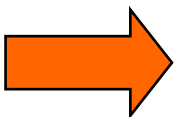


Tratamento de Exceção em Java

Parte I

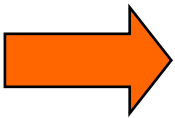
Introdução

- 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.

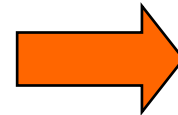


Tratamento de Exceções

- 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.

Tratamento de Exceções

- Blz,
 - O tratamento das exceções é a nossa salvação....
 - Mas o que é uma exceção mesmo.... ????
 - É a mesma coisa que um erro....???

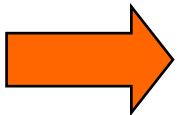


Erros e Exceções

- Uma condição de Erro qualquer no processamento de um código é chamado de Exceção.
 - Ou seja, qualquer condição anormal (erro) que perturbe o fluxo normal do programa, enquanto o programa está em execução, é uma exceção.
- Por exemplo, exceções podem ocorrer quando:
 - O arquivo que você tentou abrir não existe;
 - A conexão da rede está rompida;
 - Os operandos sendo manipulados estão fora da faixa prescritas;
 - O arquivo .class que você está interessado em carregar está perdido.
 - Erros de digitação dos dados de entrada pelo usuário
 - Erros de dispositivos, como impressora sem papel
 - Limitações físicas como disco cheio
 - Erros de código, como acesso a pilha vazia
 - tentar abrir uma URL inexistente,
 - etc

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

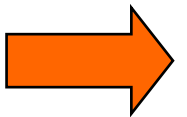
- Erros

- 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

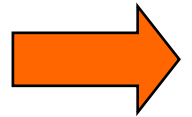


Exceções

- 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.

Tratamento de Exceções

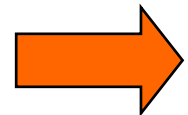
- Blz,
 - Mas como é feito o tratamento dessas exceções no código????....
 - “Na mão” ou tenho alguma ajuda da linguagem....



Tratamento de Exceções

- Em linguagens sem tratadores de exceção, precisamos implementar dois mecanismos deselegantes:
 - 1) Funções/procedimentos que devem sinalizar exceções, retornando um valor adicional indicando o erro
 - 2) No código principal, precisamos acrescentar vários If's/Case's, que indicam o que deve ocorrer nas situações de erro
 - Exemplo:
 - Em uma função abre_arquivo(nome_o_arq), poderíamos pensar em uma outra função que poderá retornar os seguintes valores:..
 - 0:arquivo aberto sem problemas
 - 1:arquivo não encontra o
 - 2:chama a a função erra a
 - 3:arquivo já aberto
 - 4:arquivo não pode ser escrito
 - .Etc....

Isso acaba gerando código com essa cara...



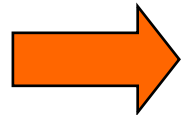
Tratamento de Exceções

```
int cod_erro = abre_arquivo(nome_arq);  
switch(cod_erro) {  
    case 0:  
        /* executar algoritmo normal, onde outras  
        situações  
        de erro podem ser geradas */  
    case 1:  
        /* voltar no loop e solicitar entrada  
        do arquivo novamente */  
    case 2:  
        /* erro no código */  
    case 3:  
        /* erro no código */  
    case 4:  
        /* informar que usuário não possui  
        direitos */
```

Ai meu Deus... ☹

Tratamento de Exceções

- Pô, ninguém merece.... ☹
 - Mas Java é uma linguagem gente boa... 😊
 - O que ela pode fazer por mim nesse caso...



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

- Os principais métodos de **java.lang.Throwable** são:
 - void **printStackTrace()**:
 - lista a sequência de métodos chamados até o ponto onde a exceção foi lançada;
 - String **getMessage()**:
 - retorna o conteúdo de um atributo que contém uma mensagem indicadora da exceção;
 - **toString()**
 - também é implementado e retorna uma descrição breve da exceção.

Exceções e Java

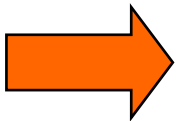
- A classe *Throwable* possui duas subclasses:

- A Classe *Error*

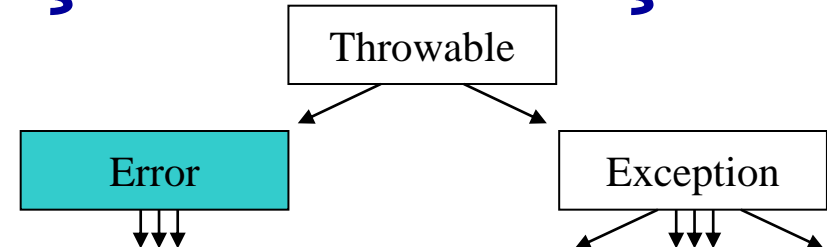
- A Classe *Exception*

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

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



Hierarquia de Herança de Exceção



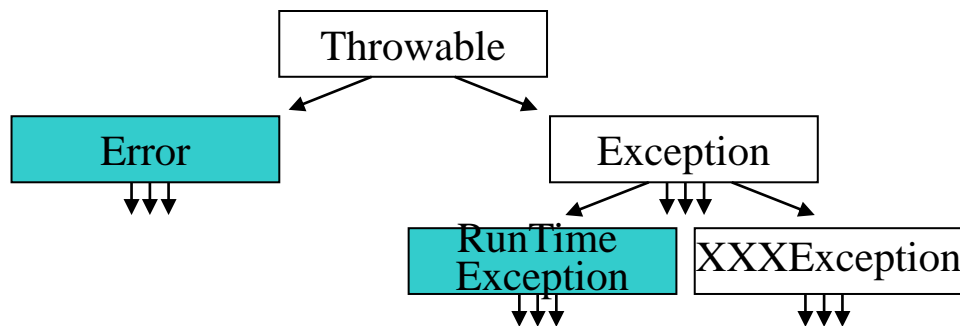
- Descrição das subclasses de *Throwable*

- *Error*

- Descreve erros internos (JVM) e falta de recursos do sistema em tempo de execução (raro)
 - Programador praticamente não trata nem passa objetos deste tipo

Hierarquia de Herança de Exceção

- Descrição das subclasses de *Throwable*
 - *Exception* (dividida em 2)
 - *RuntimeException*:
 - As exceções de *runtime* normalmente acontecem por causa de código que não é bem feito, por exemplo:
 - Conversão errada,
 - Acesso a *array* fora dos limites,
 - Programador tenta usar uma variável antes que ela tenha sido configurada para conter um objeto.
 - Regra: A CULPA é NORMALMENTE do programador !
 - Não tratá-las NÃO implica em erro de compilação
 - Não é necessário tentar capturar estas exceções

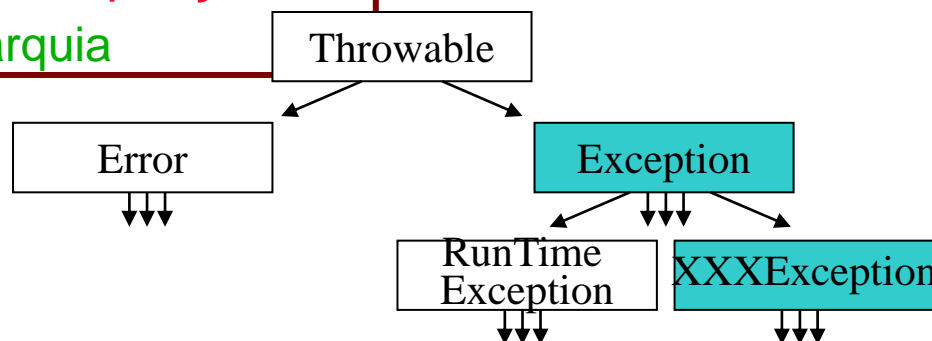


Hierarquia de Herança de Exceção

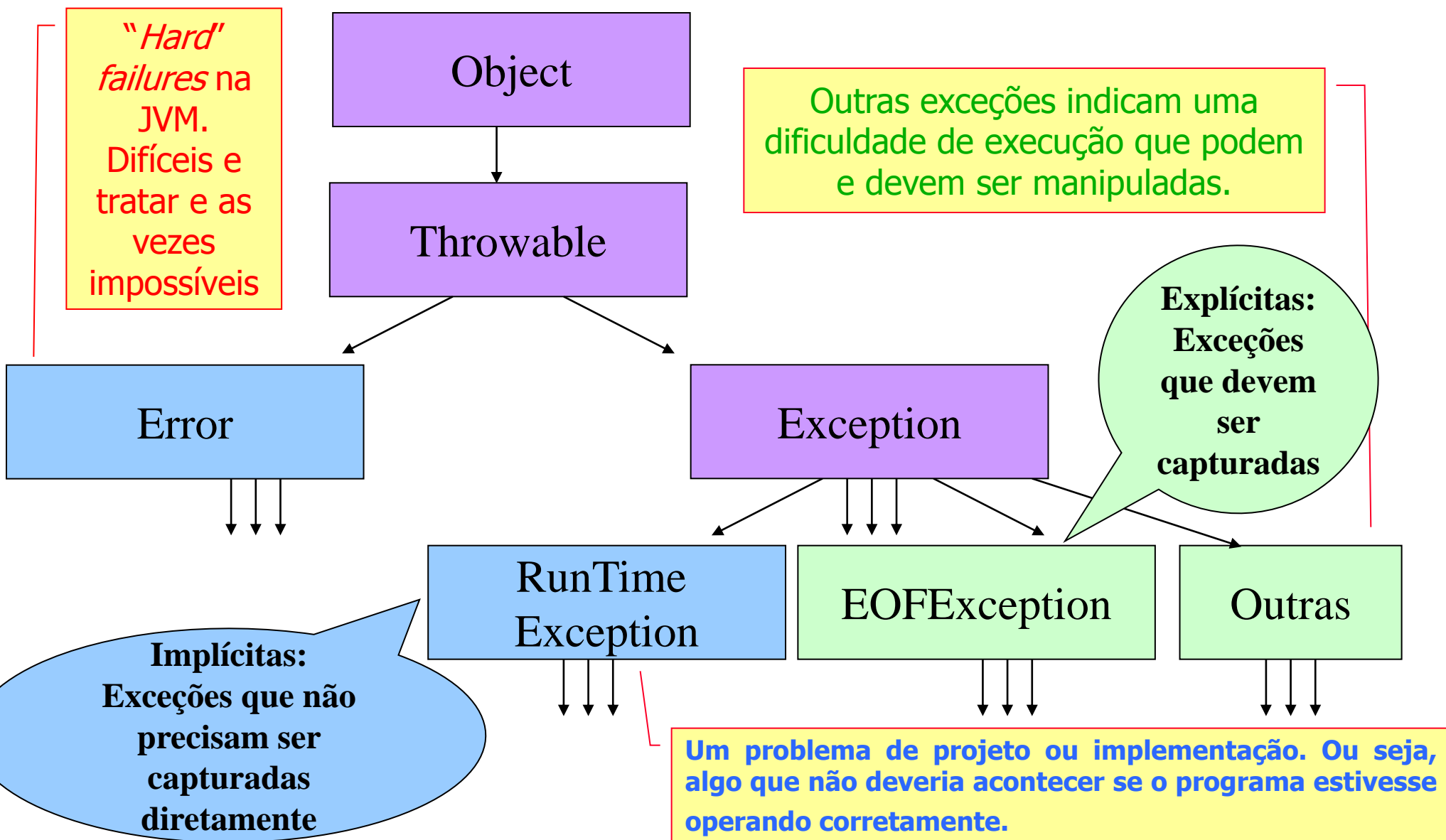
- Descrição das subclasses de *Throwable*
 - *Exception* (dividida em 2)
 - *Outras subclasses de Exception*
 - As outras exceções são interessantes pois indicam que algo estranho está acontecendo,
 - Acontecem devido a um problema que precisa ser tratado
 - Por exemplo
 - Ler um arquivo e o arquivo termina antes do fim do que o programador esperava
 - Quando um URL não está no formato correto (talvez digitado de forma errada).

– Não tratá-las IMPLICA em erro de compilação

- Programador deve tratar esta hierarquia



Hierarquia de Herança de Exceção

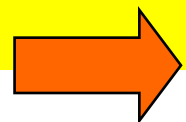


Exceções Comuns

- 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

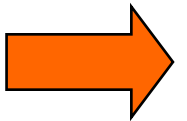
Dúvida: Professor, vou ter que decorar tudo isso....???

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**

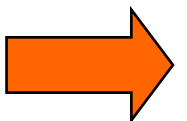
- Anunciar uma exceção que pode ser lançada por um método, ou seja a declaração de métodos que podem lançar exceções;

- 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

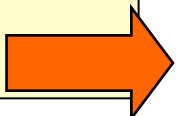
- Nele fica o código que deve sempre ser executado, ocorrendo uma exceção ou não.
 - Um bom uso é para liberar recursos que são utilizados no try
 - Ex.: fechar um arquivo, a conexão com banco de dados, etc.
- O finally é executado sempre, até mesmo se existir um retorno do método (return) dentro do 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

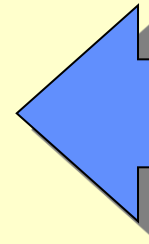
```
try {  
    // ... executar algo que possa causar uma exceção  
}  
catch ( TipoExcecao variavel) {  
    // ... tratar exceção para TipoExcecao  
}  
finally {  
    // ... ao final executar sempre este código  
}
```



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  
}
```



Vários
blocos
catch

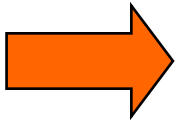
Capturando Exceções – Bloco try-catch / finally

- **Atenção!!!**
 - Quando uma exceção ocorre, procura-se um tratador adequado;
 - As cláusulas catch são checadas em sequência;
 - Em um bloco *catch* múltiplo, o primeiro bloco *catch* que “combina” será executado, e o restante será ignorado.
 - **Portanto, 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???

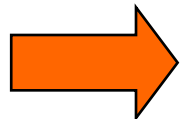
```
lerArquivo()  
{  
    abrir o arquivo;  
    determinar seu tamanho;  
    alocar memória suficiente;  
    ler o arquivo para a memória  
    fechar o arquivo;  
}
```



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

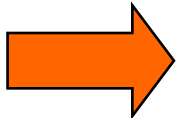
- Solução 2
 - Tratamento “fácil” de erros
 - E agora... Melhorou..

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

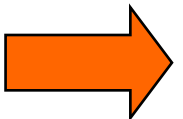
ou

🕒 **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*

```
...  
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 }
```

Anunciando uma exceção - throws

- Importante

- Não se esqueça que a inclusão de uma cláusula *throws* à definição do método significa...

- .. Que o método **poderá lançar** uma exceção se algo sair errado.

- A cláusula *throws* simplesmente **oferece informações extras** à definição do seu método sobre exceções em potencial

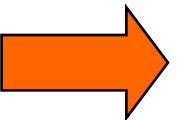
- E permite que Java se certifique que os usuários do método estejam usando o método corretamente.

➡ Pense na descrição de exceções do **método como um contrato** entre o projetista desse método e quem chama o método.

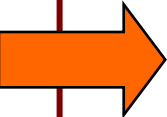
- Além dos parâmetros comuns de um métodos,
- Temos agora informações explícitas sobre as ações anormais que o método pode fazer e a classe que pode tratar.

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

- Muitas exceções são lançadas pelo *runtime* Java:
 - Por exemplo,
 - Se só tiver o bloco try-catch-finally, caso ocorra alguma coisa, a JVM lança a exceção que é capturada por estes blocos ou por métodos dentro das próprias classes Java.
- Também é possível....
 - Lançar qualquer uma das exceções padrão que as bibliotecas de classes Java definem:
 - Ex: RuntimeException
 - Ou então criar e lançar as suas próprias exceções.
- OK, mas como fazer esse lançamento de exceção?? 

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 {
```

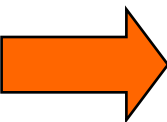
- Ele anuncia que....

- ...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

- ...Além disso 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()**

Lançando uma exceção - throw

- Importante:
 - Se não lançar/repassar novamente a exceção usando a throw,
 - A execução do código de tratamento da exceção termina no catch do meuMetodo() (supondo que não tenha o finally)
 - Com o throw, a execução volta para o método que chamou o meuMetodo()
 - E a execução do código de tratamento da exceção termina no catch do método que chamou o meuMetodo() (supondo que não tenha o finally)

Exceções que um método pode lançar

- **Importante:**
 - Um método só pode lançar as exceções explícitas listadas na cláusula **throws** ou subclasses dessas exceções
 - Um método pode lançar qualquer exceção implícita, mesmo que ela não tenha sido declarada na cláusula **throws**.

Exceção explícita
(**Exception**) definida
pelo **throws**

Exceção implícita
(herda de
RuntimeException), não
definida pelo **throws**

```
...  
void meuMetodo ( ) throws IOException {  
    if ( i == 0 )  
        throw new IOException();  
    if (i == -1)  
        throw new ArithmeticException ();  
}  
...
```

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

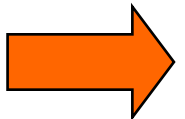
```
class ListaI {  
    public void insere(ListaE e) {  
        ... throw new Error("Elemento nulo"); ...  
    }  
    public void duploInsere(ListaE e1, ListaE e2) {  
        insere(e2); // OK!  
        insere(e1); // OK!  
    }  
}
```

Dicas

- Importante:
 - Existe uma péssima prática de programação em Java
 - Que é a de escrever o `catch` e o `throws` com `Exception`
 - Existem códigos que sempre usam `Exception`
 - Pois isso cuida de todos os possíveis erros.
 - O maior problema disso é generalizar o erro.
 - Se alguém joga algo do tipo `Exception` para quem o chamou,
 - Quem recebe não sabe qual o tipo específico de erro ocorreu
 - E não vai saber como tratar o mesmo.

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??....

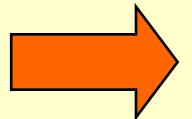


Criando novas exceções

- 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*)

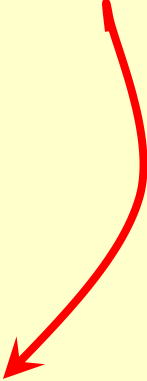
```
class MinhaExcecao extends Exception {  
    public MinhaExcecao () { }  
    public MinhaExcecao (String msg) {  
        super(msg);  
    }  
}
```



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  
        }  
    }  
}
```



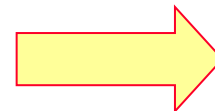
```
class MinhaExcecao  
    extends Exception {  
        .....  
    }
```

Exercício Junto....

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

⇒ Vamos agora analisar juntos um exemplo que tenha:

- try/catch
- throws
- throw



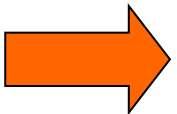
Exercício Junto....

- 1. O que o programa está fazendo?
- 2. E se em vez de fazer o try em torno do for inteiro, colocá-lo apenas encapsulando o bloco de dentro do for (linhas 22 e 23).
- 3. Agora retire o try/catch e coloque ele em volta da chamada do metodo2 .
- 4. Faça a mesma coisa, retirando o try/catch novamente e colocando em volta da chamada do metodo1. Rode os códigos, o que acontece?

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


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 17 e descomentar a linha 21?
 - 3. Qual o resultado se comentarmos as linhas 10 e 21 e descomentar a linha 17?
 - 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?

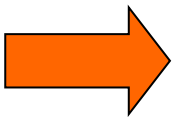


Exercício Junto....

```
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 }
```

Exercícios

- Blz... Agora é hora de exercitar.....
- Tente resolver ou analisar os seguintes problemas...
 - Em dupla
 - Anotar em um .txt a resposta das perguntas e apresentar ao professor no final da aula



Exercício

- O que o programa está fazendo?

```
1 public class CarroBomba {
2
3     public void ligar() {System.out.println("Ligado!");}
4     public void desligar() {System.out.println("Desligado!");}
5
6     public void mover() throws SuperAquecimentoException {
7         System.out.print("Qual o estado: ");
8         String temperatura = Console.readString();
9         if (temperatura.equals("anormal")) {
10             throw new SuperAquecimentoException("Vai explodir tudo!!!");
11         }
12     }
13
14     public static void main(String[] args) {
15         CarroBomba c = new CarroBomba();
16         try {
17             c.ligar();
18             c.mover();
19         }
20         catch(SuperAquecimentoException e) {
21             System.out.println("Agora sujou....");
22             System.out.println(e);
23         }
24         finally {
25             c.desligar();
26         }
27     }
28 }
29
30 class SuperAquecimentoException extends Exception {
31     public SuperAquecimentoException () { }
32     public SuperAquecimentoException (String msg) {
33         super(msg);
34     }
35 }
36 }
```

Exercício

- Qual o resultado do código ao lado?
- E se “comentar” o código da linhas 09 até 19?

```
1  import java.util.ArrayList;
2
3  public class Estudos{
4
5      public static void main(String[] args){
6
7          try {
8              // cria uma ArrayList que conterá strings
9              ArrayList<String> nomes = new ArrayList<String>();
10
11              // adiciona itens na lista
12              nomes.add("Carlos");
13              nomes.add("Maria");
14              nomes.add("Fernanda");
15              nomes.add("Osmar");
16
17              // fornecemos um índice inválido
18              String nome1 = nomes.get(4);
19              System.out.println("O valor obtido foi: " + nome1);
20
21
22              String nome2 = "Java";
23              // vamos fornecer um índice inválido
24              System.out.println(nome2.charAt(4));
25
26          } catch (StringIndexOutOfBoundsException e) {
27              System.out.println(e);
28
29          } catch (IndexOutOfBoundsException e) {
30              System.out.println(e);
31
32          } finally {
33              System.exit(0);
34          }
35      }
36  }
```

Exercício

- Qual o resultado do código ao lado?
- O que deve ser feito para corrigi-lo??

```
1 public class Estudos1{  
2  
3     public static void main(String args[]){  
4  
5         try{  
6  
7             Object x = new Integer(0);  
8             System.out.println((String)x);  
9  
10        } catch (ClassCastException e) {  
11            System.out.println(e);  
12        }  
13    }  
14 }  
15
```

Exercício

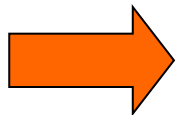
- O que o programa está fazendo?

```
1 public class ParImpar {
2
3     public static void imprimePar(int num) throws ExcecaoImpar{
4         if ((num %2) == 0)
5             System.out.println(num);
6
7         else throw new ExcecaoImpar(num);
8     }
9
10    public static void main(String [] args){
11        try {
12            imprimePar(2);
13            imprimePar(3);
14        }
15        catch (ExcecaoImpar e){
16            System.out.println(e);
17        }
18    }
19 }
20
21 class ExcecaoImpar extends Exception {
22     private int x;
23     public ExcecaoImpar(){}
24     public ExcecaoImpar(String msg){
25         super(msg);
26     }
27
28     public ExcecaoImpar(int x){
29         this.x = x;
30     }
31
32     public String toString(){
33         return "O numero "+x +" eh impar!";
34     }
35 }
```

Exercício

- O que o programa está fazendo?
 - E se passar um valor inteiro (como String) na linha 14?

```
1 public class MainTratamentoExcecoes {  
2  
3     public static void verificaZero(int numero) throws Exception {  
4         if(numero == 0) throw new Exception("Numero nao pode ser zero");  
5     }  
6  
7     public static void processaString(String arg) throws ErroStringTemA {  
8         if(arg.indexOf("A") != -1) throw new ErroStringTemA("String " + arg + " possui A");  
9     }  
10  
11     public static void main(String[] args) {  
12         int valor = 0;  
13         try {  
14             valor = Integer.parseInt(args[0]);  
15             verificaZero(valor);  
16         }  
17         catch(NumberFormatException nfe) {  
18             System.out.println("Erro ao converter valor. Valor recebido: " + args[0]);  
19         }  
20         catch(Exception e) {  
21             System.out.println("Erro desconhecido. Veja a mensagem: " + e.getMessage());  
22         }  
23  
24         try {  
25             processaString("testeA");  
26         } catch (ErroStringTemA e) {  
27             e.printStackTrace();  
28         }  
29     }  
30 }
```



Exercício

```
1 public class ErroStringTemA extends Exception {  
2  
3     public ErroStringTemA() {  
4         super();  
5     }  
6     /**  
7      * @param message  
8      */  
9     public ErroStringTemA(String message) {  
10         super(message);  
11     }  
12     /**  
13      * @param message  
14      * @param cause  
15      */  
16     public ErroStringTemA(String message, Throwable cause) {  
17         super(message, cause);  
18     }  
19     /**  
20      * @param cause  
21      */  
22     public ErroStringTemA(Throwable cause) {  
23         super(cause);  
24     }  
25 }  
26  
27
```

Exercício

- Retomada
 - O que o programa está fazendo?

```
1 public class Retomada {  
2  
3     public static void main(String[] args) {  
4         boolean continua = true;  
5         while (continua) {  
6             continua = false;  
7             try {  
8                 System.out.print("Entre um inteiro positivo: ");  
9                 int i = Console.readInt();  
10                if (i <= 0) throw new NaoPositivoException();  
11            }  
12            catch(NaoPositivoException e) {  
13                System.out.println("Tente novamente!!!");  
14                continua = true;  
15            }  
16        }  
17    }  
18  
19 }  
20  
21 class NaoPositivoException extends Exception {  
22     public NaoPositivoException () { }  
23     public NaoPositivoException (String msg) {  
24         super(msg);  
25     }  
26 }
```

Exercício

- Perdendo uma Exceção
 - O que o programa está fazendo?

```
1 public class Perda {  
2  
3     void infarto() throws InfartoException {  
4         throw new InfartoException();  
5     }  
6  
7     void resfriado() throws ResfriadoException {  
8         throw new ResfriadoException();  
9     }  
10  
11     public static void main(String[] args) throws Exception {  
12         Perda p = new Perda();  
13         try {  
14             p.infarto();  
15         }  
16         finally {  
17             p.resfriado();  
18         }  
19     }  
20 }  
21  
22 class InfartoException extends Exception {  
23     public String toString() {  
24         return "Urgente!";  
25     }  
26 }  
27  
28 class ResfriadoException extends Exception {  
29     public String toString() {  
30         return "Descanse!";  
31     }  
32 }
```

Exercício

- Qual o resultado do código ao lado se digitar 3?
- Qual o resultado do código ao lado se digitar 33?
- Qual o resultado do código ao lado se digitar 01?
 - O que pode ser feito para capturar o que aconteceu...

```

1  import java.util.*;
2
3  public class Estudos2{
4      public static void main(String[] args){
5          String palavra = "Java";
6
7          Scanner in = new Scanner(System.in);
8
9          System.out.print("Informe um inteiro: ");
10         int indice = in.nextInt();
11
12         try{
13             System.out.println("O caractere no índice " +
14                 "informado é " + palavra.charAt(indice));
15         }
16         catch(StringIndexOutOfBoundsException e){
17             System.out.println("Erro: " + e.getMessage());
18         }
19     }
20 }

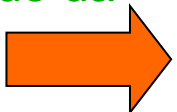
```

Exercício - Throws e Herança

- Se a definição do método redefine um método da superclasse que inclui a cláusula *throws*,
 - Existem regras especiais sobre como o seu método redefinido trata de *throws*.
- O novo método não exige o mesmo conjunto de exceções listadas na cláusula *throws* do pai.
 - Classes mais especializadas podem tratar um conjunto reduzido (mais específico) de exceções, inclusive nenhuma.

```
public class RadioPlay {  
    public void startPlaying () throws SoundException {  
        // ...  
    }  
}  
  
public class StereoPlay extends RadioPlay {  
    public void startPlaying () {  
        // ...  
    }  
}
```

- A recíproca não é verdadeira:
 - Um método de subclasse não pode lançar mais exceções do que o seu método da superclasse
 - Sejam exceções de tipos diferentes ou classes de exceções mais genéricas.



Exercício

- Herança: Aprofundando
 - Analise a estruturação das classes. “Brinque” com o código verificando o que acontece quando algumas linhas são descomentadas..

`DirecaoPerigosa.java`

Parte II

Exercício

- Atividade valendo ponto em sala de aula....
 - Em dupla.. Entregar até o final da aula!!!!
 - 1) Explique o que está acontecendo e a saída apresentada nos códigos que estão em Atividades Extra - Exceção.pdf
 - 2) Dado o pacote SistemaProfessor-Aluno.zip
 - Termine a implementação das classes Cadastro e Simulacao de acordo com as instruções que estão no código fonte.
 - Consulte a API sempre que necessário.