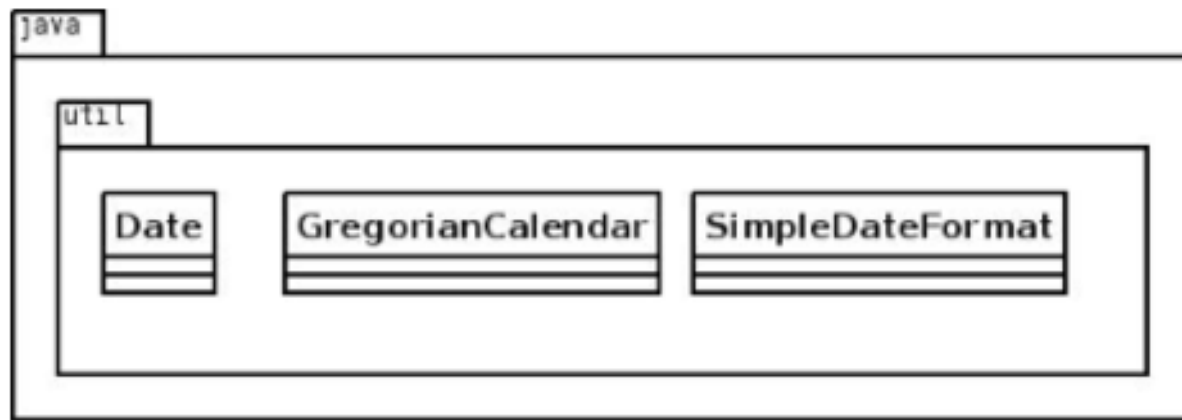
Deu certo?

Verifique a pasta **src** do projeto no sistema de pasta do SO. O que aconteceu?

Pacotes

Os **diretórios** estão diretamente relacionados aos **pacotes** e costumam agrupar classes de funcionalidades similares ou relacionadas.

Por exemplo, no pacote **java.util** temos as classes **Date**, **SimpleDateFormat** e **GregorianCalendar**; todas elas trabalham com datas de formas diferentes.



Padrão de nomenclatura dos pacotes

Padrão de nomenclatura

Padrão da nomenclatura dos pacotes

O padrão da sun para dar nome aos pacotes é relativo ao nome da empresa que desenvolveu a classe:

```
br.com.nomedaempresa.nomedoprojeto.subpacote  
br.com.nomedaempresa.nomedoprojeto.subpacote2  
br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3
```

Os pacotes só possuem letras minúsculas, não importa quantas palavras estejam contidas nele. Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes.

As classes do pacote padrão de bibliotecas não seguem essa nomenclatura, que foi dada para bibliotecas de terceiros.

<https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html>

Organização - Discussão

Fica fácil notar que a palavra chave **package** indica qual o pacote/diretório contém esta classe.

- Crie um novo pacote no projeto **OrganizandoJavaV2**
- Crie o nome do pacote utilizando `br.com.<seuNome>.organizando`
- Em duplas, criem a classe `Cliente` (cada um em seu projeto)
 - Adicione atributo `nome`
 - Um da dupla irá implementar **`getNome()`** o outro **`retornaNome()`**

Fully Qualified Name

Para utilizar uma classe que está em outro pacote eu posso utilizar o ***Fully Qualified Name*** da classe.

<pacote>.<Classe>

Que basicamente é o verdadeiro nome de uma classe, por isso duas classes com o mesmo nome em pacotes diferentes não conflitam.

```
br.com.tiago.lib.Cliente eu = new br.com.tiago.lib.Cliente();
```

Import

Caso você importe o pacote, não precisará se referenciar a classe através do ***Fully Qualified Name***.

```
import br.com.tiago.lib.Cliente;
```

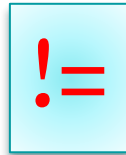
package, import, class

É muito importante manter a ordem! Primeiro, aparece uma (ou nenhuma) vez o **package**; depois, pode aparecer um ou mais **imports**; e, por último, as declarações de classes.

Acesso a classe

As classes só são visíveis para outras no **mesmo pacote** e, para permitir que a classe de Teste veja e acesse a classe **Cliente** em outro pacote, a classe **Cliente** precisa ser **pública**:

```
class Cliente {  
    public String name;  
    public int idade;  
}
```



```
public class Cliente {  
    public String name;  
    public int idade;  
}
```



Vamos trabalhar – Discussão em sala

Vamos utilizar a classe criada pelo colega.

Como importar a classe do colega?

Vamos fazer juntos!



Perguntas?

Exceções no Java

Introdução

Quando se cria programas de computador em Java, há possibilidade de ocorrerem erros imprevistos durante sua execução, esses erros são conhecidos como **exceções** e podem ser provenientes de alguma falha como: comunicação, leitura e escrita de arquivos, entrada de dados inválidos, acesso a elementos fora de índice, entre outros.


Podemos destacar que a **exceção** é um evento não esperado que ocorre no sistema quando está em tempo de execução (**Runtime**). Geralmente quando o sistema captura alguma exceção o fluxo do código fica interrompido.

Introdução

As exceções ocorrem quando algo imprevisto acontece, elas podem ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis.


Exceções - externos

Alguns possíveis **motivos externos** (que não dependem da aplicação) 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.
- 

Exceções - lógica

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.
- 

Tratando as exceções

Antes de resolvermos o nosso problema, vamos ver como a **Java Virtual Machine** age ao se deparar com situações inesperadas, como divisão por zero ou acesso a um índice da **array** que não existe.

Pilha de chamadas

Stack - pilha

Verifique o código 1 enviado pelo slack.

```
public static void main(String[] args) {
    System.out.println("inicio do main");
    metodo1();
    System.out.println("fim do main");
}

static void metodo1() {
    System.out.println("inicio do metodo1");
    metodo2();
    System.out.println("fim do metodo1");
}

static void metodo2() {
    System.out.println("inicio do metodo2");
    int[] array = new int[10];
    for (int i = 0; i <= 15; i++) {
        array[i] = i;
        System.out.println(i);
    }
    System.out.println("fim do metodo2");
}
```

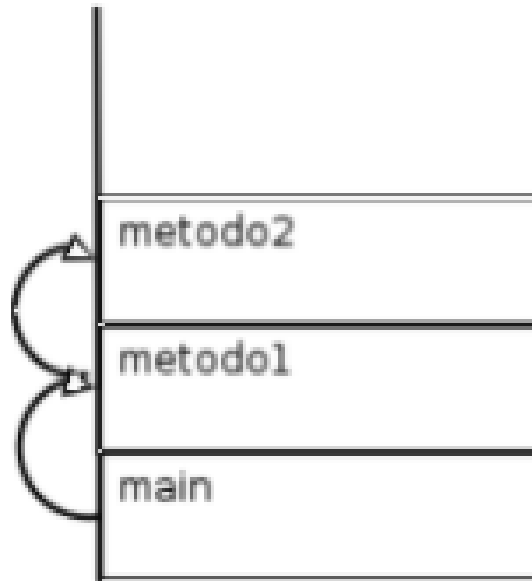
Stack - pilha

Repare o método **main** chamando **metodo1** e esse, por sua vez, chamando o **metodo2**. Cada um desses métodos pode ter suas próprias variáveis locais, isto é: o **metodo1** não enxerga as variáveis declaradas dentro do **main** e por aí em diante.

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    metodo1();  
    System.out.println("fim do main");  
}  
  
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    metodo2();  
    System.out.println("fim do metodo1");  
}  
  
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```


Stack - pilha

Toda invocação de método é empilhada em uma estrutura de dados que isola a área de memória de cada um. Quando um método termina (retorna), ele volta para o método que o invocou. Ele descobre isso através da **pilha de execução (stack)**: basta remover o marcador que está no topo da pilha:



Stacktrace – rastro de pilha

Ao executar o código o que houve?

Qual foi a saída? O que isso representa?

```
inicio do main
```

```
inicio do metodo1
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
```

```
inicio do metodo2
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5|
```

```
6
```

```
7
```

```
8
```

```
9
```

```
at testeerro.TesteErro.metodo2 (TesteErro.java:33)
```

```
at testeerro.TesteErro.metodo1 (TesteErro.java:25)
```

```
at testeerro.TesteErro.main (TesteErro.java:19)
```

Stacktrace – rastro de pilha


Essa é o conhecido **rastro da pilha** (**stacktrace**). É uma saída importantíssima para o programador verificar onde está ocorrendo o problema e quais métodos estão sendo afetados.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
```

```
at testeerro.TesteErro.metodo2 (TesteErro.java:33)
```

```
at testeerro.TesteErro.metodo1 (TesteErro.java:25)
```

```
at testeerro.TesteErro.main(TesteErro.java:19)
```



Stacktrace – rastro de pilha

O sistema de exceções do Java funciona da seguinte maneira: quando uma exceção é lançada (**throw**), a **JVM** entra em estado de alerta e vai ver se o método atual toma alguma precaução ao tentar executar esse trecho de código.

Como o **metodo2** não está tratando esse problema, a **JVM** para a execução dele anormalmente, sem esperar ele terminar, e volta um **stackframe** pra baixo, onde será feita nova verificação: "o **metodo1** está se precavendo de um problema chamado **ArrayIndexOutOfBoundsException**?" "Não...". Volta para o **main**, onde também não há proteção, então a **JVM** mata a Thread corrente.

Try catch finally

Try-catch

Uma maneira de tentar contornar esses imprevistos é realizar o tratamento dos locais no código que podem vir a lançar possíveis exceções, como por exemplo, campo de consulta a banco de dados, locais em que há divisões, consulta a arquivos de propriedades ou arquivos dentro do próprio computador.

Para tratar as exceções em Java são utilizados os comandos try e catch.

Try-catch

```
try
{
    //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_excecao_1 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_2 e)
{
    //ação a ser tomada
}
```

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.

Try-catch

Exemplo de código sem **try-catch**. Testem (código no slack).

```
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);  
    }  
}
```


Try-catch

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

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

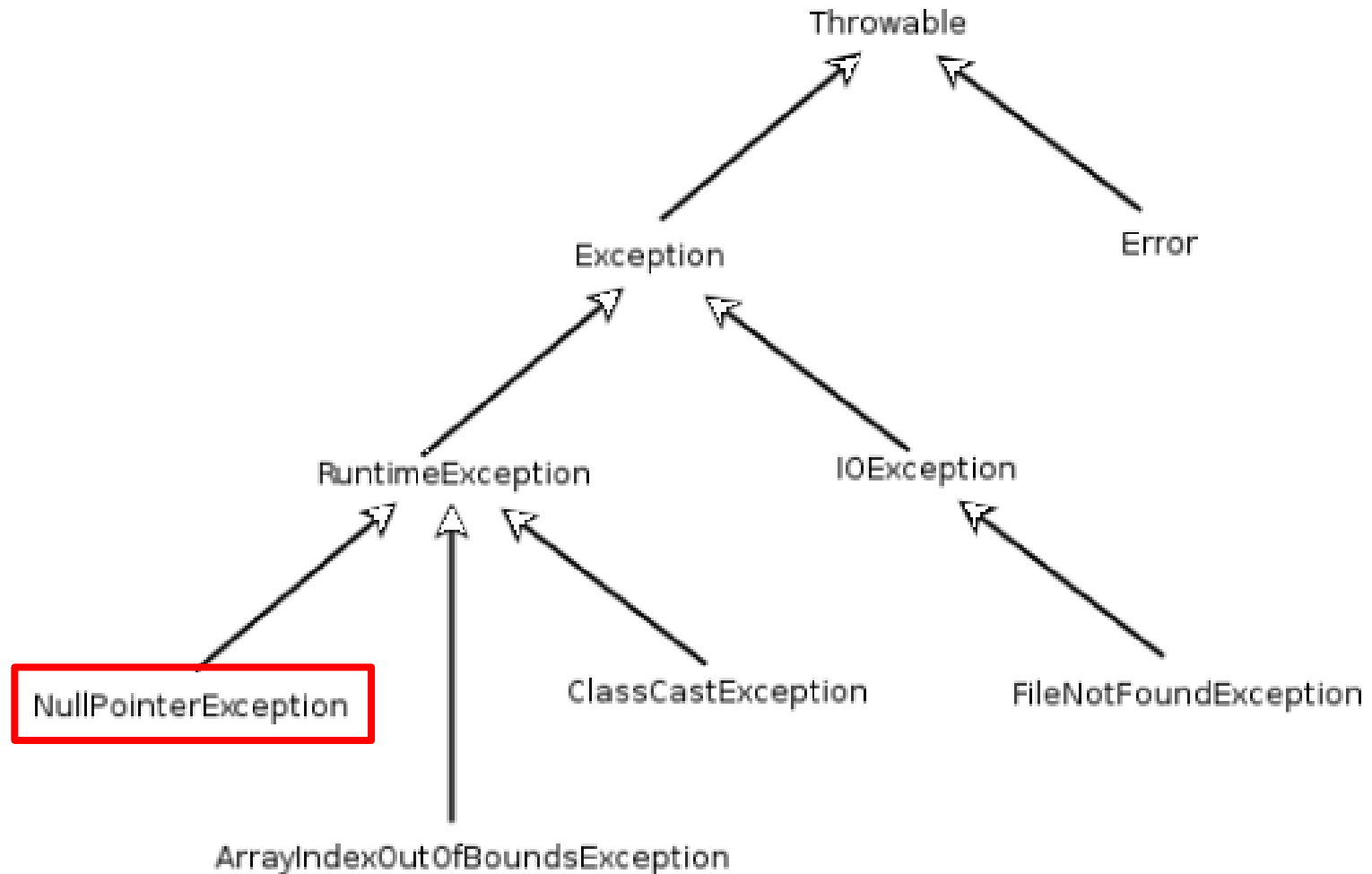
Vamos reformular o código usando o **try-catch**.



Try-catch

```
public static void main(String args[])
{
    String frase = null;
    String novaFrase = null;
    try
    {
        novaFrase = frase.toUpperCase();
    }
    catch (NullPointerException e) // captura
    {
        //tratamento
        System.out.println("O frase inicial está n
        frase = "Frase vazia";
        novaFrase = frase.toUpperCase();
    }
    System.out.println("Frase antiga: "+frase);
    System.out.println("Frase nova: "+novaFrase);
}
```

Hierarquia das classes de exceção



Vamos dar uma olhada na documentação do Java? Como encontrar?

Hierarquia das classes de exceção

<https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/NullPointerException.html>

Qual a diferença entre **Exception** e **Error**? Discuta com o professor.

Try-catch

Vamos voltar ao nosso exemplo do **array**.

Capture a exceção e trate a mesma em torno do **for**, que está no método 2.

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    metodo1();  
    System.out.println("fim do main");  
}  
  
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    metodo2();  
    System.out.println("fim do metodo1");  
}  
  
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```

Try-catch (Discussão em sala)

O que aconteceu de diferente? O erro continua sendo exibido? Por que?

```

início do main
início do metodo1
início do metodo2
0
1
2
3
4
5
6
7
8
9
erro: java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo2
fim do metodo1
fim do main
```

início do main
início do metodo1
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
início do metodo2
0
1
2
3
4
5
6
7
8
9

at testeerro.TesteErro.metodo2 ([TesteErro.java:33](#))
at testeerro.TesteErro.metodo1 ([TesteErro.java:25](#))
at testeerro.TesteErro.main ([TesteErro.java:19](#))

A exceção foi tratada e a JVM não “matou” nosso programa, e o mesmo continuou sua execução.

Try-catch (Discussão em sala)

Agora coloque o **try-catch** dentro **for** exatamente onde irá ocorrer o erro.
O que aconteceu?

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    metodo1();  
    System.out.println("fim do main");  
}  
  
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    metodo2();  
    System.out.println("fim do metodo1");  
}  
  
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```

Try-catch (Discussão em sala)

Agora coloque o try-catch dentro for exatamente onde irá ocorrer o erro.
O que aconteceu?

```
início do main
início do metodo1
início do metodo2
0
1
2
3
4
5
6
7
8
9
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 10
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 11
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 12
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 13
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 14
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 15
fim do metodo2
fim do metodo1
fim do main
```


Try-catch (Discussão em sala)

Agora vamos colocar o tratamento no **método 1** agora, em torno da chamada do **método 2**.

O que aconteceu?

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    metodo1();  
    System.out.println("fim do main");  
}  
  
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    metodo2();  
    System.out.println("fim do metodo1");  
}  
  
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```

Try-catch (Discussão em sala)

Agora vamos colocar o tratamento no **método 1** agora, em torno da chamada do **método 2**.

O que aconteceu?

```
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo1
fim do main
```

Try-catch (Discussão em sala)

Conclusão?

O local onde a exceção é tratada na **stack** influencia no resto da execução do programa?

```
inicio do main  
inicio do metodo1  
inicio do metodo2
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

```
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 7  
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 8  
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 9  
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 9  
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 9  
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 9  
fim do metodo2  
fim do metodo1  
fim do main
```

```
inicio do main  
inicio do metodo1  
inicio do metodo2
```

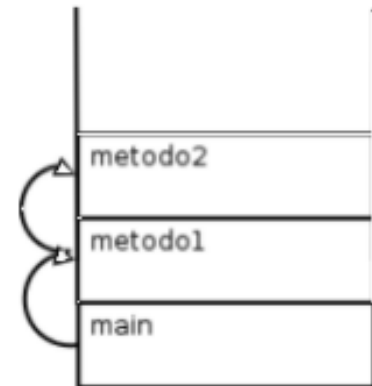
```
0  
1  
2  
3  
4  
5  
6
```

```
7  
8  
9  
erro: java.lang.ArrayIndexOutOfBoundsException: 10  
fim do metodo2  
fim do metodo1  
fim do main
```

```
inicio do main  
inicio do metodo1  
inicio do metodo2
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```


```
Meu erro: java.lang.ArrayIndexOutOfBoundsException: 10  
fim do metodo1
```



Try-catch (Discussão em sala)

Conclusão

A partir do momento que uma **exception** foi **capturada**(caught, pega, tratada, handled), a execução volta ao normal **a partir daquele ponto**.



E onde está o finally?

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.

Comando finally

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

```
try
{
    //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_excecao_1 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_2 e)
{
    //ação a ser tomada
}
catch(tipo_excecao _n e)
{
    //ação a ser tomada
}
finally
{
    //ação a ser tomada
}
```

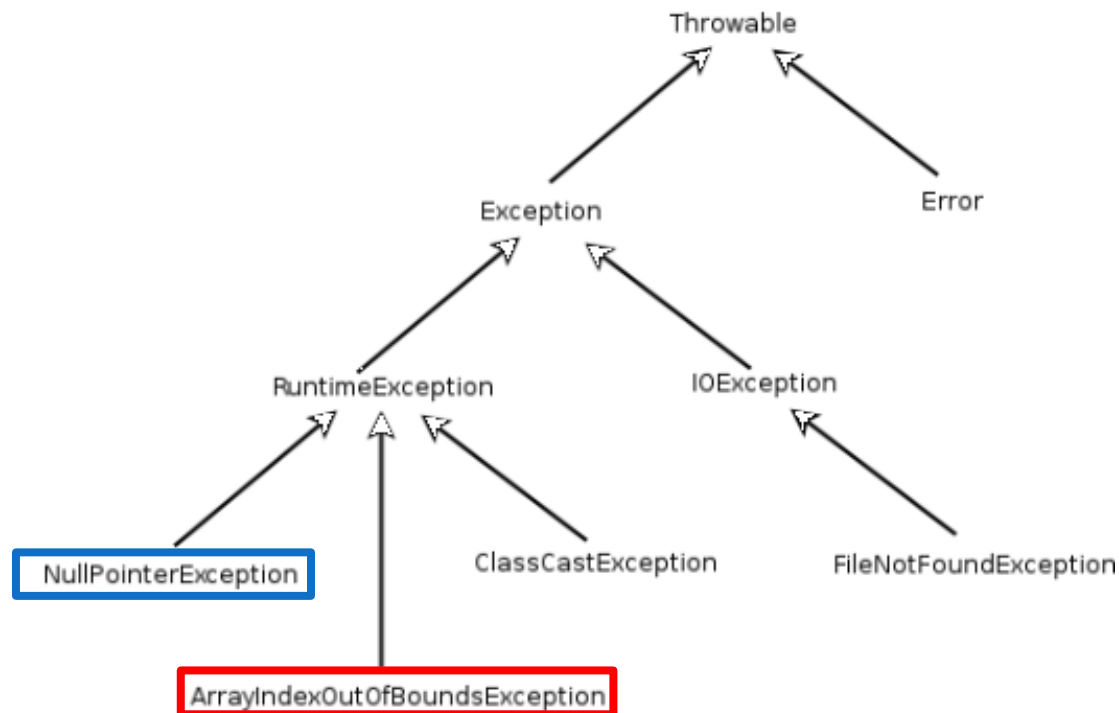
Comando finally (Discussão em sala)

Me de mais um exemplo de uso do **finally**.

Mais um pouco sobre exceção

Hierarquia das classes de exceção

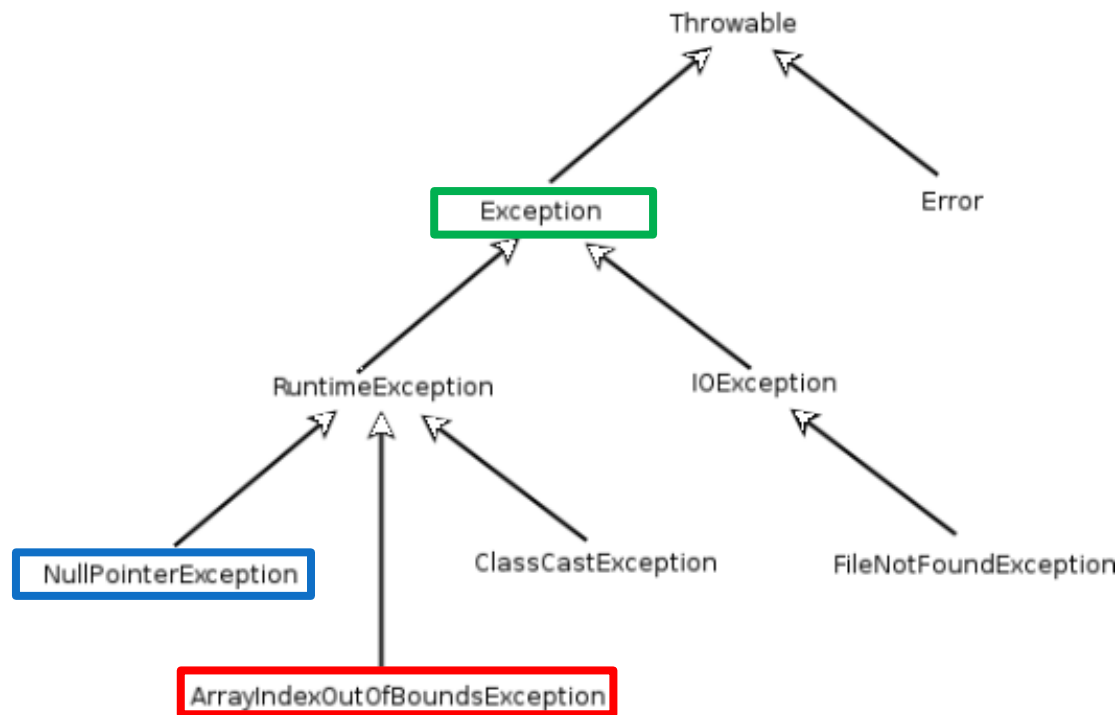
```
try
{
    //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_excecao_1 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_2 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_n e)
{
    //ação a ser tomada
}
finally
{
    //ação a ser tomada
}
```



E se eu quiser tratar todas as exceções possíveis?

Hierarquia das classes de exceção

```
try
{
    //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_excecao_1 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_2 e)
{
    //ação a ser tomada
}
catch(tipo_excecao_n e)
{
    //ação a ser tomada
}
finally
{
    //ação a ser tomada
}
```



E se eu quiser tratar todas as exceções possíveis?

Throw exceptions

Lançar exceções - Throw exceptions

Volte no exemplo do **array**, e coloque um tratamento de exceção em torno do **for** do **metodo2** e também no **metodo1** em torno da chama do **metodo2()**.

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    metodo1();  
    System.out.println("fim do main");  
}  
  
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    metodo2();  
    System.out.println("fim do metodo1");  
}  
  
static void metodo2() {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    for (int i = 0; i <= 15; i++) {  
        array[i] = i;  
        System.out.println(i);  
    }  
    System.out.println("fim do metodo2");  
}
```

Lançar exceções - Throw exceptions

Volte no exemplo do **array**, e coloque um tratamento de exceção em torno do **for** do **metodo2** e também no **metodo1** em torno da chama do **metodo2()**.

```
static void metodo1() {
    System.out.println("inicio do metodo1");
    try {
        metodo2();
    } catch (Exception e) {
        System.out.println("tratamento do erro no método 1: " + e);
    }
    System.out.println("fim do metodo1");
}

static void metodo2() {
    System.out.println("inicio do metodo2");
    int[] array = new int[10];
    try {
        for (int i = 0; i <= 15; i++) {

            array[i] = i;
            System.out.println(i);

        }
    } catch (Exception e) {
        System.out.println("Erro dentro do método 2 : " + e);
    }
    System.out.println("fim do metodo2");
}
```

Lançar exceções (Discussão em aula)

Qual o resultado?

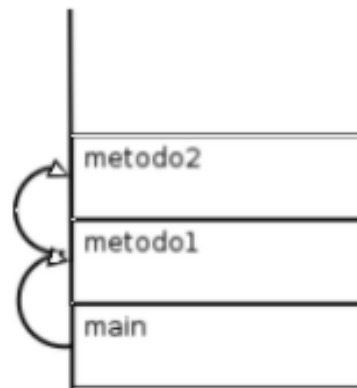
- Somente um tratamento foi executado?
- Qual tratamento foi executado?
- Ambos tratamentos foram executados?

```
inicio do main
inicio do metodo1
inicio do metodo2
0
1
2
3
4
5
6
7
8
9
Erro dentro do método 2 : java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo2
fim do metodo1
fim do main
```

Lançar exceções - Throw exceptions

Em alguns casos é preciso que os métodos ou classes que estão utilizando o método onde pode ocorrer a exceção, também saiba que a exceção ocorreu para que o fluxo seja diferente.

Nestes casos podemos apesar de tratar a exceção localmente podemos lançar a exceção para o próximo método da pilha (que realizou a chamada) utilizando o **throw**.



Lançar exceções - Throw exceptions

Para um método poder lançar uma exceção é necessário informar na assinatura do mesmo todas as exceções que o método poderá eventualmente lançar.

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

E no tratamento da exceção basta utilizar:

```
catch(tipoExceção_1 e)
{
    throw new novoTipoExcecao(e);
}
```

Apenas se for um novo tipo de erro.

Lançar exceções - Throw exceptions

Exemplo lançando o mesmo tipo de erro:

```
static void metodo1() {  
    System.out.println("inicio do metodo1");  
    try {  
        metodo2();  
    } catch (Exception e) {  
        System.out.println("tratamento do erro no método 1: " + e);  
    }  
    System.out.println("fim do metodo1");  
}
```

```
static void metodo2() throws ArrayIndexOutOfBoundsException {  
    System.out.println("inicio do metodo2");  
    int[] array = new int[10];  
    try {  
        for (int i = 0; i <= 15; i++) {  
            array[i] = i;  
            System.out.println(i);  
        }  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("Erro dentro do método 2 : " + e);  
        throw (e);  
    }  
    System.out.println("fim do metodo2");  
}
```

Posso criar minha própria exceção?

Criando exceções

Assim como qualquer objeto, em Java também é possível criar suas próprias exceções. Imagine um cenário em que nenhuma exceção existente faça sentido para ser lançada por você.

Imagine que por algum motivo você precisa que uma exceção seja lançada quando a letra “B” ou “b” não existe e determinada frase, como **não existe nenhuma exceção específica para este caso** **será necessário criar uma exceção**.

Criando exceções

Criando uma exceção para ser lançada toda vez que uma letra “b” ou “B” não é encontrada em uma determinada frase.

```
public class SemLetraBException extends Exception {  
    @Override  
    public String getMessage() {  
        return "Não existe letra B em sua frase";  
    }  
}
```

Utilizando exceções customizadas

Abaixo segue um exemplo que é utilizada a exceção criada anteriormente.

```
public static void main(String[] args) {  
    System.out.println("inicio do main");  
    //metodo1();  
    try {  
        metodoFrase();  
    } catch (Exception e) {  
        System.out.println("Deu erro no metodoFrase-detalhes: " + e);  
    }  
  
    System.out.println("fim do main");  
}  
  
static void metodoFrase() throws SemLetraBException {  
    String frase = "Sou um teste!";  
    if (!frase.contains("b") || !frase.contains("B")) {  
        throw new SemLetraBException();  
    }  
}
```

Exception

Checked exception

Exception

Como criar uma exceção customizada?

Exception

Exercício pratico

Perguntas?

Referências

- Deitel
 - DevMedia (Robson Fernando)
 - Oracle
 - Caelum
- 