Disciplina: Linguagem de Programação II
Prof. Lucas Gonçalves Nadalete

Aula 10 - Tratamento de Exceções

Objetivos da Manipulação de Exceção

- Detecção mais fácil de erros sem a escrita de um código especial para testar valores retornados
- Manter um código de manipulação de exceções nitidamente separado do código que gerará a exceção
- Lidar com as diferentes exceções possíveis usando o mesmo código de manipulação de exceções

Contextualização

- O tratamento de erros é uma característica fundamental, pois é praticamente impossível criar uma aplicação totalmente livre de erros
- Erros que não necessariamente são ocasionados por falhas do desenvolvimento
- Porém, se não tratados, os erros oriundos daquelas situações previstas, devem ser consideradas falhas no desenvolvimento de software
- Com isso, faz-se necessário "preparar alguma resposta nestas situações"

Exemplos

- Em uma aplicação que se comunica com o banco de dados, este pode estar fora do ar em determinada situação
- Um arquivo de configuração poder ter sido acidentalmente removido
- O usuário pode ter digitado um valor inválido

Introdução

- Por exemplo, se o BD estiver fora do ar, o que fazer?
  - A mensagem de erro deve ser impressa no console?
  - A mensagem de erro deve ser impressa em uma área visível pelo usuário (interface gráfica, aplicação web ...) ?
  - Um e-mail deve ser enviado ao administrador do Banco de Dados ?
  - A operação deve ser logada em um arquivo?
  - Os dados do usuário devem ser salvos em um arquivo temporário?

O que fazer?

- Não existe uma resposta certa para esta pergunta, pois tudo depende da situação
- Porém, não podemos simplesmente ignorar o erro...
- Infelizmente, não é dada a devida atenção ao tratamento de erros de aplicações
- Todavia, é uma ótima prática de desenvolvimento

POO X PE

- Na **programação estruturada**, desenvolver programas que lidam com erros é monótono e transforma o código-fonte do aplicativo em um **emaranhado e confuso**

- Na **programação orientada a objetos**, especificamente na Linguagem Java, existe um mecanismo sofisticado para manipulação de erros que produz códigos de manipulação eficientes e organizados: **a manipulação de exceções**

Exceptions

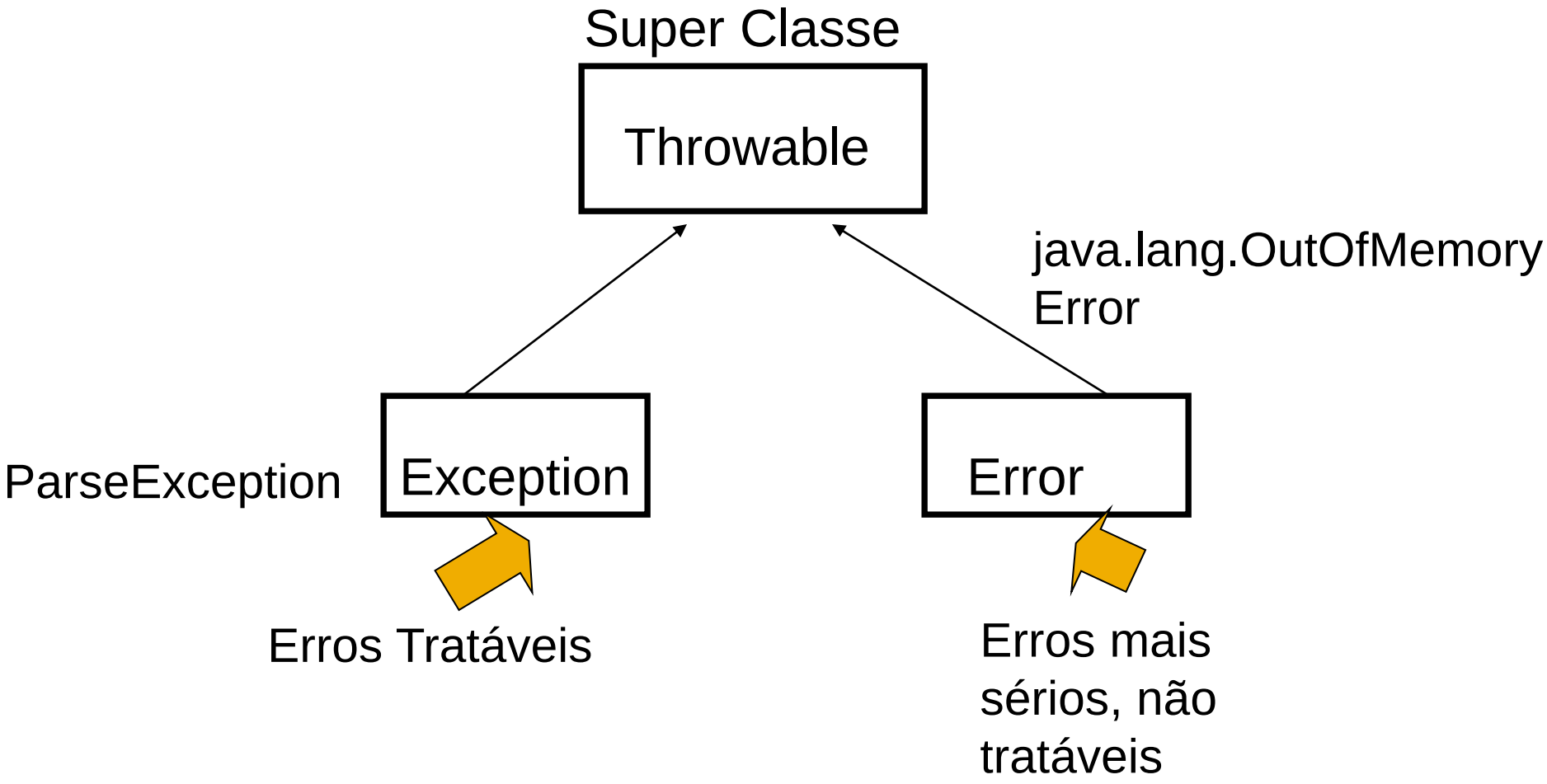
- Chamamos de *Exception* o mecanismo de tratamento de erros na plataforma Java

- Uma *Exception* é a indicação de que algum problema ocorreu durante a execução do programa

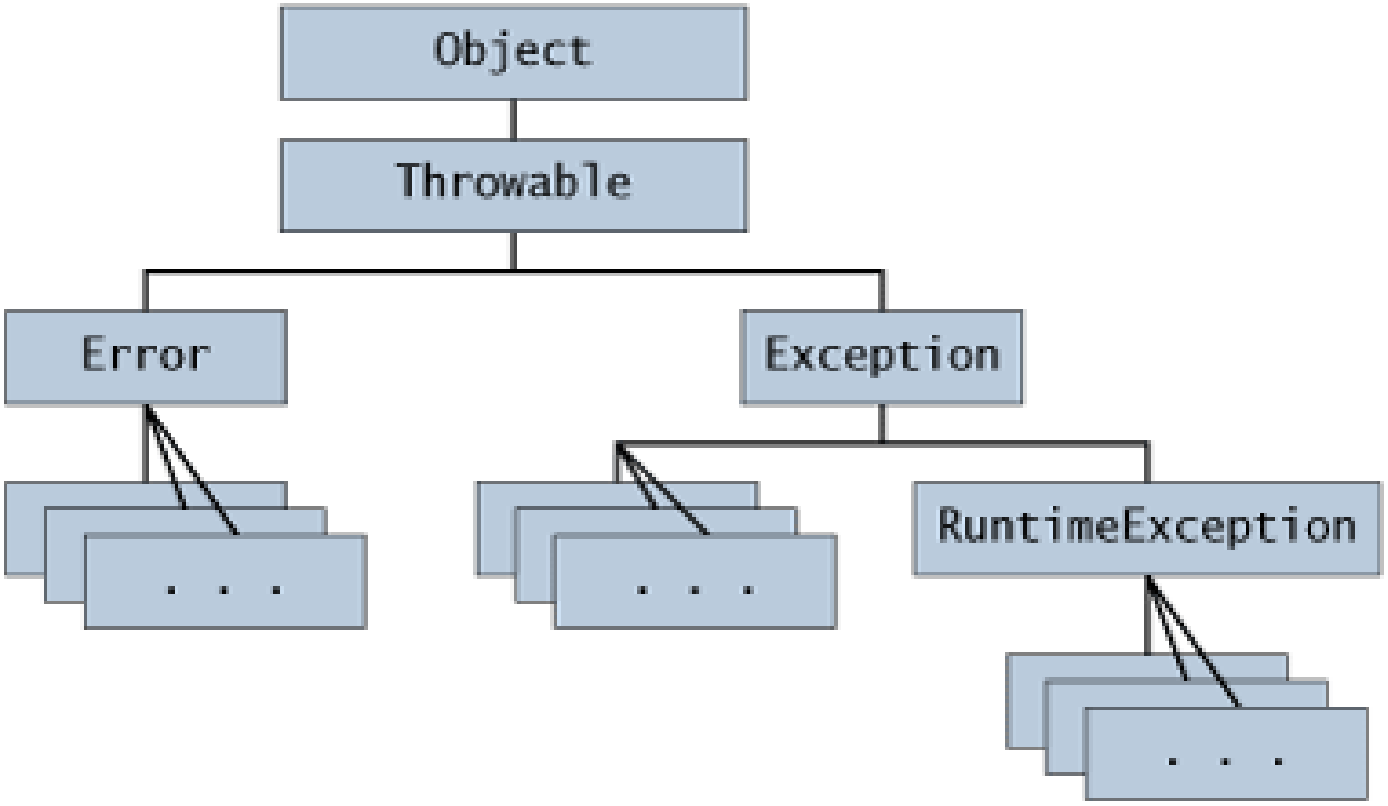
Exceptions (2)

- *Exceptions* são classes comuns em Java e devem tratar os erros passíveis de ocorrer.
- Uma única *Exception* seria uma saída míope para resolvê-los, pois existem muitos tipos nativos da linguagem cada um tratando um conjunto específico de problemas, por exemplo:
  - `SQLException, IOException, NullPointerException, NumberFormatException, ArrayIndexOutOfBoundsException...`

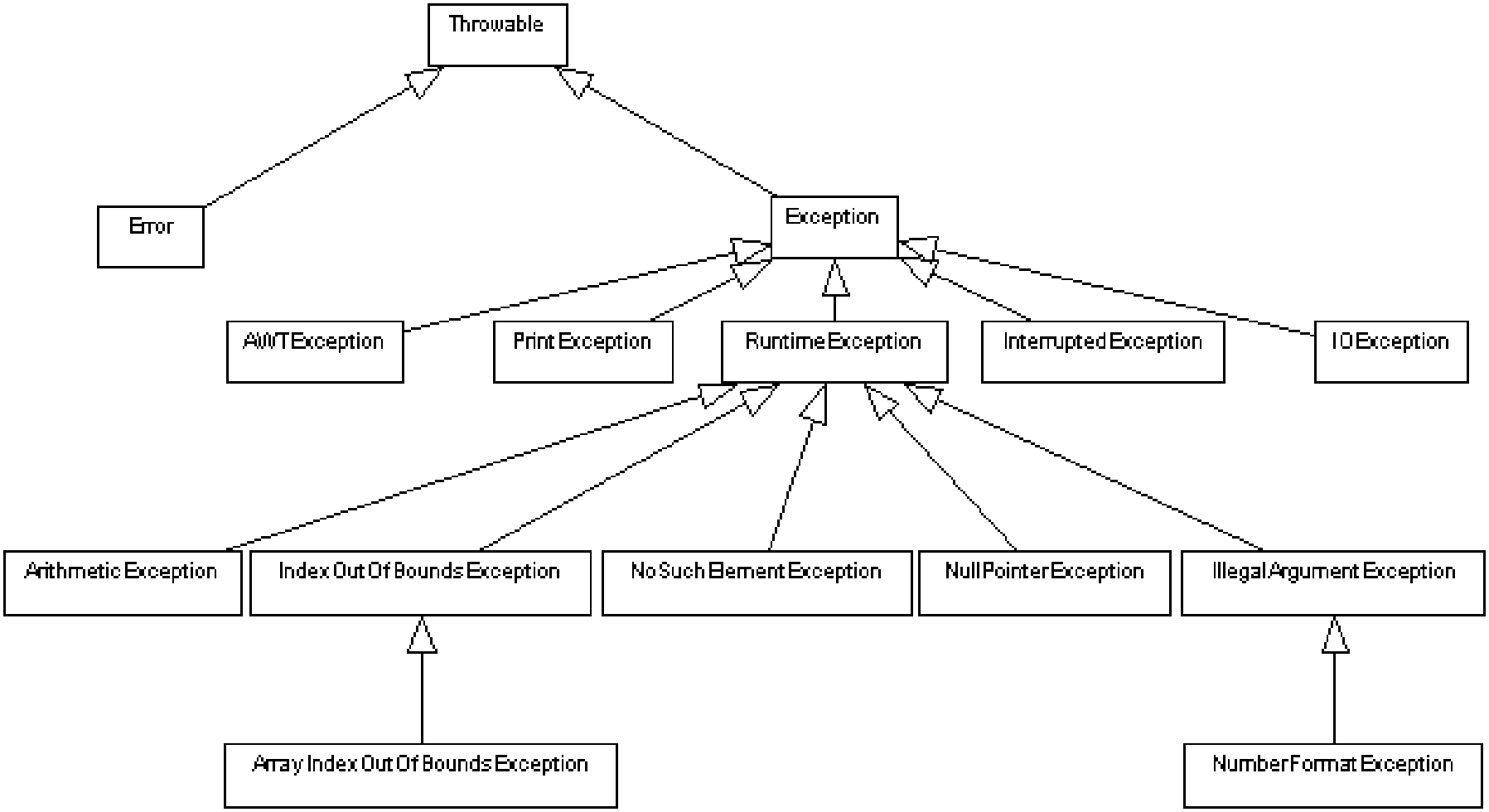
Hierarquia das Exceptions



Hierarquia de Exceptions em Java



Hierarquia de Exceptions



Exemplo de Classes de Exceções

Problema	Classe que identifica o problema
Índice de um Array fora do intervalo permitido	IndexOutOfBoundsExceptions
Problemas em operações aritméticas tal como dividir por zero	ArithmeticException
Uso de referência que não aponta para nenhum objeto	NullPointerException

Tipos de Exceção em Java

- Java possui dois tipos de exceções
- ***Checked Exceptions*** são exceções que devem ser usadas para modelar falhas contornáveis.
  - Devem sempre ser declaradas pelos métodos que as lançam e precisam ser tratadas (a menos que explicitamente passadas adiante)
- ***Unchecked Exceptions*** são exceções que devem ser usadas para modelar falhas incontornáveis.
  - Não precisam ser declaradas e nem tratadas

Unchecked Exceptions

- São exceções que devem ser usadas para modelar falhas incontornáveis
- Não precisam ser declaradas nem tratadas e nem lançadas
- Todas são filhas de *RuntimeException*

Unchecked Exceptions

- Para criarmos uma classe que modela uma *unchecked exception*, devemos estender a classe *Error* ou *RuntimeException*
- Esse tipo de exceção não será verificada pelo compilador
- Tipicamente não criamos exceções desse tipo, elas são usadas pela própria linguagem para sinalizar condições de erro

Checked Exceptions

- Todas as exceções que **não são filhas** da classe ***RuntimeException***
- Neste tipo, o desenvolvedor **é obrigado a realizar** alguma operação **no caso de erro**
- Atenção: Um código com um método que lance uma checked exception sem tratá-lo em algum local não compila!
  - Ex. Método sleep() da Classe Thread

Throws

- Indicamos que um método poderá efetivamente lançar uma Exception utilizando-se a clásula throws
- Para capturar uma exceção, utilizamos a cláusula try-catch

Sintaxe Try-catch

- **try**: é usada para indicar/cercar um bloco de código que possa ocorrer uma exceção
- **catch**: serve para manipular as exceções, ou seja, tratar o erro propriamente
- **finally**: O importante é saber que esse bloco sempre será executado, útil para liberação de recursos, registros de logs, etc.

Sintaxe try-catch

Combinações possíveis para a sintaxe try-catch do Java:

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

Combinações válidas

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

Combinações inválidas

Exemplo

```
import java.io.*;
public class C16Exemplo11 {
        public static void main(String[] args) {
                int num = 0, den = 0;
                BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("Digite dois números seguidos da tecla ENTER:");
                try {
                        num = Integer.parseInt (teclado.readLine());
                        den = Integer.parseInt(teclado.readLine());
                        System.out.println (num+"/"+den+" = "+ (num/den));
                }
                catch (NumberFormatException e){
                        System.out.println ("Erro de formato.");
                }
                catch (IOException e) {
                        System.out.println ("Erro de E/S."); }
                finally {
                        System.out.println("Saindo do Bloco \"Try\".");}

                System.out.println("Fim do programa");

                }
}
```

Trecho de código tratado

Tratamento explícito da exceção

Trecho que será sempre executado

A Cláusula throws

- Os **erros que devem ser tratados são lançados** em métodos ou construtores de classes
  - É necessário sinalizar esta possibilidade, para que o desenvolvedor tenha o conhecimento que deve tratá-los corretamente
- A indicação que estas **falhas podem acontecer** são definidas pela existência da clásula **throws**
- Use quando **o erro não for de responsabilidade da classe**.
  - Ela simplesmente será executada e lançará qualquer erro para ser mais adequadamente tratado em uma classe própria para isso.

Lançando Exceções

- O throws é usado da forma descrita a seguir para listar as exceções que um método pode lançar:

```
1 package br.com.grupoew.exceptionsEmJava;
2
3 public class EntendendoExceptions {
4     void myFunction() throws MyException1, MyException2 {
5         // Código do método
6     } //myFunction
7 } //EntendendoExceptions
```

Pode lançar mais de um tipo de exceção

Exceções lançadas pela JVM

- Exemplo: NullPointerException

```
1  package br.com.grupoew.exceptionsEmJava;
2  public class NPE {
3      static String s;
4      public static void main(String [] args) {
5          System.out.println(s.length());
6      } //main
7  } //NPE
```

- Neste caso, o programa tenta acessar um objeto usando uma variável de referência com um valor atual **null**. O compilador não encontra problemas desse tipo antes do tempo de execução.

Exceções lançadas pela JVM

- StackOverflowException

  Exemplo de Recursividade infinita:

```
public void go() {
    go();  // recursão
}
```

  - Iniciando a pilha de chamadas, **main** chamará o **go()**, que chamará **go()**, que chamará **go()**… não há JVM/SO que aguente!

Repassando Exceção

- Se quisermos usar o método div sem tratar a exceção, deve-se declarar que a exceção deve ser passada adiante:

```
public void f() throws DivByZeroException {
        Calc calc = new Calc();
        int div = calc.div(a,b);
        System.out.println(div);
}
```

O método div desenvolvido lança uma exceção e deve ser tratado!

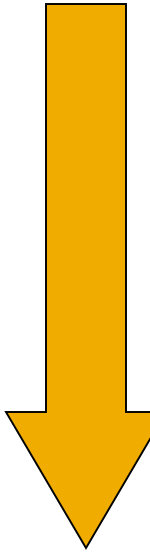
Tratando Múltiplas Exceções

```
    try {
    ...
    } catch (Exception1 e1) {
    ...
    } catch (Exception2 e2) {
    ...
    } finally {
    ...
}
```

Tratando Múltiplas Exceções

- Devemos declarar primeiramente as Exceptions mais específicas, ou seja, as classes filhas, e depois, as mais genéricas.

Ex:
```
try { ... }
        catch (ClasseExceptionFilha) {}
        catch (ClasseExpcetionPai) {}
        catch (ClasseExceptionAvo) {}
```



Multicatch (jdk > 7)

```
catch (IOException ex) {
     logger.log(ex);
     throw ex;
catch (SQLException ex) {
     logger.log(ex);
     throw ex;
}
```

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

Métodos da Classe Throwable

- **printStackTrace**(): É amplamente usado, sendo bastante útil para os desenvolvedores. Ele dá uma visão detalhada do erro, indicando qual método originou a falha, assim como cada método que a chamaram. Essa informação é impressa no console.
- **getMessage**(): Análogo ao anterior, porém não fornece detalhes da origem do erro e das chamadas dos métodos, apenas informa por meio de uma mensagem o tipo da exceção ocorrida.
- O printStackTrace é muito mais útil ao programador

Criando minhas Exceptions

- Como vimos, a API do Java já tem um conjunto de classes específicas que tratam diversos tipos de exceção
- Mas e se quisermos criar a nossa própria classe de Exceção, tem como?
  - Sim, claro!
- Lembrando-se que Exception é uma classe qualquer e, portanto, filha de Object. Logo, para usá-la basta estender a classe Exception.

Minha Exceção

```
public class FatecException extends Exception {
   public FatecException (String msg) {
      super(msg); //informa a msg para getMessage()
      metodoPersonalizado();
  }
   public void metodoPersonalizado() {
      System.out.println("Trate sua exceção como
quiser");
      System.out.println(getMessage());
  }
}
```

Questões

- O código a seguir é possível?

```
try{ }
finally { }
```

- Quais tipos de exceções podem ser tratadas pelo seguinte código:

```
catch (Exception e) { }
```

- O que há de errado em usar esse tipo de tratamento de exceção?

Questões

- O que há de errado com o seguinte trecho de código? Esse código vai compilar?

```
try { }
catch (Exception e) { }
catch (ArithmeticException a) { }
```