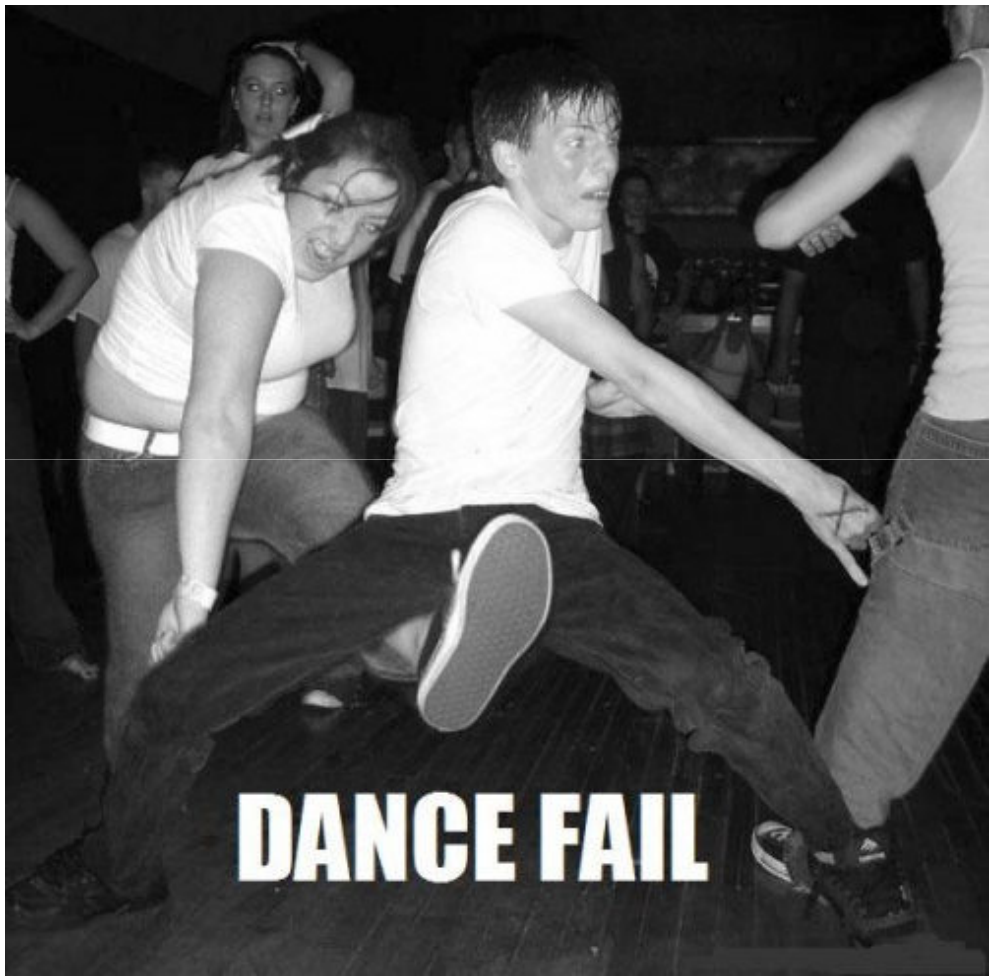
Programação Orientada a Objetos e a Linguagem Java

Rodrigo da Cruz Fujioka
rodrigofujioka@gmail.com

Controle de Exceções em Java.

- Este módulo explora detalhes sobre o tratamento de erros em Java.
  - Exception definição
  - Exception Ocorrência
  - Exception Tratamento
  - Exception Propagação

Erro.



Erro.

Erro.

- O termo “exceção” (do inglês “**exception**”) é um atalho para a frase “evento excepcional”.

Definição: Uma exceção é um evento, que ocorre durante a execução de um programa, que interrompe o fluxo normal das instruções do programa.

(http://download.oracle.com/javase/tutorial/essential/exceptions/definition.html)

Erro em Java.

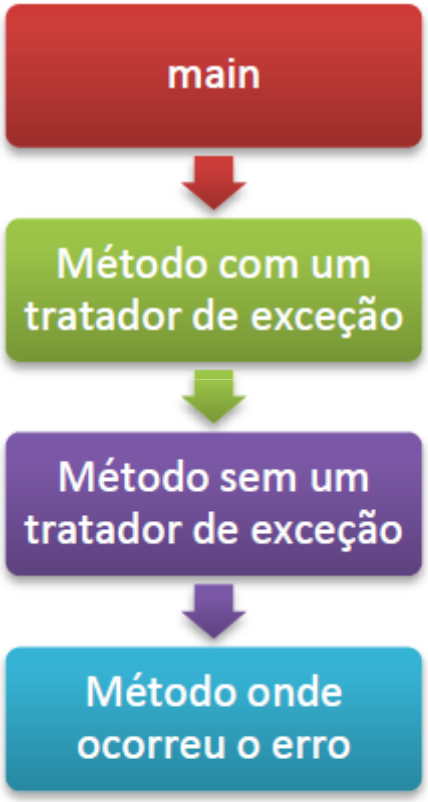


Figura 1: Pilha de chamadas

Erro em Java.

Figura 2: Busca na pilha de chamadas por um tratador de exceção

Erro em Java.

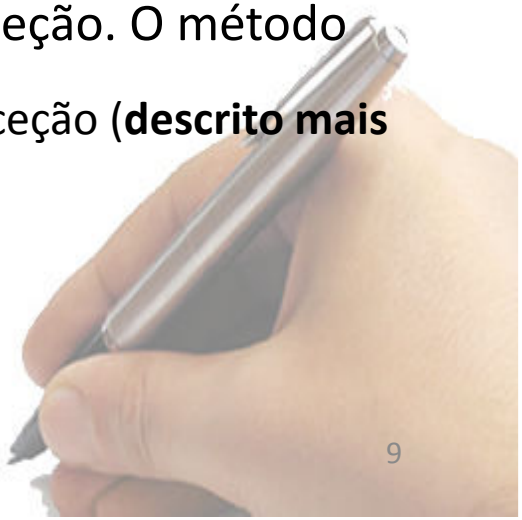
Código válido na linguagem de programação Java precisa honrar o Requisito "**Capturar** ou **Relançar**"

Try – Throws - Finally em Java.

- Isso significa que código que dispara **exceções** precisa estar dentro de uma das seguintes instruções:

- Uma instrução ***try*** que captura a exceção.
  - **O bloco try precisa prover um tratador para a exceção (descrito mais adiante).**

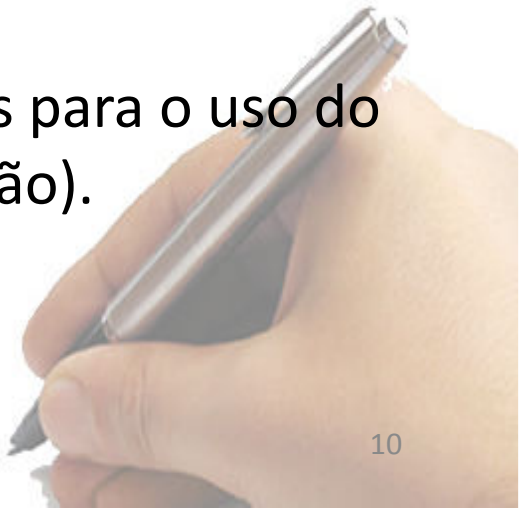
- Um método que especifica que ele pode disparar a exceção. O método precisa fornecer uma cláusula ***throws*** que lista a exceção (**descrito mais adiante**).



Try – Throws - Finally em Java.

- **try -** é utilizada para indicar um bloco de código onde é possível que ocorra uma exceção.
- **catch –** serve para manipular as exceções, ou seja, tratar o erro
- **finally –** sempre será executado depois do bloco try/catch. O importante é saber que esse bloco sempre será executado
- (exceto nos casos de encerramento da jvm System.exit()).

- Veja abaixo as combinações válidas e inválidas para o uso do **try{}, cacth{} e finally{}** (questão de certificação).



Try – Throws - Finally em Java.

Combinações válidas:

try{}	catch{}	
try{}	finally{}	
try{}	catch{}	finally{}

Inválidas não Compila:

try{}	–	–
catch{}	finally{}	–
try{}	finally{}	catch{}

Opa!.

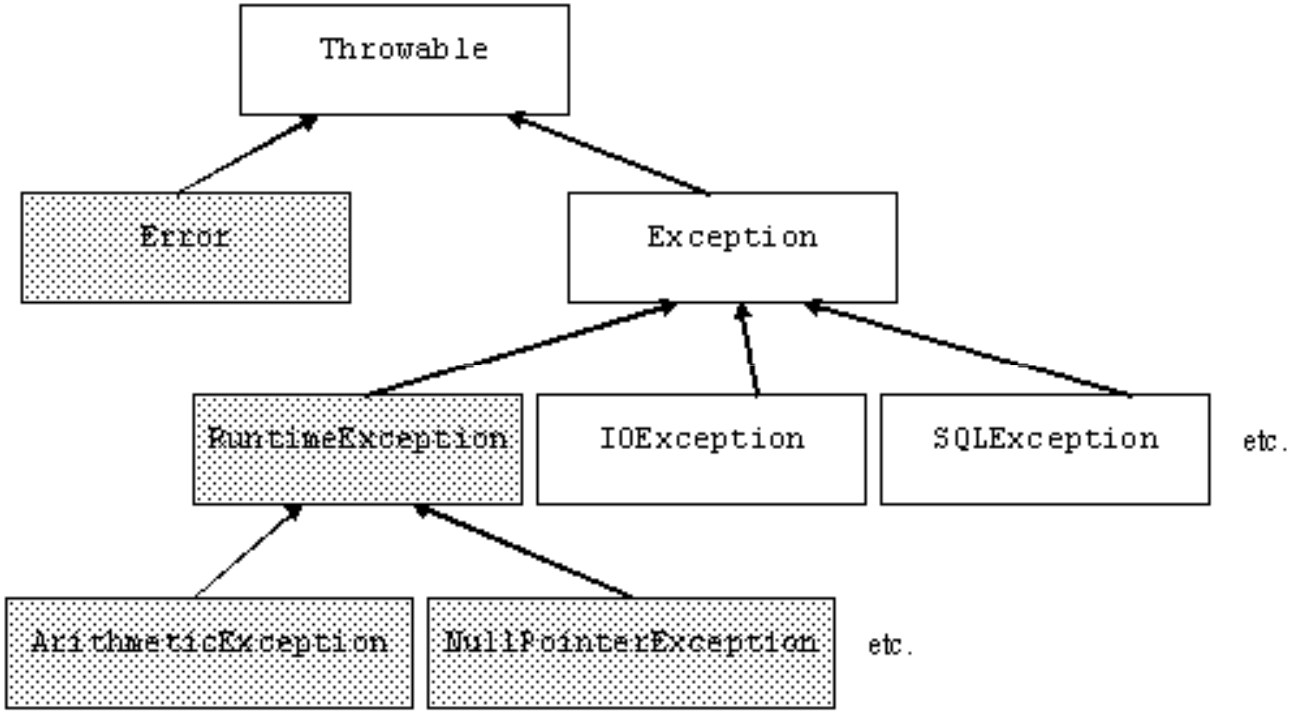
Responda!

todas as exceções estão sujeitas ao

Requisito Capturar ou relançar?

Não

Exceções



http://www.learn-java-tutorial.com/Java-Exceptions.cfm

Tipos de Exceção

- Exceções checadas (Checked Exceptions) .
- Exceção não checadas(Unchecked Exceptions).
- ERRO

Exceções Checadas

- São condições excepcionais que um aplicativo bem-escrito deveria antecipar .

  – Exceções checadas são sujeitas ao **Requisito Capturar** ou **Especificar.** Todas as exceções são exceções checadas, exceto aquelas indicadas por Error**, RuntimeException e suas subclasses.**

ERRO

- Estas são condições excepcionais que são externas ao aplicativo, e que o aplicativo não consegue antecipar.

- Erros não estão sujeitos ao Requisito Capturar ou Especificar. Erros são exceções indicadas por **Error** e suas subclasses

Exceção em tempo de execução

- Estas são condições excepcionais que são internas ao aplicativo, e que o aplicativo geralmente não consegue antecipar.

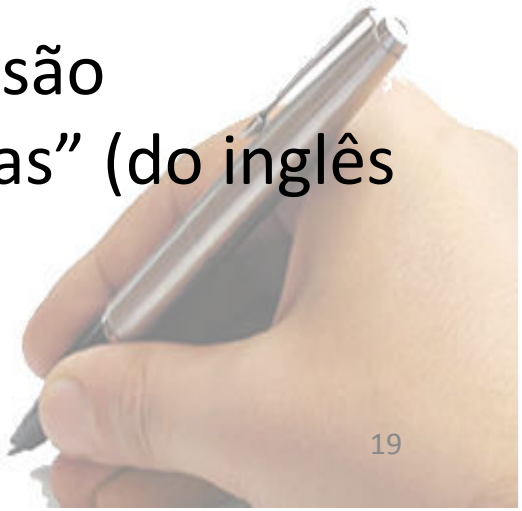
- Geralmente, indicam bugs de programação, com erros de lógica ou utilizações incorretas de uma API

Exceção em tempo de execução

- Exceções de tempo de execução não estão sujeitas ao Requisito Capturar ou Especificar.

- Exceções de tempo de execução são indicadas por **RuntimeException e suas subclasses.**

- Erros e exceções de tempo de execução são conhecidas como “exceções não checadas” (do inglês “unchecked exceptions”).



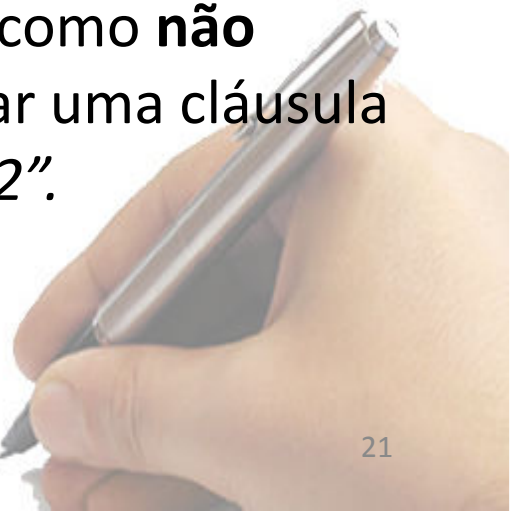
Exceções

- Toda **exceção verificada** deriva da class **Exception.**
- As **não verificadas** ou não-checadas, deriva da class **RuntimeException**.
- **Throwable –** é o pai de todas as exceções.
- **Error –** não são exceções e sim erros que jamais poderiam ter acontecido. ex.: estouro da memória.

Exceções

- **Exception-** as classes que deveriam lançar exceções e não erros de programação. *Exemplo: tentar abrir um arquivo que não existe. Então, é lançado uma exceção verificada, porque a classe de leitura de arquivos deriva de Exception.*

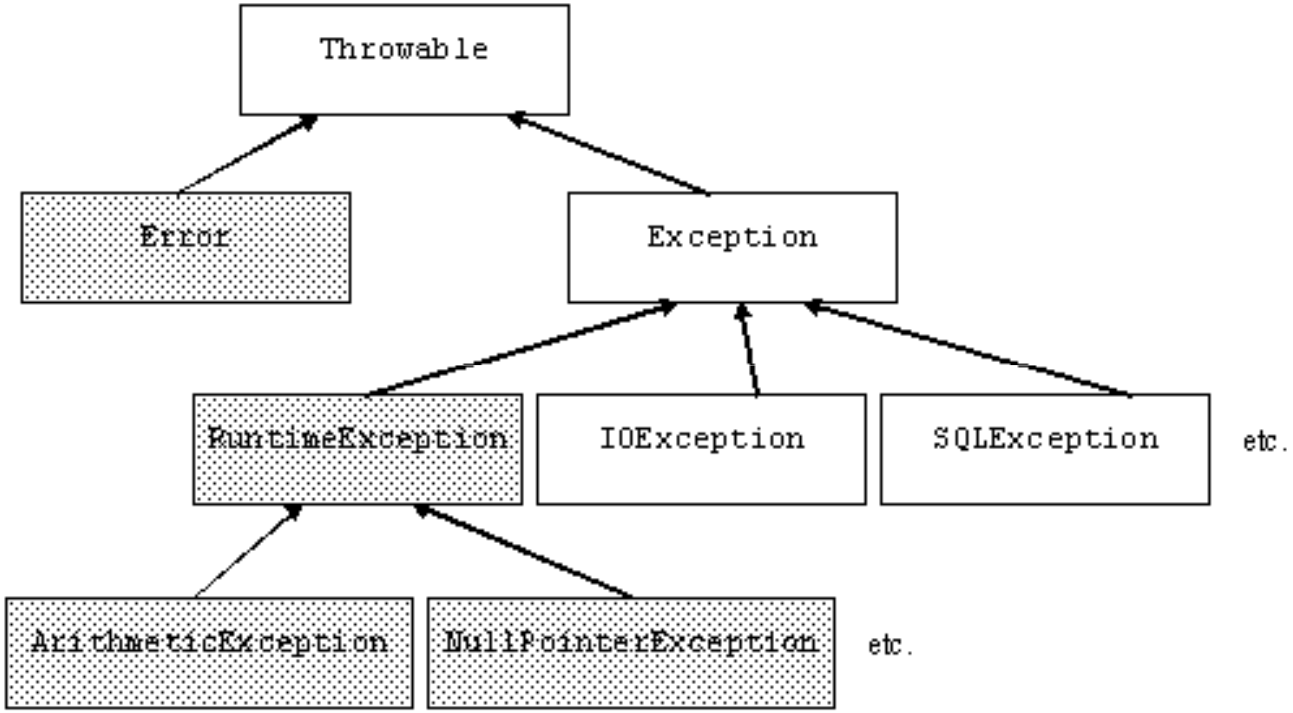
- ***RuntimeException* –** são exceções que indicam erros de programas (não de lógica, pois senão não passaria pelo compilador). Esse tipo de exceção é conhecido como **não verificada.** Sendo assim, não é requisito declarar uma cláusula try{} e catch{}. *Ex.: tentar converter "dois" em "2".*



Exceções

Obs.: Implicitamente, todas as classes em Java automaticamente já lançam uma exceção de RuntimeException.

Exceções



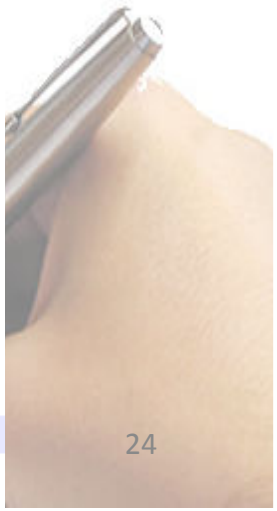
http://www.learn-java-tutorial.com/Java-Exceptions.cfm

Capturando e Relançando


```java
/**
 * Classe utilizada para exemplo na aula de exeções em java.
 * @author Rodrigo Fujioka - www.rodrigofujioka.com
 * @version 2010.
 */
public class Execoes {

    public void metodoPerigosoTratador(){

        try{
        /*
         * o seguinte trecho de código vai gerar uma exceção.
         * neste caso o programador pode identificar os pontos perigosos
         * e realizar um tratamento.
         */
        Integer inteiro = Integer.parseInt("er");

        }catch (Exception e) {
            System.out.println("Ocorreu algum erro na conversão de valor");
        }
    }

    public void metodoPerigosoLancador() throws Exception{

        /*
         * o seguinte trecho de código vai gerar uma exceção.
         * neste caso o programador pode identificar os pontos perigosos
         * e Relançar.
         */
        Integer inteiro = Integer.parseInt("er");
    }
}
```

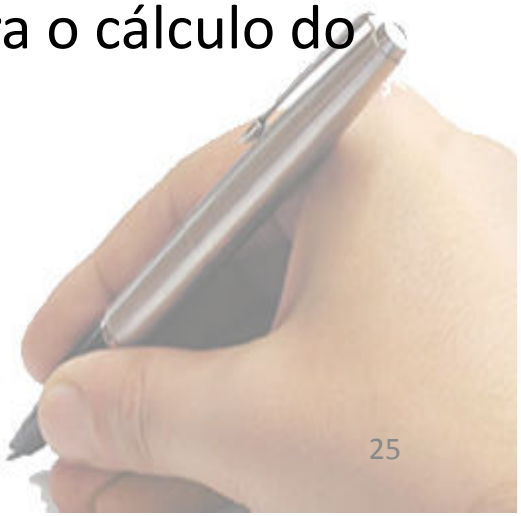
Trata

Lança



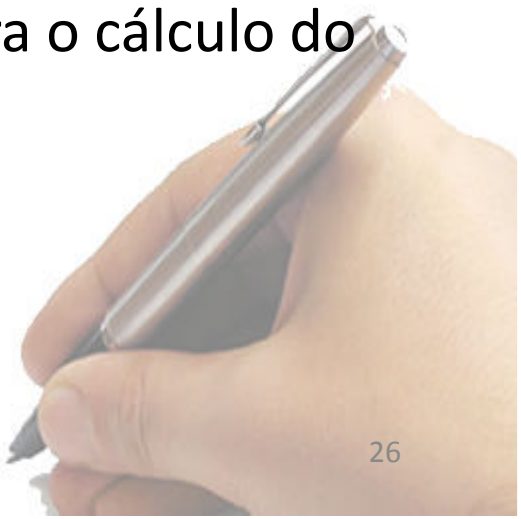
Regras para Gerar Exceção

- **Primeira regra:** coloque **no mínimo** uma mensagem explicativa na sua exceção. Quantas vezes você não ficou nervoso com o seu colega desenvolvedor, que jogou uma exceção sem mensagem *nenhuma* e não há como ter a mínima idéia do que aconteceu? Melhorando um pouco fica: throw new Exception("Um dos argumentos para o cálculo do desconto é Inválido");



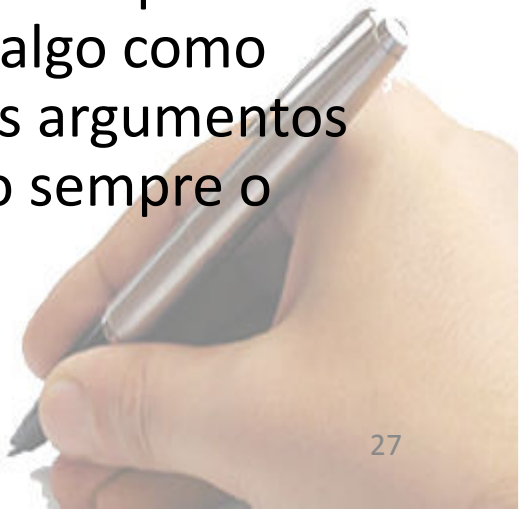
Regras para Gerar Exceção

- **Primeira regra:** coloque **no mínimo** uma mensagem explicativa na sua exceção. Quantas vezes você não ficou nervoso com o seu colega desenvolvedor, que jogou uma exceção sem mensagem *nenhuma* e não há como ter a mínima idéia do que aconteceu? Melhorando um pouco fica: throw new Exception("Um dos argumentos para o cálculo do desconto é Inválido");



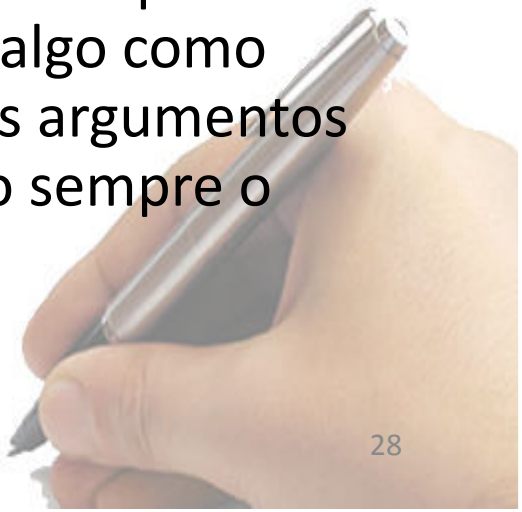
Regras para Gerar Exceção

- **Segunda regra**: use sempre exceções específicas ao seu caso, isto é, se o problema foi no cálculo de um desconto faça algo do tipo
- **throw new CalculoDeDescontoException**("Um dos argumentos é inválido");
- e não **throw new Exception**("Não há como calcular o desconto pois um dos argumentos é inválido");.
- Aqui você poderia terminar com **muitas** exceções específicas demais. Como alternativa, poderia então fazer algo como throw new LogicaDeNegocioException("Um dos argumentos para o calculo do desconto é inválido"). Usando sempre o bom senso.



Regras para Gerar Exceção

- **Segunda regra**: use sempre exceções específicas ao seu caso, isto é, se o problema foi no cálculo de um desconto faça algo do tipo
- **throw new CalculoDeDescontoException**("Um dos argumentos é inválido");
- e não **throw new Exception**("Não há como calcular o desconto pois um dos argumentos é inválido");.
- Aqui você poderia terminar com **muitas** exceções específicas demais. Como alternativa, poderia então fazer algo como throw new LogicaDeNegocioException("Um dos argumentos para o calculo do desconto é inválido"). Usando sempre o bom senso.



Regras para Gerar Exceção

Checked ou Unchecked?

Regras para Gerar Exceção

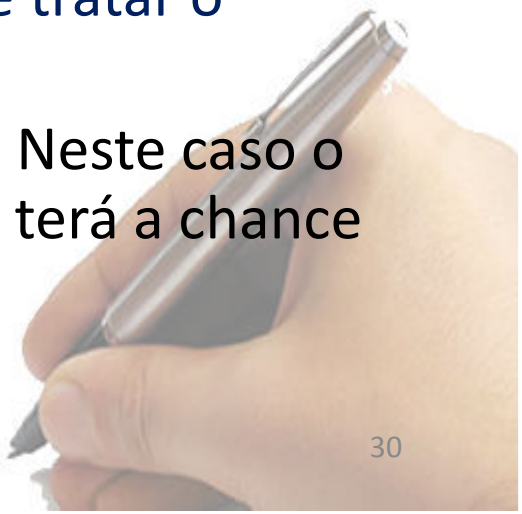
Pare um pouco para pensar no significado de um try-catch:

Uma chance de se recuperar do erro. Uma chance de tratar a exceção.

Tendo isso em mente fica mais fácil decidir entre lançar uma **checked** exception ou uma **unchecked** exception. Tente responder a seguinte pergunta:

Quero dar a chance a quem chama meu código de tratar o possível erro?

Se a resposta for sim, *use uma **checked** exception*. Neste caso o chamador vai ser obrigado a fazer o try-catch e terá a chance de se recuperar do erro.



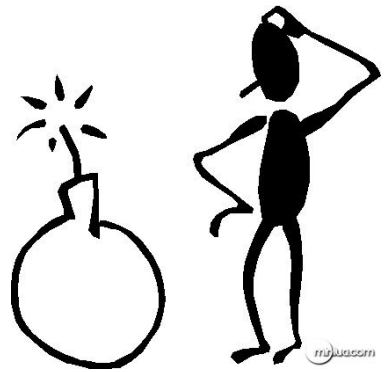
Regras para Gerar Exceção

Pare um pouco para pensar no significado de um try-catch:

Uma chance de se recuperar do erro. Uma chance de tratar a exceção.

Tendo isso em mente fica mais fácil decidir entre lançar uma **checked** exception ou uma **unchecked** exception. Tente responder a seguinte pergunta:

Quero dar a chance a quem chama meu código de tratar o possível erro?



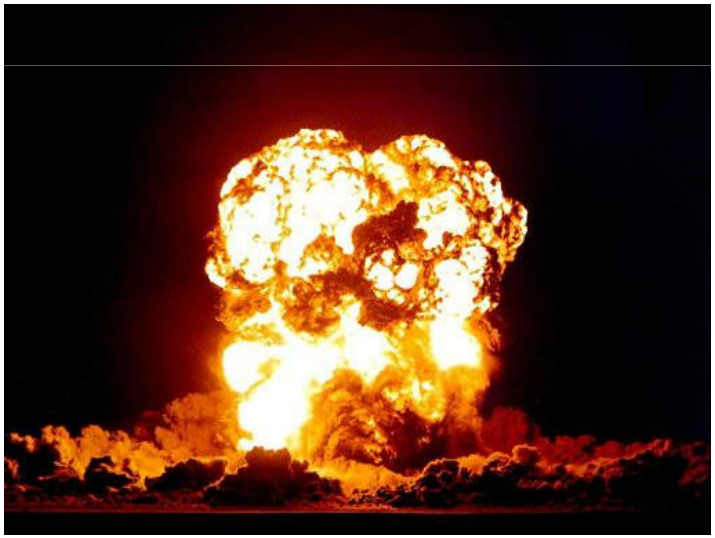
Regras para Gerar Exceção

Se a resposta for sim, *use uma **checked** exception*. Neste caso o chamador vai ser obrigado a fazer o try-catch e terá a chance de se recuperar do erro.

Regras para Gerar Exceção

Terceira regra: use **checked** exceptions para

erros recuperáveis.



Regras para Gerar Exceção

E quando os erros não são recuperáveis.

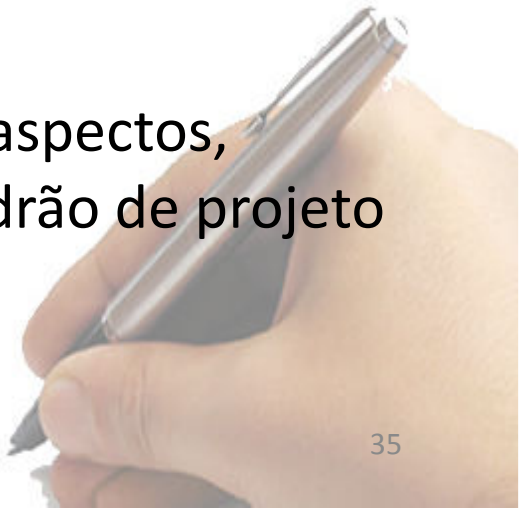
(não parece muito legal jogar o Stacktrace na cara de um usuário, parece?)

Regras para Gerar Exceção

- Implementar este componente interceptador não é tão difícil atualmente:

- Ambiente WEB: recursos como *Servlet Filters* que interceptam todas as requisições em um ambiente web ou tratamento de erro declarativo pertencente à especificação de servlets/jsp (seção *error-page* do web.xml).

- Ambientes desktop: programação orientada a aspectos, proxies ou coisas menos avançadas como o padrão de projeto *Decorator.*



Regras para Gerar Exceção

- Implementar este componente interceptador não é tão difícil atualmente:

- Ambiente WEB: recursos como *Servlet Filters* que interceptam todas as requisições em um ambiente web ou tratamento de erro declarativo pertencente à especificação de servlets/jsp (seção *error-page* do web.xml).

- Ambientes desktop: programação orientada a aspectos, proxies ou coisas menos avançadas como o padrão de projeto *Decorator.*

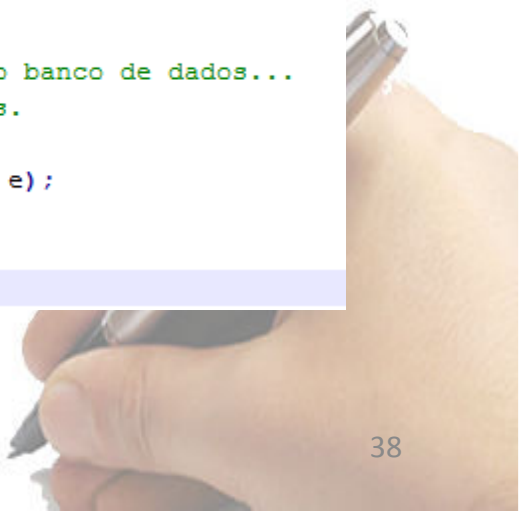
Regras para Gerar Exceção

- **Quarta regra:** use unchecked exceptions para erros irrecuperáveis.
- **Ei, a API que eu uso não segue estas regras!**
  Você, como bom samaritano, agora segue as boas práticas. Mas o JDBC por exemplo, não segue. Supondo que você seja obrigado a usá-lo:

Regras para Gerar Exceção

- **Quarta regra:** use unchecked exceptions para erros irrecuperáveis.
- **Ei, a API que eu uso não segue estas regras!**
  Você, como bom samaritano, agora segue as boas práticas. Mas o JDBC por exemplo, não segue. Supondo que você seja obrigado a usá-lo:

```
1   try {
2     PreparedStatement stmt = con.prepareStatement(query);
3     // ...
4   } catch (SQLException e) {
5     // Epa! SQLException é erro grave, irrecuperável! Pode ter queimado o banco de dados...
6     // Não deveria ser checked exception e, além disso, é genérica demais.
7     // Serve para muitos casos!
8     throw new AcessoADadosException("Problema na criação do Statement",  e);
9     // Agora sim, encapsulei a SQLException na minha exceção específica.
10    // Além disso ela é RuntimeException: unchecked!
11  }
```



Regras para Gerar Exceção

- Neste caso, AcessoADadosException é uma unchecked exception, já que o seu erro é irrecuperável e você não quer dar a chance de ninguém se recuperar dele. Além de que, agora a sua exceção é mais específica e descreve melhor o seu problema, sua API ficou mais limpa: não vão ser mais necessários try-catch(SQLException) nem throws SQLException espalhados pelo seu sistema todo.

- De fato, o caso mais comum é o de erros irrecuperáveis. Por isso essa prática de *transformar* checked exceptions (mal empregadas) em unchecked exceptions têm se tornado comum.

Regras para Gerar Exceção

- Neste caso, AcessoADadosException é uma unchecked exception, já que o seu erro é irrecuperável e você não quer dar a chance de ninguém se recuperar dele. Além de que, agora a sua exceção é mais específica e descreve melhor o seu problema, sua API ficou mais limpa: não vão ser mais necessários try-catch(SQLException) nem throws SQLException espalhados pelo seu sistema todo.

- De fato, o caso mais comum é o de erros irrecuperáveis. Por isso essa prática de *transformar* checked exceptions (mal empregadas) em unchecked exceptions têm se tornado comum.

Regras para Gerar Exceção

- **Quinta regra**: **nunca** faça:

```
} catch(Exception e) {
    System.out.println(e);
}
```

Regras para Gerar Exceção

- O código anterior pega **muito mais erros do que ele está realmente esperando** e o pior, **não está preparado para tratá-los**. Sempre relance erros que você não está preparado para tratar.

- Nunca pegue um erro e ignore-o. O rollback de uma transação pode depender da ocorrência de um erro irrecuperável que você acaba de esconder do resto do sistema!

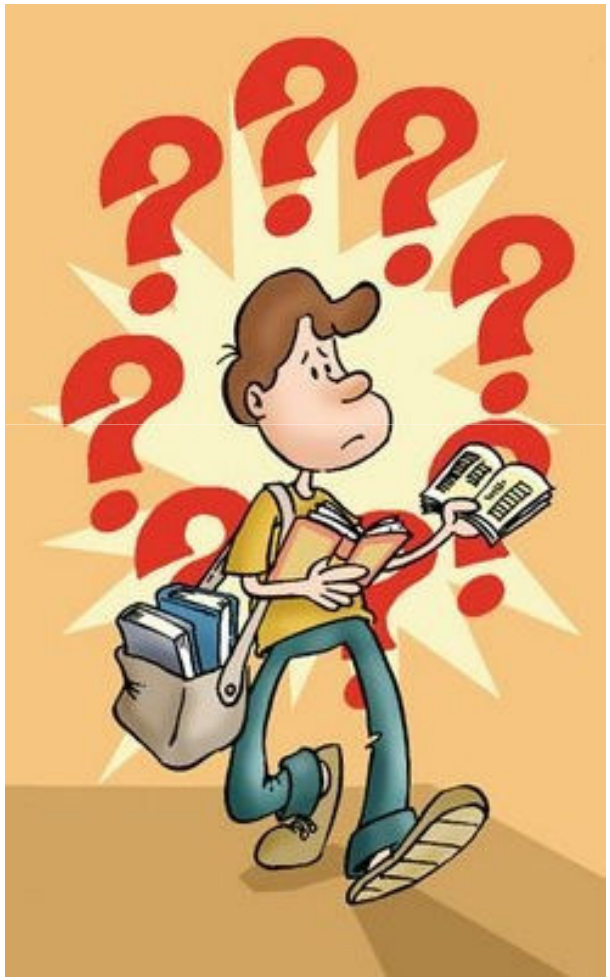
UnReachable Code.

*Execoes.java

```
1  /**
2   * Classe utilizada para exemplo na aula de
3   * exeções em java.
4   * @author Rodrigo Fujioka
5   * @author www.rodrigofujioka.com
6   * @version 2010.
7   */
8  public class Execoes {
9
10                          public void metodoPerigosoTratador(){
11
12                              try{
13                              /*
14                               * o seguinte trecho de código vai gerar uma exceção.
15                               * neste caso o programador pode identificar os pontos perigosos
16                               * e realizar um tratamento.
17                               */
18                              throw new Exception();
19                              Integer inteiro = Integer.parseInt("er");
20
21                              }catch (Exception e) {
22                                  System.out.println("Ocorreu algum erro na conversão de valor");
23                              }
24                          }
```

Multiple markers at this line
- The local variable inteiro is never read
- Unreachable code

Dúvidas?



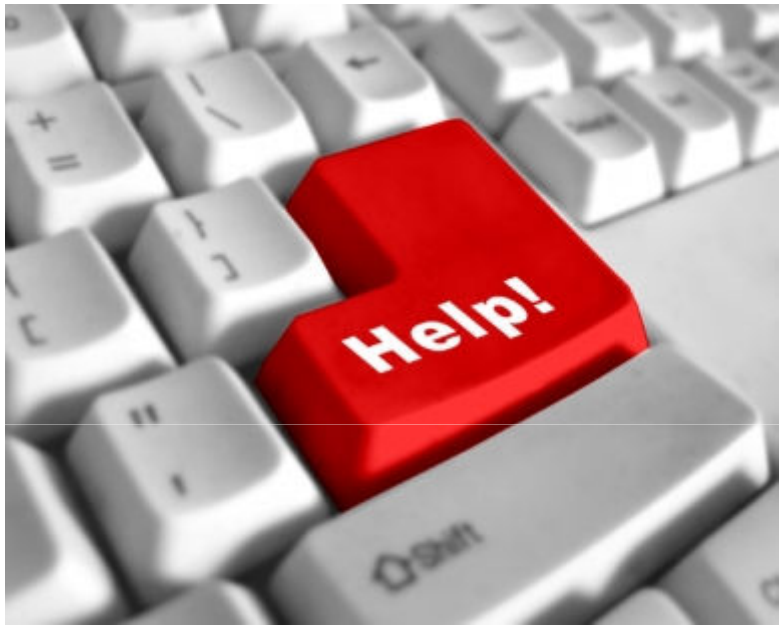
Links Complementares.

- http://wiki.locaweb.com.br/pt-br/Desvendando_Exception_JAVA
- http://java.dzone.com/articles/handling-exceptions-java-using
- http://www3.ntu.edu.sg/home/ehchua/programming/java/J5a_Exception.html
- http://blog.caelum.com.br/2006/10/07/lidando-com-exceptions/

- Livros do curso: Use A cabeça java,
- Java Como Programar, etc.

O Slide é apenas um Guia.

Ajuda!



rodrigofujioka@gmail.com
ou
rcf4@cin.ufpe.br