JAVA – TRATAMENTO DE EXCEÇÕES

- TRATAMENTO DE ERROS TRADICIONAL
  - O Erro é verificado e caso ocorra, é tratado no ponto é possível a sua ocorrência.
  - Ocorre uma mistura entre o código para resolver o problema e o código utilizado no tratamento do erro
  - Tratamento do erro, interfere na lógica do problema e alguns casos torna mais díficil a compreensão do mesmo.
  - Em algumas situações o erro deve ser propagado por vários níveis até que seja tratado.

TRATAMENTO DE EXCEÇÕES

- TRATAMENTO DE ERROS TRADICIONAL
  - Exemplo – Código sem tratamento de erros!

```
ler_um_arquivo
{
  abrir_arquivo;
  determinar_seu_tamanho;
  alocar_memória;
  ler_arquivo;
  fechar_arquivo;
}
```

TRATAMENTO DE EXCEÇÕES

- TRATAMENTO DE ERROS TRADICIONAL - Exemplo

```
errorCode ler_um_arquivo{
  errorCode = 0;
  abrir_arquivo;
  if (arquivo_aberto){

      determinar_seu_tamanho;
    if (tamanho_disponivel){
      alocar_memória;
      if (memoria_alocada){
        ler_arquivo;
```

```
if (erro_Leitura)
          errorCode = -4;
       } else
        errorCode = -3;
    } else
     errorCode = -2;
    fechar_arquivo;
    if (erro_Fechar_arquivo)
      errorCode = -5
  } else
     errorCode = -1;
  return errorCode; }
```


TRATAMENTO DE EXCEÇÕES

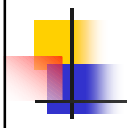
- TRATAMENTO DE ERROS TRADICIONAL - Exemplo

```
errorCode leArquivo{
  errorCode = 0;
  abrir_arquivo;
  if (arquivo_nao_aberto)
    return -1;
  determinar_seu_tamanho;
  if (tamanho_nao_disponivel) {
    fechar_arquivo;
    return -2;}
  alocar_memória;
  if (memoria_nao_alocada){
    fechar_arquivo;
    return -3;}
```

```
ler_arquivo;
  if (erro_Leitura){
    fechar_arquivo;
    return -4;}
  fechar_arquivo;
  if (erro_Fechar_arquivo)
     return -5
  return errorCode;}
```

TRATAMENTO DE EXCEÇÕES

- EXCEÇÃO
  - Indicação de um problema ocorrido durante o processamento
- TIPOS DE EXCEÇÃO
  - Erros de hardware; divisão por zero; tentativa de acessar elementos fora dos limites de um vetor; valores de parâmetros inválidos em um método; esgotamento da memória; utilizar um objeto não criado; etc.



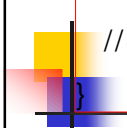
TRATAMENTO DE EXCEÇÕES

- VANTAGENS
  - Separação do código de tratamento de erro do código do programa
  - Propagação do erro através da pilha de funções

```
metodo1 {
 int error;
 error = metodo2();
 if (error)
   //ProcessaErro
 else
   //continua
}
```

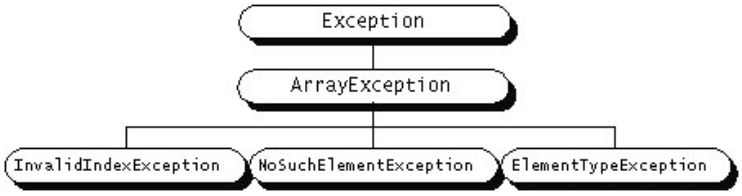
```
int metodo2() {
  int error;
  error = metodo3();
  if (error)
    return error;
  else
   //continua
}
```

```
int metodo3() {
  int error;
  error = leArquivo();
  if (error)
    return error;
  else
   //continua
}
```



TRATAMENTO DE EXCEÇÕES

- VANTAGENS
  - Agrupamento dos vários tipos de erros em classes



TRATAMENTO DE EXCEÇÕES

- QUANDO UTILIZAR
  - Em situações onde o método é incapaz de completar sua tarefa
  - Em situações onde o método não trata as exceções ocorridas, mas apenas sinaliza sua ocorrência, como em bibliotecas de classes.

TRATAMENTO DE EXCEÇÕES

- COMO UTILIZAR
  - Para utilizar o tratamento de exceções, basicamente é necessário "ouvir e capturar" as possíveis exceções que podem ocorrer e além disso "disparar" exceções.
  - Para "ouvir/capturar" utiliza-se os blocos "**`try/catch`**"
  - Para "disparar" exceções utiliza-se as instruções "throws/trow"
  - Além disso é possível criar classes específicas para o tratamento de exceções

OUVINDO / CAPTURANDO EXCEÇÕES

- As exceções que poderão ocorrer serão "ouvidas" através de um bloco que inicia pela instrução **`try`**
- Ao final deste bloco, um ou mais, blocos que iniciam pela instrução **`catch`**, irão "capturar" e tratar as exceções que podem ter sido disparadas.
- Após o último bloco **`catch`**, um bloco **`finally`** opcional fornece um código que sempre será executado e pode ser utilizado para evitar perdas de recurso (Ex. Arquivo aberto)
- Logo que uma exceção ocorre o processamento deixa o bloco **`try`** e começa a pesquisar nos blocos **`catch`** um tratamento específico para aquele tipo de erro

OUVINDO / CAPTURANDO EXCEÇÕES

- Exemplo:

```
try{
    ... }
catch(ExceptionType1 e){
    ... }
catch (ExceptionType2 e){
    ... }
catch (Exception e){
    ... }
finally {
    ...
}
```


DISPARANDO EXCEÇÕES

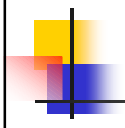
- Para que um método possa disparar uma exceção é necessário colocar a cláusula **`throws`** na definição do mesmo, indicando quais tipos de exceção o mesmo pode retornar
- O método que irá retornar a exceção deve criar a mesma. Uma exceção é também um objeto!
- Após criar a exceção a mesma deve ser disparada com a cláusula **`throw`**
- A classe base de exceções em java é a classe **`Exception`**

CRIANDO CLASSES DE EXCEÇÕES

- Para criar uma classe que será responsável pelo tratamento de exceções a mesma deve extender a classe **`Throwable`**

Exemplo:

```
public class MyException extends
    Throwable
```



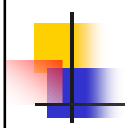
CRIANDO CLASSES DE EXCEÇÕES

- Importante:

catch (Exception e) → Captura todas as exceções

catch (Error err) → Captura todos os erros do tipo Error

catch (Throwable t) → Captura todos os erros do tipo Error e Exception



TRATAMENTO DE EXCEÇÕES - EXEMPLO

```
private double calculaTotal(String sC, String sJ, String sP, String sM) throws
        NumberFormatException {
  if (sC.equals(""))
   throw new NumberFormatException("O CAMPO CAPITAL ESTÁ VAZIO");
  if (sJ.equals(""))
    throw new NumberFormatException("O CAMPO JUROS ESTÁ VAZIO");
  if (sPrazo.equals(""))
    throw new NumberFormatException("O CAMPO ANOS ESTÁ VAZIO");
  //converte o texto para double
  dCapital = Double.parseDouble(sC);
  dJuros = Double.parseDouble(sJ);
  dPrazo = Double.parseDouble(sP);
  dTotal = 0;
  if (dPrazo == 0)
    throw new NumberFormatException("O NÚMERO DE ANOS É IGUAL A ZERO");
  if (sMetodo.equals(aMetodos[0])){
    dTotal = ((dCapital * (1 + (dJuros/100)*dPrazo))/(dPrazo))/12;
  if (sMetodo.equals(aMetodos[1])){
    dTotal = ((dCapital * Math.pow((1 + (dJuros/100)),dPrazo))/dPrazo)/12;
  }
  return dTotal;
}
```


TRATAMENTO DE EXCEÇÕES - EXEMPLO

```
try {
  dTotal = calculaTotal(txtCapI.getText(),txtJur.getText(), txtPra.getText(),sMetodoCalculo);
  ...
}
catch (NumberFormatException ex)
{
  JOptionPane.showMessageDialog(null,ex);
  JOptionPane.showMessageDialog(null,ex.getMessage());
}
```

TRATAMENTO DE EXCEÇÕES - EXEMPLO

```
try {
  dTotal = calculaTotal(txtCapl.getText(),txtJur.getText(), txtPra.getText(),sMetodoCalculo);
  ...
}
catch (NumberFormatException ex)
{
  JOptionPane.showMessageDialog(null,ex);
  JOptionPane.showMessageDialog(null,ex.getMessage());
}
```


ASSINATURA COMPLETA DE UM MÉTODO

- A assinatura de um método é criada da seguite forma:

[*nívelAcesso*] [*modificador*] *tRetorno* **nomeMetodo** (T0 P0,...,Tn Pn)
[***throws*** *tExceção*]

[*nívelAcesso*] - `private`; `public` ou `Protected`

[*modificador*] – Permite alterar a forma como o método se comporta (`static; abstract; final; native; synchronized`)

[*tRetorno*] - Tipo de retorno. Se não houver nenhum retorno, deve ser utilizado **`void`**. O tipo pode ser básico ou então qualquer objeto da linguagem

T0,...Tn – Tipos dos parâmetros utilizados

P0,...,Pn – Parâmetros

[***throws*** *tExceção*] – Tipo de exceção que pode ser disparada pelo método

ESTRUTURA COMPLETA DE UMA CLASSE

```
[nívelAcesso] [modificadorClasse] class NomeClasse
[extends baseClass] [implements interface1, interface2,...] {
    [nívelAcesso] [modificadorAtributo] tipo variavel_membro1;
    [nívelAcesso] [modificadorAtributo] tipo variavel_membroN;
    //Construtores
    [nívelAcesso] NomeClasse(T0 P0, ..., Tn Pn);
    //Destrutor
    protected void finalize();
    //Modificadores
    [nívelAcesso] [modificador] tRetorno setXXX(T0 P0,...,Tn Pn) ...
    //Acessores
    [nívelAcesso] [modificador] tRetorno getXXX(T0 P0,...,Tn Pn) ...
    //Outros métodos (privados; protegidos e públicos)
    [nívelAcesso] [modificador] tRetorno metodo(T0 P0,...,Tn Pn)
    [throws tExceção]
    }
```


ESTRUTURA COMPLETA DE UMA CLASSE

- A seguir é mostrado os possíveis valores utilizados na construção da classe:

  [*nívelAcesso*] - `private`; `public` ou `Protected`

  [*modificadorClasse*] - Permite alterar a forma como a classe se comporta (`abstract`;`final`)

  baseClass – Classe base utilizada pela herança

  interfacei.. – Nome da interface implementada

  [*modificadorAtributo*] - Permite alterar a forma como o atributo se comporta (`static`; `abstract`; `final`; `transient`; `volatile`)

  [***tipo***] – Tipo da variável membro ou atributo. O tipo pode ser básico ou então qualquer objeto da linguagem

ESTRUTURA COMPLETA DE UMA CLASSE

[*modificador*] – Permite alterar a forma como o método se comporta (`static; abstract; final; native; synchronized`)

[***tRetorno***] - Tipo de retorno do método. Se não houver nenhum retorno, deve ser utilizado **`void`**. O tipo pode ser básico ou então qualquer objeto da linguagem

T0,...Tn – Tipos dos parâmetros utilizados

P0,...,Pn – Parâmetros

[***throws*** *tExceção*] – Tipo de exceção que pode ser disparada pelo método

PACOTES

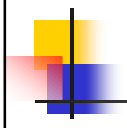
- Um pacote (package) consiste em um conjunto de classes e interfaces, relacionados entre si e normalmente sua criação está ligada a composição de bibliotecas
- Através da utilização de pacotes é possível utilizar classes com mesmo nome, porém proveniente de diferentes pacotes.
- Por convenção os pacotes devem ter o seguinte nome: com.company.package
- Toda classe pertence a um pacote, quando o nome do pacote não é informado, o compilador considera que classe pertence ao pacote "default" que neste caso é o diretório corrente.

PACOTES

- Para criar um pacote basta colocar a seguinte declaração no arquivo .java:

```
package nomeDoPacote
```

- Esta declaração deve ser a primeira declaração existente no arquivo.
- Somente as classes, métodos e variáveis públicas (public) podem ser utilizadas externamente ao pacote.



PACOTES

- Para organizar os arquivos em um pacote a regra é a seguinte:
  - Criar cada classe ou interface em um arquivo .java diferente.
  - Colocar estes arquivos em uma hierarquia de diretório equivalente à estrutura de nome do pacote

Exemplo:

Nome do pacote – com.autoenge.people

Arquivos: **pessoa.java; cliente.java; empregado.java; vendedor.java; gerentel nt.java; gerenteVendas.java**

Diretório onde estão os arquivos: / **com**/ **autoenge**/ **people**

PACOTES

- Uma forma para utilizar uma classe contida em um pacote é qualificando seu nome completamente:

  ```
  java.io.File
  com.autoenge.people.vendedor
  ```

  Exemplo:

  File d = new java.io.File("fname.txt");

- Para evitar a qualificação completa do nome do pacote basta utilizar a keyword "**import**"

  ```
  import nomeCompletoDoPacote.className
  ```

- Para importar todas as classes existentes em um pacote utiliza-se a seguinte sintaxe:

  ```
  import nomeCompletoDoPacote.*
  ```


PACOTES

- Dois pacotes são automaticamente carregados quando se trabalha com a linguagem java:
  - O pacote java.lang
  - O pacote default (diretório corrente)
- Exemplo

  import java.io.*;

  File d = new File("fname.txt")