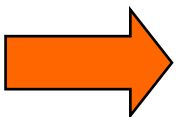


Tratamento de Exceção

Parte I

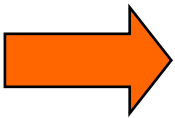
Introdução

- Quando desenvolvemos software, estamos sujeitos a muitos erros;
 - Muitos erros podem ser identificados no momento da compilação:
 - Sintaxe incorreta;
 - Identificado (variável,método,etc.) desconhecido;
 - Classe não encontrada;
 - etc.
- Porém alguns erros ocorrem somente durante a execução;
 - Podem se *bugs* :
 - Cálculos incorretos, trecho de código não implementado,etc.;
 - Podem se condições excepcionais:
 - Falha no sistema de arquivos, entrada de dados inválida,etc.;
- Em vista disso, como podemos lidar com essas situações...? Quem poderá nos ajudar..... 😊



Tratamento de Exceções

- Para lidar com essas situações nós usamos....
 -o tratamento de exceções
- Sua função é:
 - Transferir o controle de onde ocorreu alguma condição anormal
 - Para um manipulador que possa lidar com a situação.
 - Em outras palavras...
 - ... Ter a capacidade de tratar de algum problema que ocorreu durante a execução do programa.

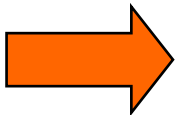


Tratamento de Exceções

- Esse tipo de abordagem permite a construção de sistemas:
 - Mais claros;
 - Organização do código para o tratamento das situações anormais
 - Mais robustos;
 - Possibilidade de tratar os problemas quando ocorre, em vez de simplesmente terminar a execução
 - Mais tolerantes a falhas.
 - Permite detectar e contornar os problemas que possam ocorrer.

Erros e Exceções

- Só que existem erros e erros....
 - Alguns mais críticos outros menos críticos....
- Da mesma forma que existe uma distinção sutil entre risco e traço,
 - Podemos realizar uma distinção entre erro e exceção..
 - Java usa essa distinção para modelar as suas classes...



Diferença entre Erros e Exceções

- Exceção

- Condições de erros **suaves** que o seu programa pode encontrar.

- Em vez de deixar o programa terminar,

- Você pode escrever código para manipular as suas exceções e continuar a sua execução.

- Ex: IOException
- Classe Exception

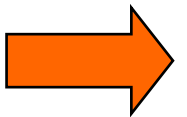
- Erros

- Condições de erros **sérias** que o seu programa pode encontrar.

- Um erro é algo que não se pode recuperar.

- Ou seja, é melhor deixar o programa terminar.

- Ex: OutOfMemoryError
- Classe Error



Exceções

- Pelo ponto de vista de exceções temos condições de erros, que podem ser classificadas em:
 - **Exceções Explícitas**
 - Devem ser tratadas
 - São as exceções que o programador deve obrigatoriamente tratar no programa
 - Sinalizam condições contornáveis
 - Um método deve declarar todas as exceções explícitas que está sujeito
 - Ex.: entrada de dados inválida, fim de arquivo, etc.
 - **Exceções Implícitas**
 - Não precisam ser tratadas diretamente
 - Não há muito o que fazer a não ser terminar o programa
 - Sinalizam condições geralmente incontornáveis
 - São as exceções que estão fora do controle do programador
 - Erros internos no ambiente de *runtime* do Java (a JVM)
 - Ou derivam de condições que deveriam ter sido tratadas pelo programador
 - Ex.: ponteiro nulo, índice fora dos limites, etc.

Exceções e Java

- Em Java, Exceções são representadas por objetos.
 - Alguma surpresa 😊
- Exceções são subclasses (*herdam*)
 - Da classe *Throwable*
- Uma instância da classe Throwable é criada quando uma exceção é lançada
 - “*Uma exceção foi lançada*” é a terminologia Java apropriada para “*aconteceu um erro*”
- As exceções podem ser lançadas:

- Pelo sistema, pelas classes ou intencionalmente nos próprios sistemas que o programador está implementando.

Exceções e Java

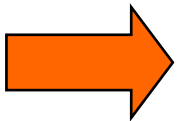
- A classe *Throwable* possui duas subclasses:

- A Classe *Error*

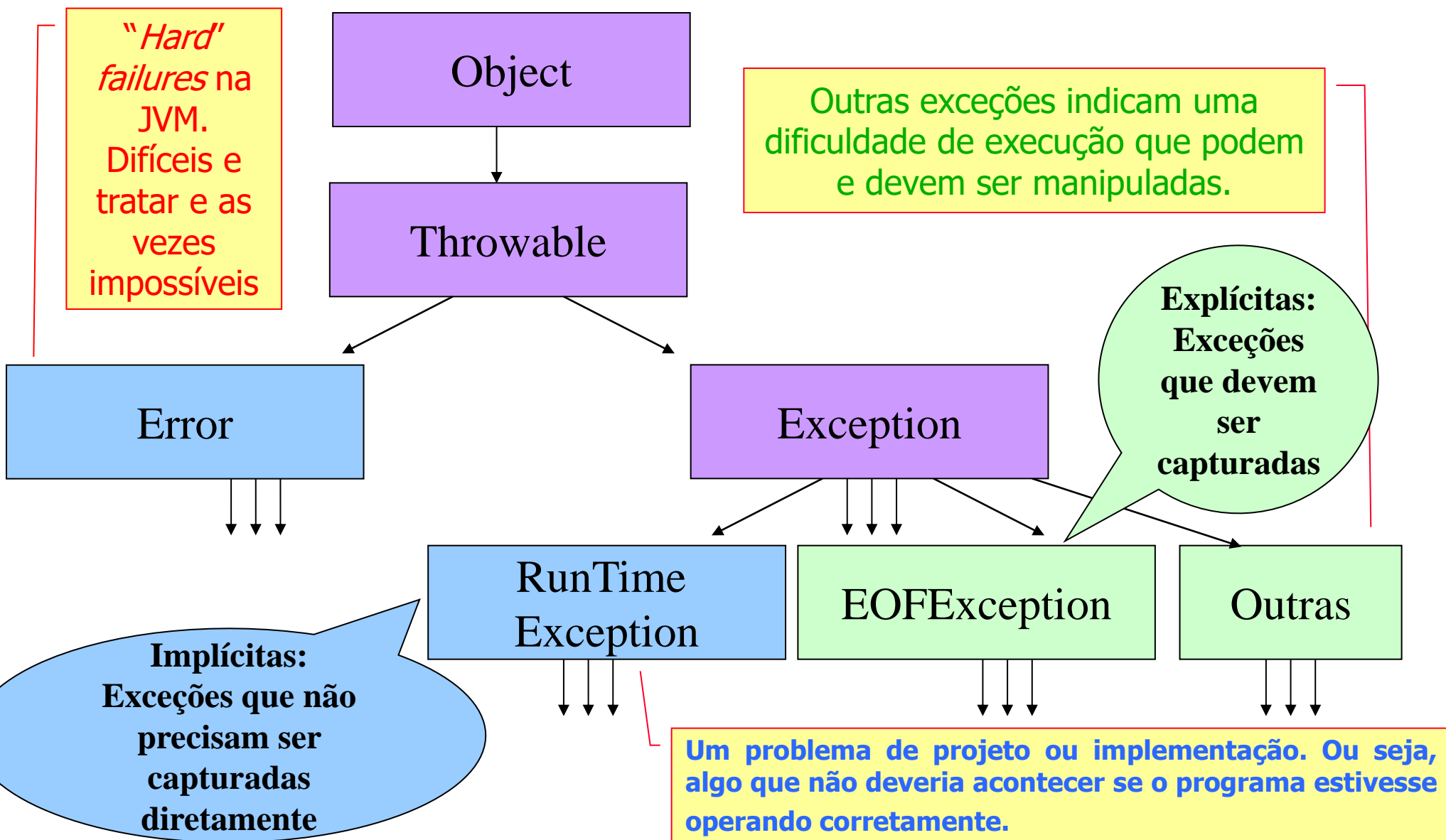
- A Classe *Exception*

- Mas quais são as principais características delas???.....

-Quando usar uma ou outra classe.....????



Hierarquia de Herança de Exceção

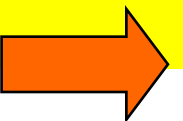


Exceções Comuns

- Exemplos de exceções que já vem na API Java...
 - ArithmeticException
 - O resultado da divisão por zero entre inteiros: `int i = 12 / 0;`
 - NullPointerException
 - Uma tentativa em acessar um atributo de objeto ou método quando o objeto não está instanciado;
 - NegativeArraySizeException
 - Uma tentativa em criar um array com uma dimensão negativa;
 - ArrayIndexOutOfBoundsException
 - Uma tentativa de acessar um elemento de um array além do tamanho normal.
 - IOException
 - AWTException
 - InterruptedException
 - Etc

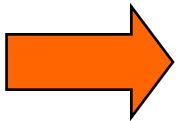
Dúvida: Professor, vou ter que decorar tudo isso....???

A cola vai ficar muito grande??...



Trabalhando com Exceções

- Blz...
 - Entendemos como Java modela e estrutura as exceções...
 - Mas como podemos usar isso no dia a dia....



Trabalhando com Exceções

- Para realizar o tratamento de exceções, o programador deve saber:

- Capturar as exceções lançadas pelos métodos que tentamos executar.

- Comandos: **try**, **catch**, **finally**

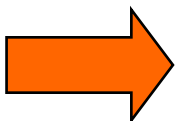
- Anunciar uma exceção que pode ser lançada por um método, ou seja a declaração de métodos que podem lançar exceções;

- Comando: **throws**

- Lançar e repassar exceções dentro de métodos.

- Comando: **throw**

Vamos conversar sobre eles...



Capturando Exceções - Bloco try-catch

- O que o try-catch efetivamente significa é:
 - “Experimente este trecho de código, que poderá causar uma exceção.
 - Se ele for executado corretamente, prossiga com o programa.
 - Se o código lançar uma exceção, apanhe-a e trate dela”
- Um bloco catch pode tratar de qualquer exceção que seja da mesma classe...
 - ou uma subclasse daquela declarada.
- As exceções que não são tratadas em blocos catch correspondentes aonde foram ocasionadas...
 -são passadas para o método anterior da pilha.
- Essa propagação ocorre sucessivamente....
 -até que algum método faça o catch ou até passar do main, chegando a JVM, que pára a aplicação e mostra a stack trace no output padrão.

Capturando Exceções - Bloco finally

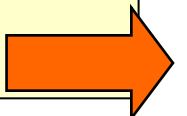
- Nele fica o código que deve sempre ser executado, ocorrendo uma exceção ou não.
 - Um bom uso é para liberar recursos que são utilizados no try
 - Ex.: fechar um arquivo, a conexão com banco de dados, etc.
- O finally é executado sempre, até mesmo se existir um retorno do método (return) dentro do try.
 - Ele só não é executado se a JVM for desligada, através de um `System.exit()` ou um erro irreversível.
- O bloco finally aparece logo após o bloco try-catch.

Capturando Exceções – Bloco try-catch / finally

- **Sintaxe: Capturar exceções lançadas**

- ☆ Tentar executar um bloco de código
- 🕒 No caso de erro, capturar exceções que foram lançadas
- 🕒 Finalmente realizar algum tipo de limpeza

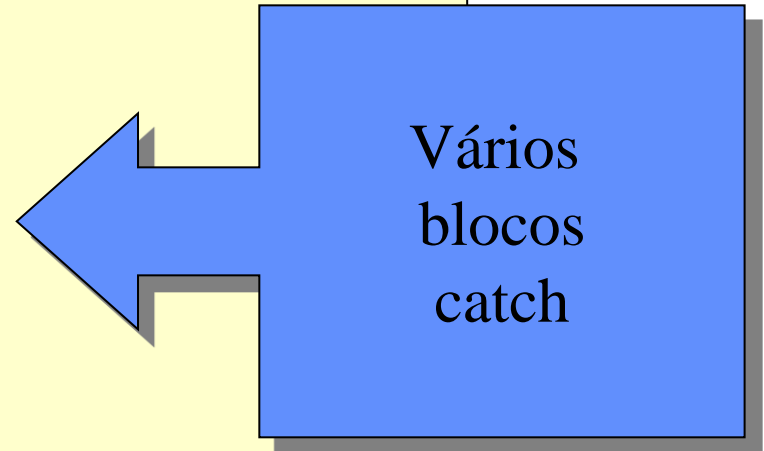
```
try {  
    // ... executar algo que possa causar uma exceção  
}  
catch ( TipoExcecao variavel) {  
    // ... tratar exceção para TipoExcecao  
}  
finally {  
    // ... ao final executar sempre este código  
}
```



Capturando Exceções – Bloco try-catch / finally

- É possível tratar diversas exceções

```
try {  
    // ... executar algo que possa causar uma exceção  
}  
catch ( TipoExcecao1 variavel1) {  
    // ... tratar exceção1  
}  
catch ( TipoExcecao2 variavel2) {  
    // ... tratar exceção2  
}  
finally {  
    // ... ao final executar sempre este código  
}
```



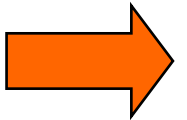
Cuidado com a ordem!

Ex.: se a captura de Exception fosse a primeira, as outras nunca se iam executadas.

Capturando Exceções

- Exemplo de Cenário: Como vocês devem ter percebido...
 - A principal vantagem da manipulação de erros por exceções é a:
 - Separação do código para manipulação de erros do código “normal” do programa.
 - Exemplo: Imagine que temos o seguinte algoritmo....:
 - Como podemos incluir código para tratamento de exceções???

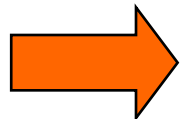
```
lerArquivo()  
{  
    abrir o arquivo;  
    determinar seu tamanho;  
    alocar memória suficiente;  
    ler o arquivo para a memória  
    fechar o arquivo;  
}
```



Capturando Exceções

- Solução 1
 - Tratamento “complicado” de erros
 - O que vocês acham...

```
tipoErro leArquivo() {  
    tipoErro códigoErro = 0;  
    abrir arquivo;  
    se (arquivo abriu) então {  
        determinar tamanho do arquivo;  
        se (conseguiu obter tamanho do arquivo) então {  
            alocar memória suficiente;  
            se (conseguiu memória suficiente) então {  
                ler o arquivo para memória;  
                se (leitura falhou) então  
                    códigoErro = -1;  
            }  
            senão  
                códigoErro = -2  
        }  
        senão  
            códigoErro = -3  
        fechar o arquivo;  
        se (arquivo não fechou)  
            códigoErro = -4  
    }  
    senão  
        códigoErro = -5  
    retorne códigoErro;  
}
```



Capturando Exceções

- Solução 2
 - Tratamento “fácil” de erros
 - E agora... Melhorou..

Principal vantagem é a separação do código para manipulação de erros do código “normal” do programa.

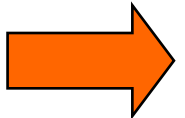
```
lerArquivo() {  
    try {  
        abrir o arquivo;  
        determinar seu tamanho;  
        alocar memória suficiente;  
        ler o arquivo para a memória  
        fechar o arquivo;  
    }  
    catch (Exceção falhouAbrirArquivo) {  
        fazAlgumaCoisa;  
    }  
    catch (Exceção falhouDeterminarTamanho) {  
        fazAlgumaCoisa;  
    }  
    catch (Exceção falhouAlocarMemória) {  
        fazAlgumaCoisa;  
    }  
    catch (Exceção falhouLerArquivo) {  
        fazAlgumaCoisa;  
    }  
    catch (Exceção falhouFecharArquivo) {  
        fazAlgumaCoisa;  
    }  
}
```

Anunciando uma exceção

- Blz
 - Até agora aprendemos a capturar as exceções que foram lançadas:
 - Sabemos que o compilador verifica se o programador lidou com as exceções de um método

 Mas como ele sabe quais exceções deveriam ser informadas???

– Como podemos declarar essa informação???



Anunciando uma exceção - throws

→ A resposta é que o método original...

- ...deve indicar em sua assinatura as exceções que ele possivelmente poderia lançar.

- Para indicar que um método pode lançar uma exceção,
 - Deve-se usar a cláusula *throws* na definição do método
- A cláusula *throws* indica que algum código no corpo do método pode lançar uma exceção.
 - É usado para especificar quais os tipos de exceções que um método pode devolver !

Anunciando uma exceção - throws

Sintaxe: Anunciar exceções para serem lançadas

Método não apenas informa valores a serem retornados, informa também o que pode sair errado

Indica que este método pode lançar uma exceção

```
public static void sleep (long t)
```

```
throws
```

```
InterruptedException {}
```

Um método que executar o método *sleep* deve:

☆ **1. capturar a exceção lançada pelo *sleep* usando *try-catch* / *finally***

☆ DivisaoSimplesMetodoTry.java

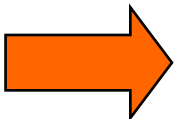
ou

🕒 **2. anunciar a exceção lançada pelo *sleep* e repassá-la usando o comando *throw*.**

🕒 Lancamentos.java

Tipo da exceção a ser lançada

Logo.....



Anunciando uma exceção - throws

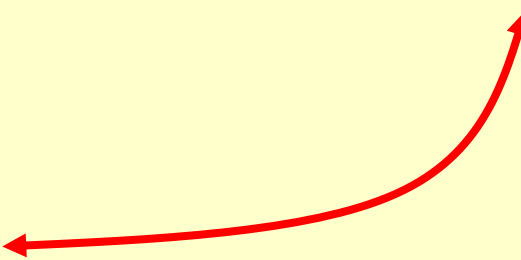
- ... para capturar a exceção lançada pelo *sleep*, definido anteriormente, devemos usar *try-catch*

```
public static void sleep (long t)
```

```
throws
```

```
InterruptedException {
```

```
...  
void meuMetodo ( ) {  
    try {  
        x.sleep(10);  
    }  
    catch (InterruptedException e) {  
        // Tratamento  
    }  
}  
...  
...
```



Anunciando uma exceção - throws

- Se o método precisar lançar vários tipos de exceções:
 - Todos devem ser colocados na cláusula *throws* separados por vírgulas:

```
public static void sleep (long t)
```

```
throws
```

```
InterruptedException, EOFException, NumberFormatException }
```

- Assim como no *catch*,
 - É possível usar uma **superclasse** do grupo de exceções para indicar que o método pode lançar quaisquer **subclasse** dessa exceção

```
public static void sleep (long t)
```

```
throws
```

```
Exception }
```

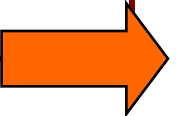
Lançando uma exceção

- Blz...

- Usamos o try/catch para capturar exceções e throws para anunciar que ele poderá acontecer...

- Mas existem dois lados em cada exceção:
 - O lado que lança a exceção e o lado que captura.

→ Quem faz o lançamento real? De onde vêm as exceções?



Lançando uma exceção - throw

- Para tanto é necessário criar uma instância da classe de exceção em questão e utilizar a instrução **throw**, para lançá-la.
 - Importante: Somente podem ser lançados objetos subclasses de *Throwable*
- Depois que uma exceção é lançada, o método termina imediatamente, sem executar qualquer outro código
 - Além do código do *finally*, se existir.

Lançando uma exceção - throw

- Lançar e repassar uma exceção é útil pois permite:
 - Tratar tanto as exceções que chegam no seu método
 - Como também permitir que o método que chamou o seu método as trate.
- Apenas usar o *try* não passa uma exceção adiante
 - Usado apenas para capturar
- A simples inclusão da cláusula *throws* não dá chance de lidar com a exceção
 - Usado para “assinatura” do método e verificações de corretude pelo compilador
- Se quiser gerenciar as exceções e passá-la adiante (lançá-la) para quem chamou é necessário usar:
 - *try-catch*, *throws* e *throw*.

Lançando uma exceção - throw

- Vamos analisar um exemplo...

- Lembram do método sleep...

```
public static void sleep (long t)
```

```
throws
```

```
InterruptedException {
```

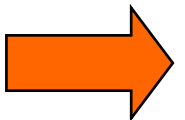
- Ele anuncia que....

- ...poderá lançar uma exceção do tipo InterruptedException

- Eu posso capturar e tratar a exceção dentro do método meuMetodo()....

```
void meuMetodo ( ) throws InterruptedException {  
    try { x.sleep(10);  
    } catch (InterruptedException e) { // Faço alguma coisa ... }  
}
```

- Ou.....



Lançando uma exceção - throw

- ...Eu posso:
 - Anunciar a exceção lançada pelo **sleep** no método meuMetodo()..
 - E repassar a responsabilidade do que fazer nesse caso para quem chamou meuMetodo() usando o comando **throw**.

```
void meuMetodo ( ) throws InterruptedException {  
    try {  
        x.sleep(10);  
    }  
    catch (InterruptedException e) {  
        // Faço alguma coisa ...  
        throw e;  
    }  
}
```

Quem executar o método **meuMetodo()** deverá capturar e tratar a exceção **InterruptedException**

Repassa a exceção capturada para o método que chamou o **meuMetodo()**

Exemplo lançamento - Exceções Explícitas

- Outros exemplos...
 - Compilação OK!! Por quê??
 - Exceções foram declaradas pelos métodos e lançadas

```
class ListaE {  
    private ListaE próximo;  
    public void insere(ListaE e) throws Exception {  
        if (e == null)  
            throw new Exception("Elemento nulo");  
        e.próximo = próximo;  
        próximo = e;  
    }  
}
```


Exemplo lançamento - Exceções Explícitas

- Outros exemplos...
 - Compilação NOT OK!! Por quê??
 - Precisam ser tratadas pelos métodos

```
class ListaE {  
    public void insere(ListaE e) throws Exception {  
        if (e == null)  
            throw new Exception("Elemento nulo");  
        e.próximo = próximo;  
        próximo = e;  
    }  
  
    public void duploInsere(ListaE e1, ListaE e2) {  
        insere(e2); // ERRO!  
        insere(e1); // ERRO!  
    }  
}
```

Exemplo lançamento - Exceções Implícitas

- Outros exemplos...
 - Compilação OK!! Por quê??
 - Não precisam ser declaradas pelos métodos

```
class ListaI {  
    private ListaE próximo;  
    public void insere(ListaE e) {  
        if (e == null)  
            throw new Error("Elemento nulo");  
        e.próximo = próximo;  
        próximo = e;  
    }  
}
```

Exemplo lançamento - Exceções

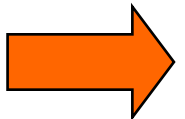
Implícitas

- Outros exemplos...
 - Compilação OK!! Por quê??
 - Não precisam ser tratadas pelos métodos

```
class ListaI {  
    public void insere(ListaE e) {  
        ... throw new Error("Elemento nulo"); ...  
    }  
    public void duploInsere(ListaE e1, ListaE e2) {  
        insere(e2); // OK!  
        insere(e1); // OK!  
    }  
}
```

Criando novas exceções

- Blz
 - Pra fechar... Até agora aprendemos a capturar, anunciar e lançar as exceções definidas na API...
 - Mas é bem comum criar uma própria classe de exceção para controlar melhor o uso de suas exceções...
 - Dessa maneira podemos passar valores específicos para ela carregar, e que sejam úteis de alguma forma.
 -mas como podemos criar nossas próprias classes exceção??....

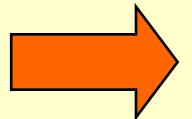


Criando novas exceções

- Para criar novas exceções..
 - Basta criar classes que estendam as classes de exceções ou suas subclasses.
- Normalmente possuem dois construtores,
 - Mais o que for necessário para a modelagem da classe
- Dependendo da superclasse
 - Você poderá ter uma exceção implícita ou explícita.

Exceção explícita
(*Exception*)

```
class MinhaExcecao extends Exception {  
    public MinhaExcecao () { }  
    public MinhaExcecao (String msg) {  
        super(msg);  
    }  
}
```



Usando as novas exceções

- **Importante:** Eu tenho que capturar exceção com o mesmo nome que foi criado (mesmo tipo)...

```
class testeExcecao {  
    void meuMetodo ( ) throws MinhaExcecao {  
    }  
  
    void outroMetodo ( ) {  
        try {  
            meuMetodo();  
        } catch (MinhaExcecao e) {  
            // tratamento  
        }  
    }  
}
```

```
class MinhaExcecao  
    extends Exception {  
        .....  
    }
```

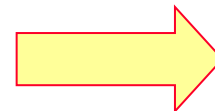


Exercício Junto....

- Blz, entendemos muita coisa hoje.....

⇒ Vamos agora analisar juntos exemplos que tenha:

- try/catch
- throws
- throw



Exercício Junto....

- 1. O que o programa está fazendo?
- 2. E se em vez de fazer o try em torno do for inteiro eu colocar o try dentro do for
 - Comentar as linhas 22 e 25 e descomentar as linhas 20 e 30.
- 3. Agora tome como base o código inicial ao lado e retire o try/catch de dentro do método2 e o coloque em volta da chamada do método2, na linha 12 .
- 4. Faça a mesma coisa, retirando o try/catch envolto na chamada do método2 e colocando em volta da chamada do método1, na linha 6

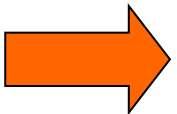
```

1 package excecoes;
2
3 class Teste {
4     public static void main(String [] args) {
5         System.out.println("inicio do main");
6         metodo1();
7         System.out.println("fim do main");
8     }
9
10    public static void metodo1() {
11        System.out.println("inicio do metodo1");
12        metodo2();
13        System.out.println("fim do metodo1");
14    }
15
16    public static void metodo2() {
17        System.out.println("inicio do metodo2");
18        int[] array = new int[10];
19
20        //for(int i = 0; i < 15; i++) {
21        try {
22            for(int i = 0; i <= 15; i++) {
23                array[i] = i;
24                System.out.println(i);
25            }
26
27        } catch (ArrayIndexOutOfBoundsException e) {
28            System.out.println("erro: " + e);
29        }
30        //}
31        System.out.println("fim do metodo2");
32    }
33 }

```


Exercício Junto....

- Responda as perguntas abaixo, de acordo com o código a seguir:
 - 1. Qual o resultado da execução do código do próximo slide.
 - 2. Qual o resultado se comentarmos a linha 18 e descomentar a linha 22?
 - 3. Qual o resultado se comentarmos as linhas 11 e 22 e descomentar a linha 18?
 - 4. O que acontece se retirarmos a cláusula **throws `ArrayIndexOutOfBoundsException`** do método **proced**?
 - Qual o resultado do código, de acordo com o especificado em 1?



Exercício Junto....

2/2

```
1 package excecoes;
2 class Lancamentos {
3     // public static void proced( ) {
4     public static void proced( ) throws ArrayIndexOutOfBoundsException {
5         try {
6             int c[ ] = { 1 };
7             c[42] = 99;
8         }
9         catch(ArrayIndexOutOfBoundsException e) {
10             System.out.println("Estouro indice array metodo: " + e);
11             throw(e);
12         }
13         System.out.println("Metodo apos o throw - Nao sou executado com throw");
14     }
15
16     public static void main(String args[ ]) {
17         try {
18             proced( );
19             int a = args.length;
20             System.out.println("a = " + a);
21             int b = 42 / a;
22             // proced( );
23         }
24         catch(ArithmeticException e) {
25             System.out.println("div por 0: " + e);
26         }
27         catch(ArrayIndexOutOfBoundsException e) {
28             System.out.println("Estouro indice array main: "+e);
29         }
30     }
31 }
```

Exercício

- Vamos implementar uma classe que tem um método que verifica e imprime se um número é Par

```
public void imprimePar(int num)
```

- Caso o número seja Impar, esse método lança uma exceção (*MinhaExcecaoImpar - que herda de Exception*) contendo o número analisado.
- Essa classe de exceção também tem o método toString() implementando, contendo informações sumarizadas sobre a exceção
- Independente se é par ou impar, imprima uma mensagem de “Fim de codificação realizado....”

Exercício

```
1 class Lancamentos {
2     // public static void proced( ) {
3     public static void proced( ) throws ArrayIndexOutOfBoundsException {
4         try {
5             int c[ ] = { 1 };
6             c[42] = 99;
7         }
8         catch(ArrayIndexOutOfBoundsException e) {
9             System.out.println("Estouro indicearray metodo: " + e);
10            throw(e);
11        }
12        System.out.println("Metodo apos o throw - Nao sou executado com throw");
13    }
14
15    public static void main(String args[ ]) {
16        try {
17            proced( );
18            int a = args.length;
19            System.out.println("a = " + a);
20            int b = 42 / a;
21            // proced( );
22        }
23        catch(ArithmeticException e) {
24            System.out.println("div por 0: " + e);
25        }
26        catch(ArrayIndexOutOfBoundsException e) {
27            System.out.println("Estouro indice array main: "+e);
28        }
29    }
30 }
```

Implemente esse
código e analise o
comportamento
conforme discutido
em sala de aula