Centro Tecnológico
Departamento de Informática

Prof. Vítor E. Silva Souza
http://www.inf.ufes.br/~vitorsouza

[Desenvolvimento OO com Java]

Exceções e controle de erros

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Conteúdo do curso

- O que é Java;
- Variáveis primitivas e controle de fluxo;
- Orientação a objetos básica;
- Um pouco de vetores;
- Modificadores de acesso e atributos de classe;
- Herança, reescrita e polimorfismo;
- Classes abstratas e interfaces;
- Exceções e controle de erros;
- Organizando suas classes;
- Utilitários da API Java.

Estes slides foram baseados na apostila do curso FJ-11: Java e Orientação a Objetos da Caelum e na apostila Programação Orientada a Objetos em Java do prof. Flávio Miguel Varejão.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

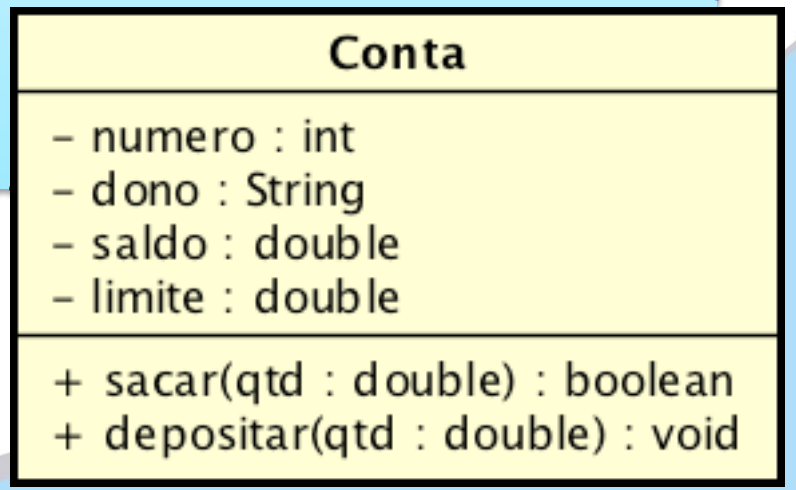
Controle de erros

```
class Conta {
  // ...

  boolean sacar(double qtd) {
     if (saldo < qtd) return false;

     saldo = saldo - qtd;
     return true;
  }
}
```

Responsabilidade no lugar certo!



Controle de erros

```
public class CaixaEletronico {
  public static void main(String[] args) {
     Conta minhaConta = new Conta();
     minhaConta.saldo = 1000;
     if (minhaConta.sacar(2000))
        System.out.println("Liberar a grana...");
     else
        System.out.println("Mostrar erro...");
  }
}
```

Responsabilidade no lugar certo?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Controle de erros

```
public class CaixaEletronico {
  public static void main(String[] args) {
     Conta minhaConta = new Conta();
     minhaConta.saldo = 1000;

     // Não é minha responsabilidade!
     minhaConta.sacar(2000)
     System.out.println("Liberar a grana...");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

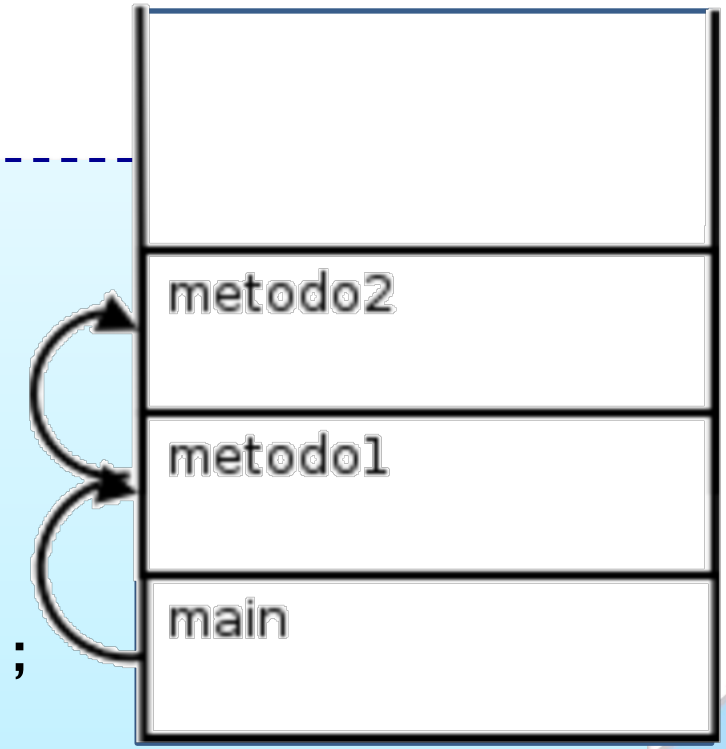
Alternativas para controle de erros

- Deixar o programa abortar;
- Teste e tratamento imediato;
- Retorno de código de erro (*magic numbers*);
- Variável global (`errno` em C);
- Parâmetro de saída;
- API de tratamento de erros (`raise()`, `signal()`, `setjmp()`, `longjmp()` em C);
- Mecanismo de exceções.

Uma exceção representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.

Exceções em ação

```
public class TesteErro {
  public static void main(String[] args) {
     System.out.println("inicio do main");
     metodo1();
     System.out.println("fim do main");
  }
  static void metodo1() {
     System.out.println("inicio do metodo1");
     metodo2();
     System.out.println("fim do metodo1");
  }
  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     for (int i = 0; i <= 10; i++) System.out.println(vet[i]);
     System.out.println("fim do metodo2");
  }
}
```



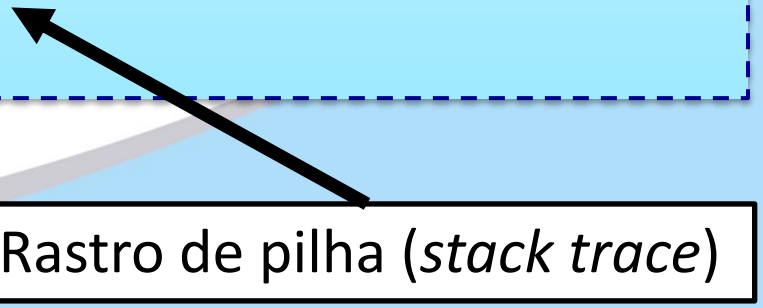
Pilha de execução

Qual a saída?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções em ação

```
inicio do main
inicio do metodo1
inicio do metodo2
1
2
3
4
5
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
  at TesteErro.metodo2(TesteErro.java:15)
  at TesteErro.metodo1(TesteErro.java:9)
  at TesteErro.main(TesteErro.java:4)
```



Exceções em ação

```
public class TesteErro {
  public static void main(String[] args) {
     System.out.println("inicio do main");
     metodo1();
     System.out.println("fim do main");
  }
  static void metodo1() {
     System.out.println("inicio do metodo1");
     metodo2();
     System.out.println("fim do metodo1");
  }
  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     for (int i = 0; i <= 10; i++) System.out.println(vet[i]);
     System.out.println("fim do metodo2");
  }
}
```

4: Não tratada, relançada... JVM (thread) morre!

9: Não tratada, relançada...

15: Exceção lançada, não tratada...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

O que é uma exceção?

- Condição provocada por uma situação excepcional que requer uma ação específica e imediata;
- Desvia o fluxo de controle do programa para um código de tratamento;
- Podem ser causadas por diversas condições:
  - *Erros sérios de hardware;*
  - *Erros simples de programação;*
  - *Condições esperadas (não são erros).*

Nosso exemplo didático é deste tipo.

Mas vamos fingir que ele é deste tipo...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tratando exceções

```
public class TesteErro {
  // ...

  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     try {
        for (int i = 0; i <= 10; i++)
           System.out.println(vet[i]);
     }
     catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("erro: " + e);
     }
     System.out.println("fim do metodo2");
  }
}
```

Qual a saída?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Tratando exceções

```
inicio do main
inicio do metodo1
inicio do metodo2
1
2
3
4
5
erro: java.lang.ArrayIndexOutOfBoundsException: 5
fim do metodo2
fim do metodo1
fim do main
```

Tratando exceções

```
public class TesteErro {
  // ...

  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     try {
        for (int i = 0; i <= 10; i++)
           System.out.println(vet[i]);
     }
     catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("erro: " + e);
     }
     System.out.println("fim do metodo2");
  }
```

Bloco supervisionado

Exceção lançada, busca tratamento

Captura (catch) da exceção, especificando a classe...

Bloco de tratamento. Exceção é disponibilizada como instância da classe (objeto!)

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Onde tratar a exceção?

```
public class TesteErro {
  // ...

  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     for (int i = 0; i <= 10; i++)
        try {
           System.out.println(vet[i]);
        }
        catch (ArrayIndexOutOfBoundsException e) {
           System.out.println("erro: " + e);
        }
     System.out.println("fim do metodo2");
  }
}
```

Qual a diferença?

Onde tratar a exceção?

```
inicio do main
inicio do metodo1
inicio do metodo2
1
2
3
4
5
erro: java.lang.ArrayIndexOutOfBoundsException: 5
erro: java.lang.ArrayIndexOutOfBoundsException: 6
erro: java.lang.ArrayIndexOutOfBoundsException: 7
erro: java.lang.ArrayIndexOutOfBoundsException: 8
erro: java.lang.ArrayIndexOutOfBoundsException: 9
erro: java.lang.ArrayIndexOutOfBoundsException: 10
fim do metodo2
fim do metodo1
fim do main
```

Atributo do objeto exceção:

Onde tratar a exceção?

```
public class TesteErro {
  // ...
  static void metodo1() {
     System.out.println("inicio do metodo1");
     try {
        metodo2();
     } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("erro: " + e);
     }
     System.out.println("fim do metodo1");
  }
  static void metodo2() {
     System.out.println("inicio do metodo2");
     int[] vet = {1, 2, 3, 4, 5};
     for (int i = 0; i <= 10; i++) System.out.println(vet[i]);
     System.out.println("fim do metodo2");
  }
}
```

E agora?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Onde tratar a exceção?

```
inicio do main
inicio do metodo1
inicio do metodo2
1
2
3
4
5
erro: java.lang.ArrayIndexOutOfBoundsException: 5
fim do metodo1
fim do main
```

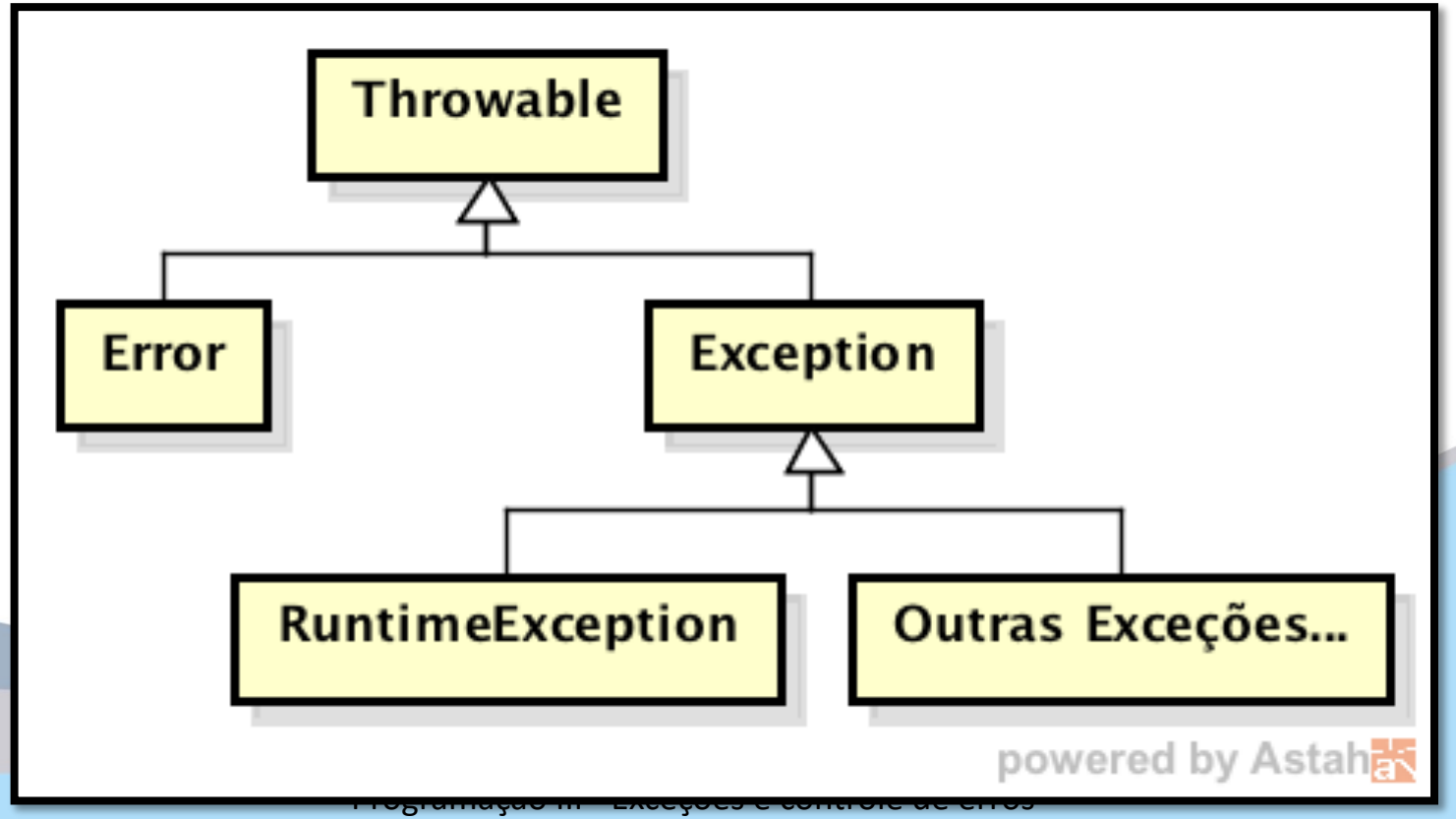
`metodo2()` não termina!

E se eu levo o tratamento para a `main()`?

O que você vai colocar dentro do `try` influencia muito a execução do programa!

Exceções

- Exceções, como (quase) tudo em Java, são objetos;
- Porém são objetos especiais: podem ser manipulados pelo mecanismo de exceções.



UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

java.lang.Throwable

- Ancestral de todas as classes que recebem tratamento do mecanismo de exceções;
- Principais métodos:
  - *`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;*
  - *O método `toString()` também é implementado e retorna uma descrição breve da exceção.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

`java.lang.Error`

- Representa um problema grave, de difícil (ou impossível) recuperação;
- Exemplos:
  - *`OutOfMemoryError`, `StackOverflowError`, etc.*
- Geralmente causam o encerramento do programa;
- Não devem ser usadas pelos programadores.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

`java.lang.Exception`

- Exceções que podem ser lançadas pelos métodos da API Java ou pelo seu programa;
- Devem ser tratadas;
- Em geral, representam situações excepcionais, porém esperadas e contornáveis;
- O programador tem contato com esta classe e suas subclasses.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

java.lang.RuntimeException

- Tipo especial de exceção;
- Não necessitam ser lançadas explicitamente pelo programa;
- Seu tratamento não é obrigatório (não checadas);
- Ex.:
  - *NullPointerException;*
  - *ArrayIndexOutOfBoundsException;*
  - *ClassCastException;*
  - *etc.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

`Exception` x `RuntimeException`

- `RuntimeException` (*unchecked exceptions*):
  - *Dão menos trabalho para codificar;*
  - *Geralmente representam bugs;*
  - *O código deve ser consertado para que ela não mais ocorra.*
- `Exception` (*checked exceptions*):
  - *Aumentam a confiabilidade do código;*
  - *Geralmente representam situações esperadas;*
  - *Em seu tratamento, a situação deve ser contornada e o programa continua a funcionar.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exemplos de RuntimeException

```
// java.lang.ArithmeticException: / by zero
int i = 1 / 0;

// java.lang.NullPointerException
Object o = null;
System.out.println(o.hashCode());

// java.lang.ArrayIndexOutOfBoundsException
int[] vet = {1, 2, 3, 4, 5};
for (int i = 0; i <= 10; i++) System.out.println(vet[i]);
```

Porque eu não sou obrigado!

Exceções checadas

```
public class TesteErro {
  public static void main(String[] args) {
     new java.io.FileInputStream("arquivo.txt");
  }
}
```

```
TesteErro.java:3: error: unreported exception
FileNotFoundException; must be caught or declared to be thrown
     new java.io.FileInputStream("arquivo.txt");
     ^
1 error
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções checadas

```
import java.io.FileNotFoundException;

public class TesteErro {
  public static void main(String[] args) {
     try {
        new java.io.FileInputStream("arquivo.txt");
     }
     catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado");
     }
  }
}
```

Exceções checadas

```
import java.io.FileNotFoundException;

public class TesteErro {
  public static void main(String[] args)
                                    throws FileNotFoundException {
     new java.io.FileInputStream("arquivo.txt");
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Lançando/delegando exceções

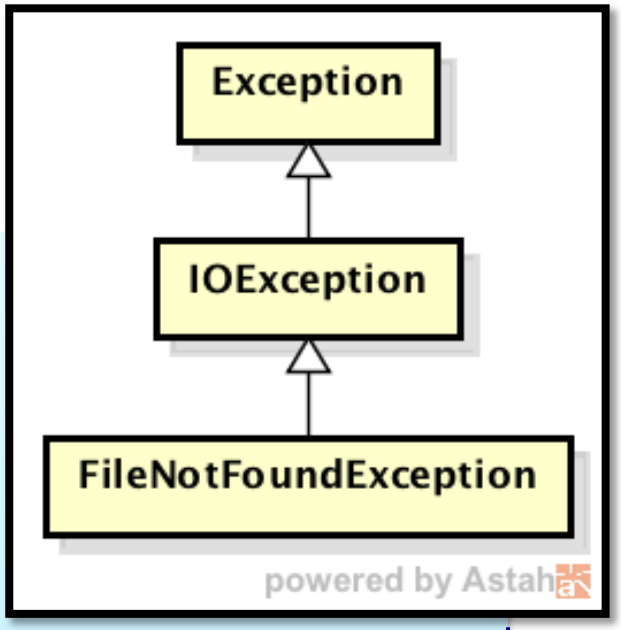
```
import java.io.FileNotFoundException;

public class TesteErro {
  public static void main(String[] args) {
     metodo1();
  }
  static void metodo1() {
     try {
        metodo2();
     }
     catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado");
     }
  }
  static void metodo2() throws FileNotFoundException {
     new java.io.FileInputStream("arquivo.txt");
  }
}
```

Imagine múltiplas camadas:
GUI, serviço, acesso a dados...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Múltiplos catch



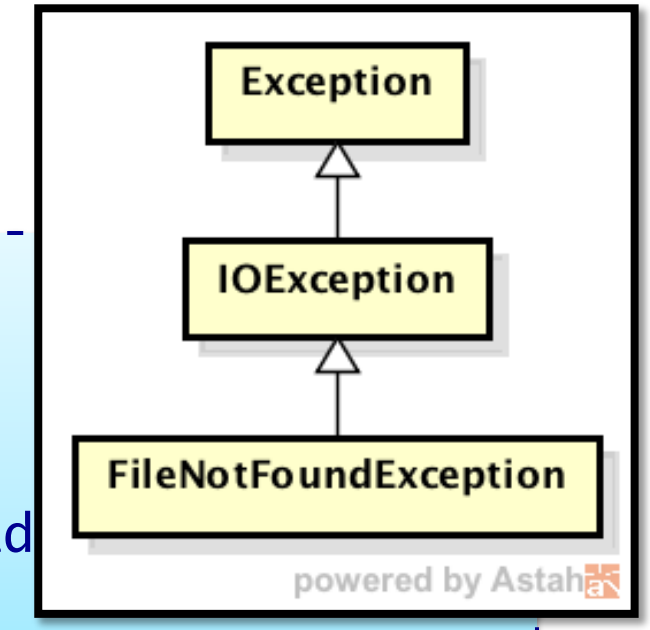

```
import java.io.FileNotFoundException;
import java.io.IOException;

public class TesteErro {
  public static void main(String[] args) {
     try {
        new java.io.FileInputStream("arquivo.txt");
     }
     catch (IOException e) {
        System.out.println("Erro de I/O");
     }
     catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado");
     }
  }
}
```

Qual a saída?

Múltiplos catch

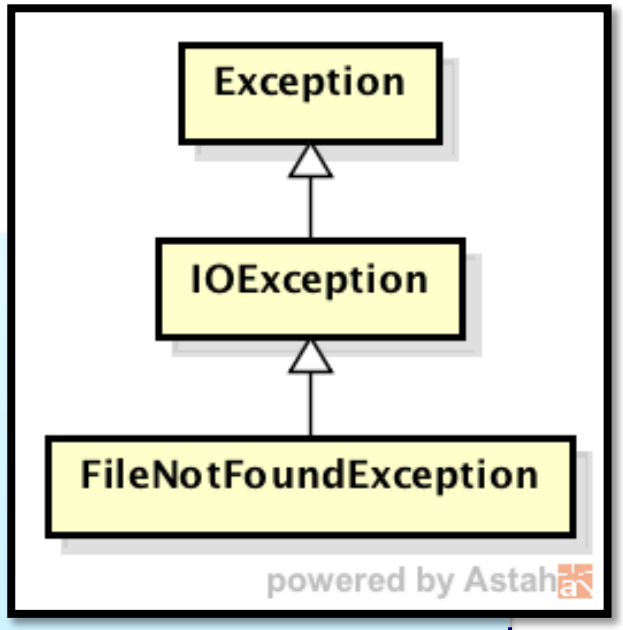
```
    catch (IOException e) {
       System.out.println("Erro de I/O");
    }
    catch (FileNotFoundException e) {
       System.out.println("Arquivo não encontrad
    }
```




```
TesteErro.java:12: error: exception FileNotFoundException has
already been caught
    catch (FileNotFoundException e) {
    ^
1 error
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Múltiplos catch




```
import java.io.FileNotFoundException;
import java.io.IOException;

public class TesteErro {
  public static void main(String[] args) {
     try {
        new java.io.FileInputStream("arquivo.txt");
     }
     catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado");
     }
     catch (IOException e) {
        System.out.println("Outros erros de I/O");
     }
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

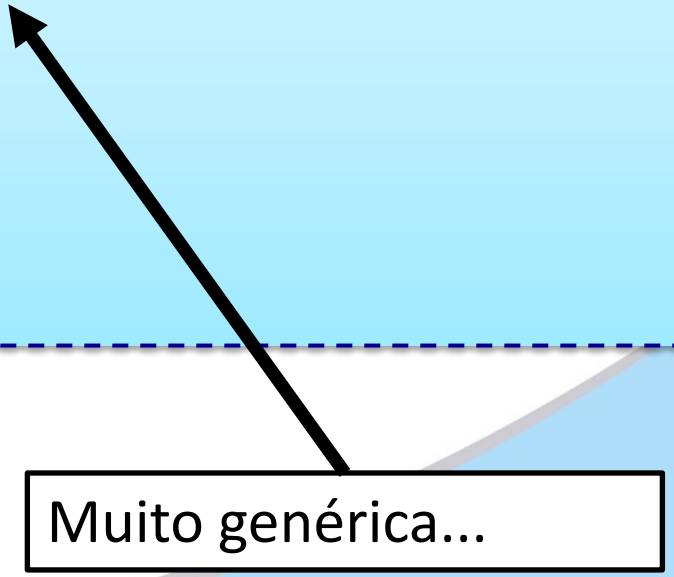
Lançando uma exceção

Conta
- numero : int - dono : String - saldo : double - limite : double
+ sacar(qtd : double) : boolean + depositar(qtd : double) : void

```
class Conta {
  // ...

  boolean sacar(double qtd) {
    if (saldo < qtd) throw new RuntimeException();

    saldo = saldo - qtd;
    return true;
  }
}
```



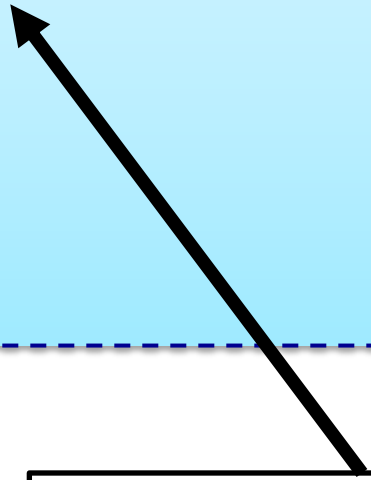
Lançando uma exceção

Conta
- numero : int - dono : String - saldo : double - limite : double
+ sacar(qtd : double) : boolean + depositar(qtd : double) : void

```
class Conta {
  // ...

  boolean sacar(double qtd) {
    if (saldo < qtd) throw new IllegalArgumentException();

    saldo = saldo - qtd;
    return true;
  }
}
```



Um pouco melhor, mas é não checada...

O que acontece se esquecerem de tratar?

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Lançando uma exceção

Conta
- numero : int - dono : String - saldo : double - limite : double
+ sacar(qtd : double) : boolean + depositar(qtd : double) : void

```
class Conta {
  // ...

  boolean sacar(double qtd) {
     if (saldo < qtd)
        throw new IllegalArgumentException("Sem saldo!");

     saldo = saldo - qtd;
     return true;
  }
}
```

Legal, mais informativo!

Mas continua sendo não checada...

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

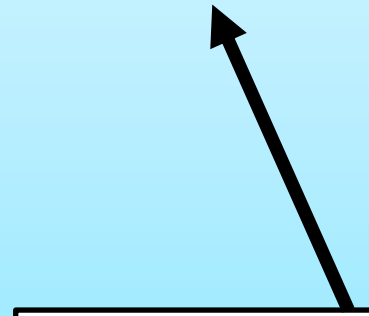
Criando nossas próprias exceções

```
public class SaldoInsuficienteException extends Exception {
  public SaldoInsuficienteException(String message) {
     super(message);
  }
}
```

```
class Conta {
  // ...

  boolean sacar(double qtd) throws SaldoInsuficienteException {
     if (saldo < qtd)
        throw new SaldoInsuficienteException("Sem saldo!");

     saldo = saldo - qtd;
     return true;
  }
}
```



Um pouco redundante?

Criando nossas próprias exceções

```
public class SaldoInsuficienteException extends Exception {
  private double saldo;
  private double qtd;

  public SaldoInsuficienteException(double saldo, double qtd) {
    super("Você tentou sacar " + qtd +
          ", porém seu saldo é de " + saldo);
    this.saldo = saldo;
    this.qtd = qtd;
  }

  // Getters?
}
```

Exceções são classes!

```
  boolean sacar(double qtd) throws SaldoInsuficienteException {
    if (saldo < qtd)
      throw new SaldoInsuficienteException(saldo, qtd);
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

E no caso de múltiplas exceções?

```
public class SaldoInsuficienteException extends Exception {
  // ...
}
```

```
public class LimiteEstouradoException extends Exception {
  public static final double LIMITE_MAXIMO = 10_000.00;

  // ...
}
```

```
  boolean sacar(double qtd) throws SaldoInsuficienteException,
                                   LimiteEstouradoException {
    if (saldo < qtd)
      throw new SaldoInsuficienteException(saldo, qtd);

    if (qtd > LimiteEstouradoException.LIMITE_MAXIMO)
      throw new LimiteEstouradoException(qtd);
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A cláusula `finally`

- Usada quando queremos que um trecho de código seja executado independente de haver ou não exceção;
- Colocada após o último tratador;
- O bloco `finally` é sempre executado!
- Todo bloco `try` deve ter um ou mais blocos `catch` ou um bloco `finally`;
- Pode ter ambos, formando uma estrutura conhecida como `try` – `catch` – `finally`.

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

A cláusula `finally`

```
try {
  // Código que pode lançar exceções...
}
catch (ExcecaoA e) {
  // Tratamento da exceção A,
  // ou qualquer subclasse de ExcecaoA.
}
catch (ExcecaoB e) {
  // Tratamento da exceção B.
  // ou qualquer subclasse de ExcecaoB.
}
finally {
  // Código executado ao final.
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Multi-catch (Java 7)

- Se o tratamento de duas exceções diferentes for o mesmo, há repetição de código:

```
// Método para abrir uma conexão com um banco de dados.
Connection conn = null;
try {
  Class.forName(driver);
  conn = DriverManager.getConnection(url, usuario, senha);
}
catch (ClassNotFoundException e) {
  System.out.println("Problemas ao abrir conexão...");
}
catch (SQLException e) {
 System.out.println("Problemas ao abrir conexão...");
}
return conn;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Multi-catch (Java 7)

- Podemos generalizar a exceção para a superclasse comum mais próxima, porém é genérico demais...

```
// Método para abrir uma conexão com um banco de dados.
Connection conn = null;
try {
  Class.forName(driver);
  conn = DriverManager.getConnection(url, usuario, senha);
}
catch (Exception e) {
  System.out.println("Problemas ao abrir conexão...");
}
return conn;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Multi-catch (Java 7)

- A partir do Java 7, resolve-se o problema com um `catch` múltiplo (*multi-catch*):

```
// Método para abrir uma conexão com um banco de dados.
Connection conn = null;
try {
  Class.forName(driver);
  conn = DriverManager.getConnection(url, usuario, senha);
}
catch (ClassNotFoundException | SQLException e) {
  System.out.println("Problemas ao abrir conexão...");
}
return conn;
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções e herança

- As exceções adicionam certa complexidade à herança devido ao mecanismo de construção e à sobrescrita de métodos;
- Construtores e exceções:
  - *Construtores são obrigados a lançar exceções declaradas no construtor da superclasse;*
  - *Construtores podem lançar exceções que não são declaradas no construtor da superclasse.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções e herança

```
// Este código gera erro de compilação:
// Unhandled exception type Exception

class Pai {
  Pai() throws Exception { }
}

class Filho extends Pai {
  Filho() {
    // Chamada implícita à super(),
    // super() lança Exception!
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções e herança

- Regras para sobrescrita:
  - *Não é obrigatório declarar que os métodos da subclasse lançam as exceções declaradas no método da superclasse que foi sobrescrito;*
  - *Métodos da subclasse não podem propagar exceções que não estão declaradas no método que foi sobrescrito;*
  - *A exceção: podem propagar exceções que sejam subclasses de uma das exceções declaradas no método que foi sobrescrito.*

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções e herança

```
// Este código gera erro de compilação:
// Exception Exception is not compatible with throws
// clause in Pai.metodo2()

class Pai {
  void metodo1() throws Exception { }
  void metodo2() throws ClassNotFoundException { }
}

class Filho extends Pai {
  @Override
  void metodo1() { } // OK!
  @Override
  void metodo2() throws Exception {
     throws new IOException();
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exceções e herança

```
public class Teste {
  public static void main(String[] args) {
    try {
      Pai p = new Filho();

      // Este método está declarado como lançando
      // ClassNotFoundException, porém a
      // implementação no filho lança outra exceção!
      p.metodo2();
    }
    catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

UNIVERSIDADE FEDERAL
DO ESPÍRITO SANTO

Exercitar é fundamental

- Apostila FJ-11 da Caelum:
  - *Seção 11.11, página 157 (Conta);*
  - *Seção 11.12, página 160 (memória da JVM).*