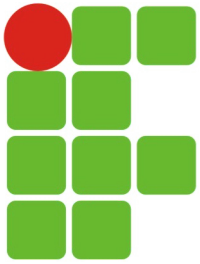
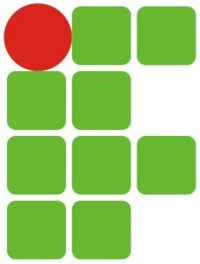


Tratamento de Exceções

Givanaldo Rocha de Souza
http://docente.ifrn.edu.br/givanaldorocha
givanaldo.rocha@ifrn.edu.br

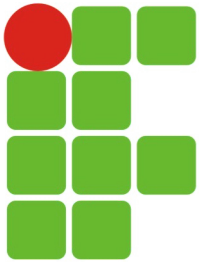


O que é ???

É um evento que interrompe o fluxo normal de processamento de uma classe.

Causa o término inesperado da classe:

- Problemas no hardware, arrays fora de faixa
- Valores de variáveis
- Divisão por zero
- Parâmetros de métodos, falha de Memória
- Erro de entrada e saída (I/O)
- Erros da aplicação
- Saldo insuficiente
- Usuário não existe
- Nota inválida



Características

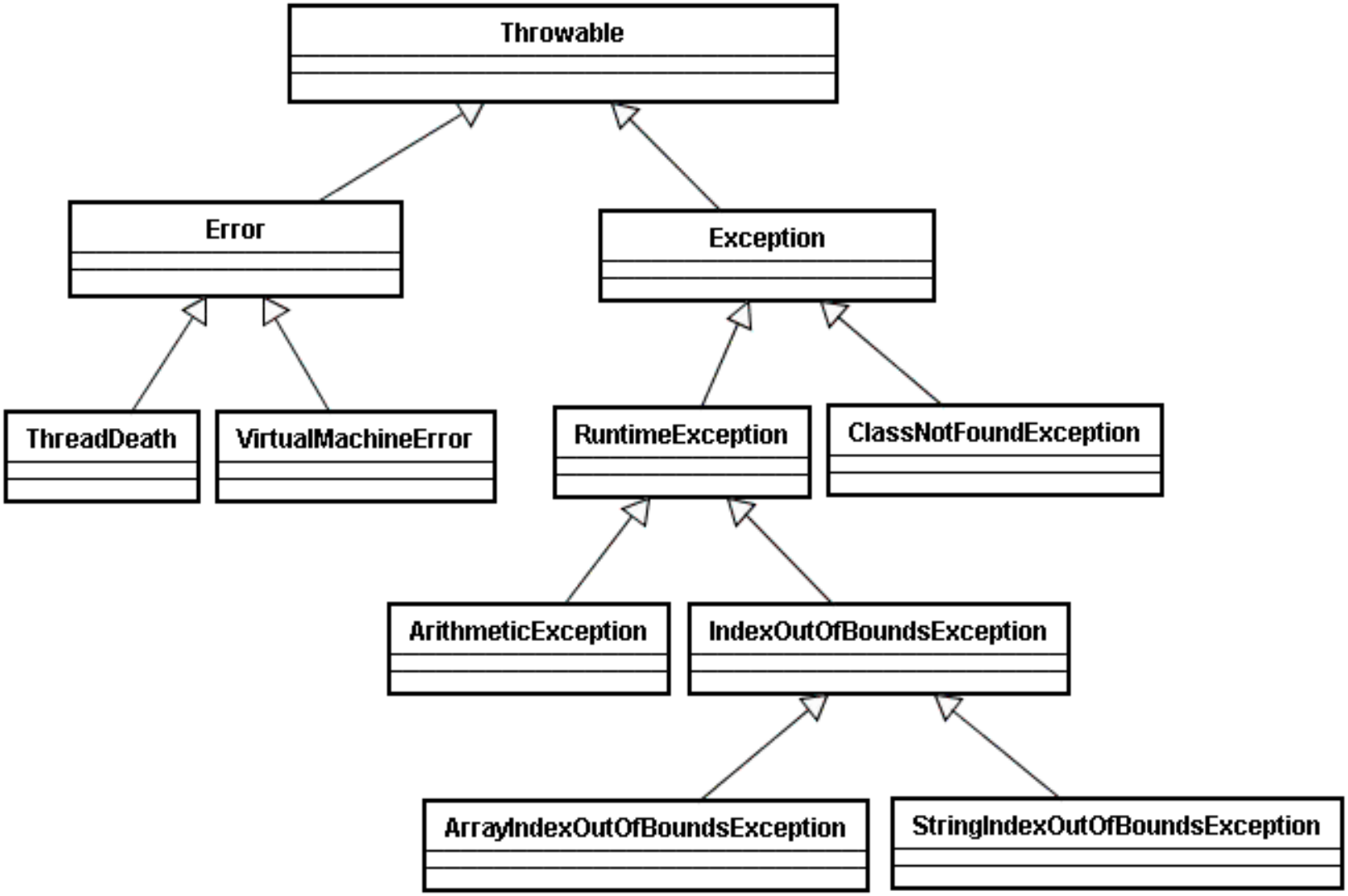
Indica que houve problema na execução de um bloco do programa, mais especificamente em um método.

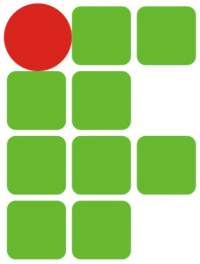
Se não for tratado, o programa pode parar.

O uso correto de exceções torna o programa mais robusto e confiável.

Hierarquia (algumas classes importantes)

http://java.sun.com/docs/books/tutorial/essential/exceptions





Funcionamento

Um método deve informar que exceções ele pode disparar (**throw**): cláusula throws na definição do método

Um bloco que tenta (**try**) chamar um método que pode disparar uma exceção deve tratá-la: chamada normal de um método, mas que deve estar em um bloco **try {...} catch {...}**

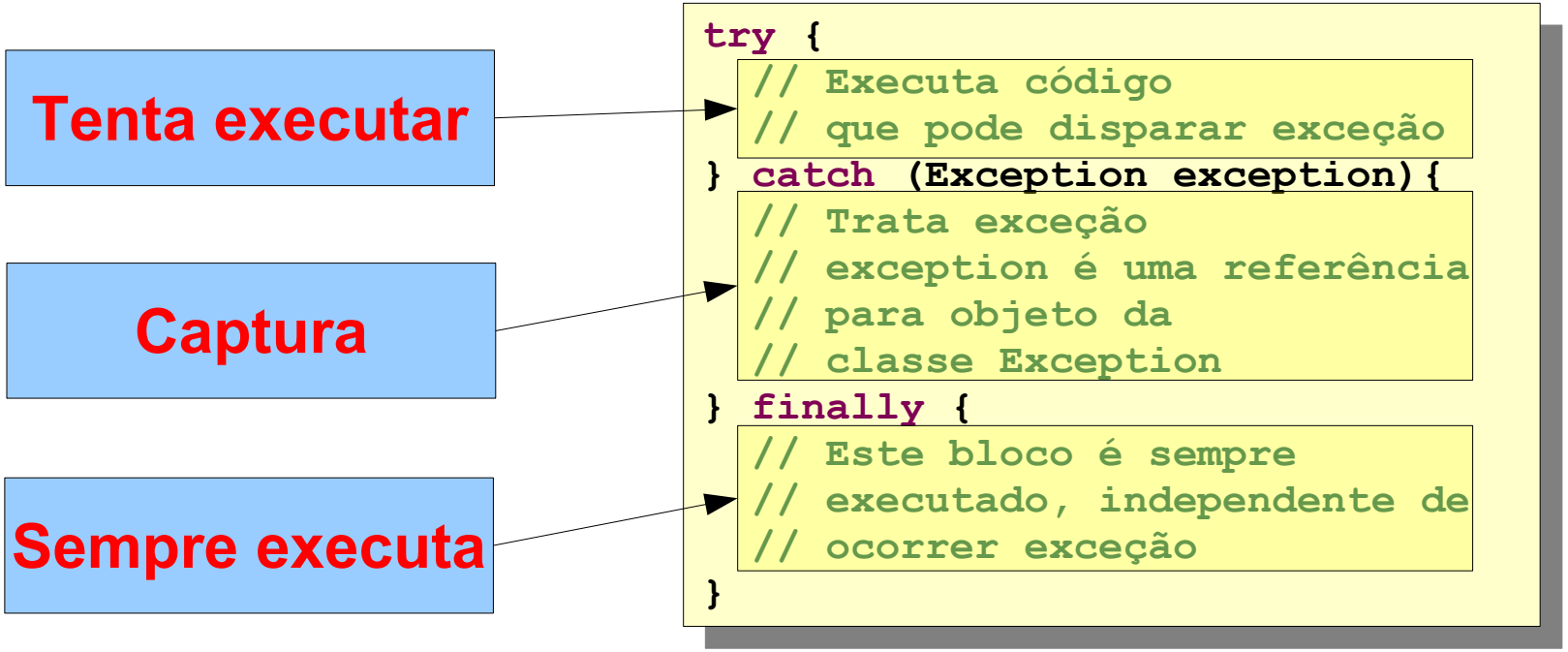
Uma exceção é um objeto que deve ser capturado (**catch**): é nesse bloco que a exceção deve ser tratada.

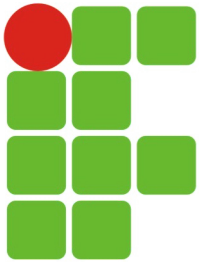
Um trecho de código pode ser executado sempre: bloco **finally**.

Tratando Exceções

Um bloco deve capturar uma exceção para tratá-la

bloco try-catch-finally



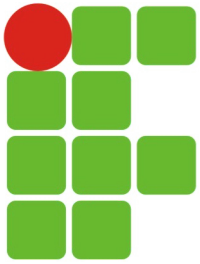


Exemplo

Método **parseInt** pode disparar exceção NumberFormatException

- Se a exceção for disparada, os comandos do bloco **try** não serão mais executados.
- O fluxo de execução muda para a captura (bloco **catch**)
- Ela pode não ser tratada, o programa encerra se for disparada

```
System.out.print("Digite um número");
String numero = sc.nextLine();
int x;
try{
  x = Integer.parseInt(numero);
  System.out.println("Numero digitado é válido");
} catch (NumberFormatException exception){
  System.out.println("Digite um número válido");
}
```

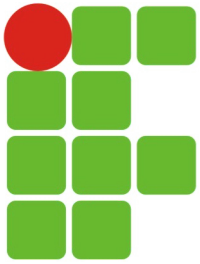


Catch

Pode haver mais de um bloco catch

- Cada bloco trata um tipo específico de exceção
- O bloco **try** contém métodos que podem disparar todas as exceções
- Um método pode disparar mais de uma exceção
- As classes de exceção seguem a ordem da mais especializada para a mais genérica.

```
try {
// bloco de comandos
} catch (Excecao1 ex1) {
// Trata exceção1
} catch (Excecao2 ex2) {
// Trata exceção2
} catch (Excecao3 ex3){
// Trata exceção3
}
```



Exemplo

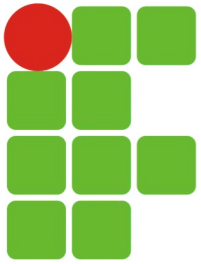
Considere o método lerArquivo

- Recebe nome do arquivo e retorna bytes do arquivo

O que pode acontecer?

- Se o arquivo não for aberto?
- Se o tamanho não puder ser determinado?
- Se ocorrer erro ao alocar memória?
- Se a leitura do arquivo falhar?
- Se houver problemas ao fechar o arquivo?

```
byte[] lerArquivo(String nome){
  byte []dados;
  // Abrir arquivo
  // Determinar tamanho
  // Alocar memória (array)
  // Ler dados do arquivo
  // Fechar arquivo
  return dados;
}
```



Exemplo (cont.)

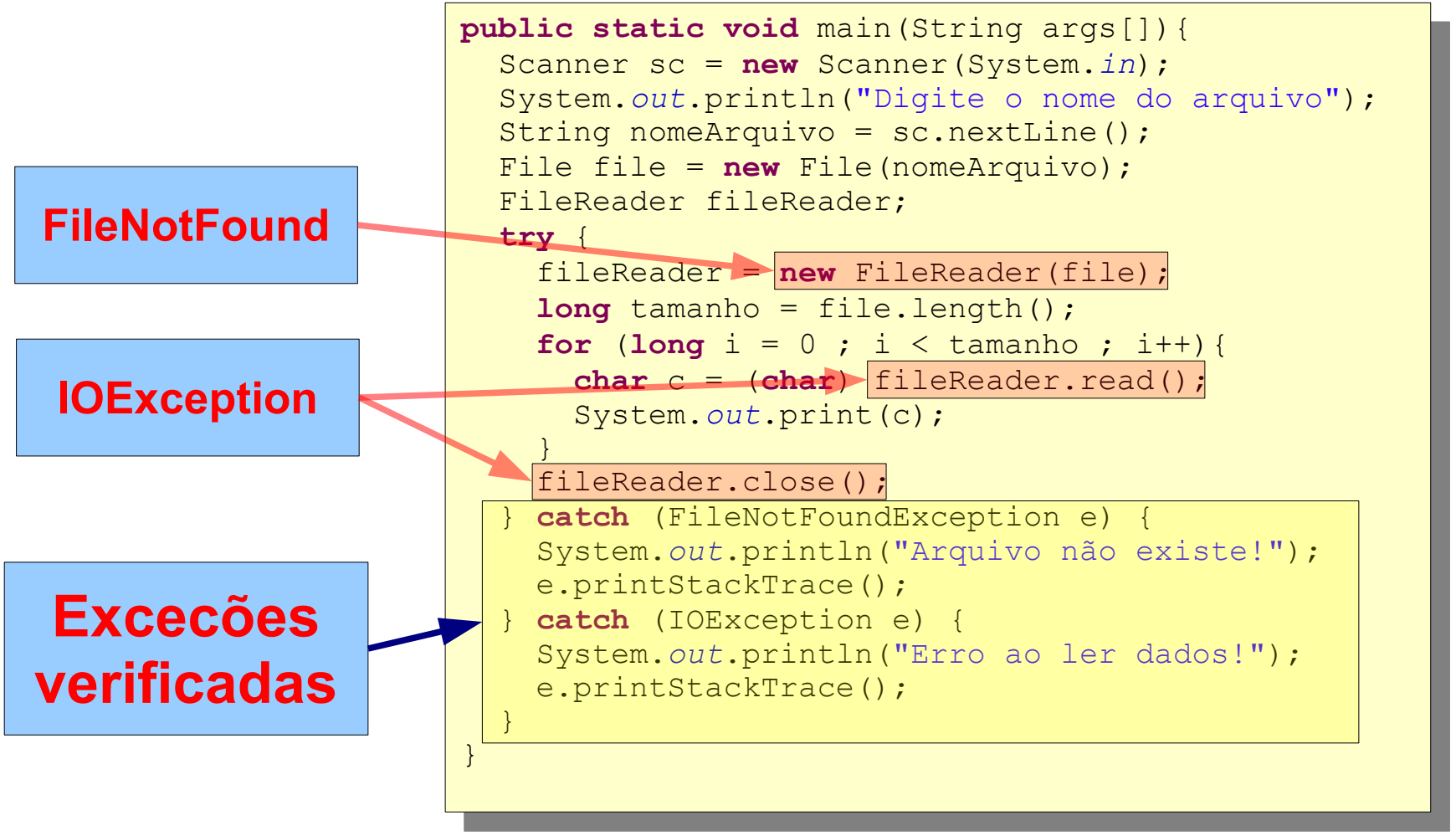
Código sem exceções fica confuso. Não se sabe o que é do trecho normal e tratamento de erros, ao contrário do código com tratamento.

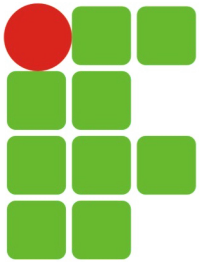
```
byte[] lerArquivo(String nome) {
  byte[] dados;
  try {
    // Abrir arquivo
    // Determinar tamanho
    // Alocar memória (array)
    // Ler dados do arquivo
    // Fechar arquivo
  } catch (FileOpenException foe) {
    // Trata aqui
  } catch (FileSizeException fse) {
    // Trata aqui
  } catch (MemoryAlocationException mae) {
    // Trata aqui
  } catch (FileReadException fre) {
    // Trata aqui
  } catch (FileCloseException fce) {
    // Trata aqui
  }
  return dados;
}
```

```
byte[] lerArquivo(String nome){
  byte []dados;
  File file = new File(nome);
  if (arquivoAberto){
    int tamanho = (int)file.length();
    if (tamanhoOK){
      dados = new byte[tamanho];
      if (alocouMemoria){
        dados = pegaBytes();
        if (!leuDados){
          erro = -1;
        }
      } else{
        erro=-2;
      }
    } else {
      erro = -3;
    }

  } else {
    erro = -4;
  }
  return dados;
}
```

Exemplo prático

Ler dados de um arquivo texto mostrando os caracteres na tela





Finally

Se ocorrer exceção:

Arquivo não é fechado: usa **finally**

Sempre será executado, ocorrendo ou não exceção

```
try {
  fileReader = new FileReader(file);
  long tamanho = file.length();
  for (long i = 0 ; i < tamanho ; i++){
    char c = (char) fileReader.read();
    System.out.print(c);
  }
} catch (FileNotFoundException e) {
  System.out.println("Arquivo não existe!");
} catch (IOException e) {
  System.out.println("Erro ao ler arquivo");
} finally {
  try {
    fileReader.close();
  } catch (IOException e) {
    //Faz nada
  }
}
```

Repassando uma exceção

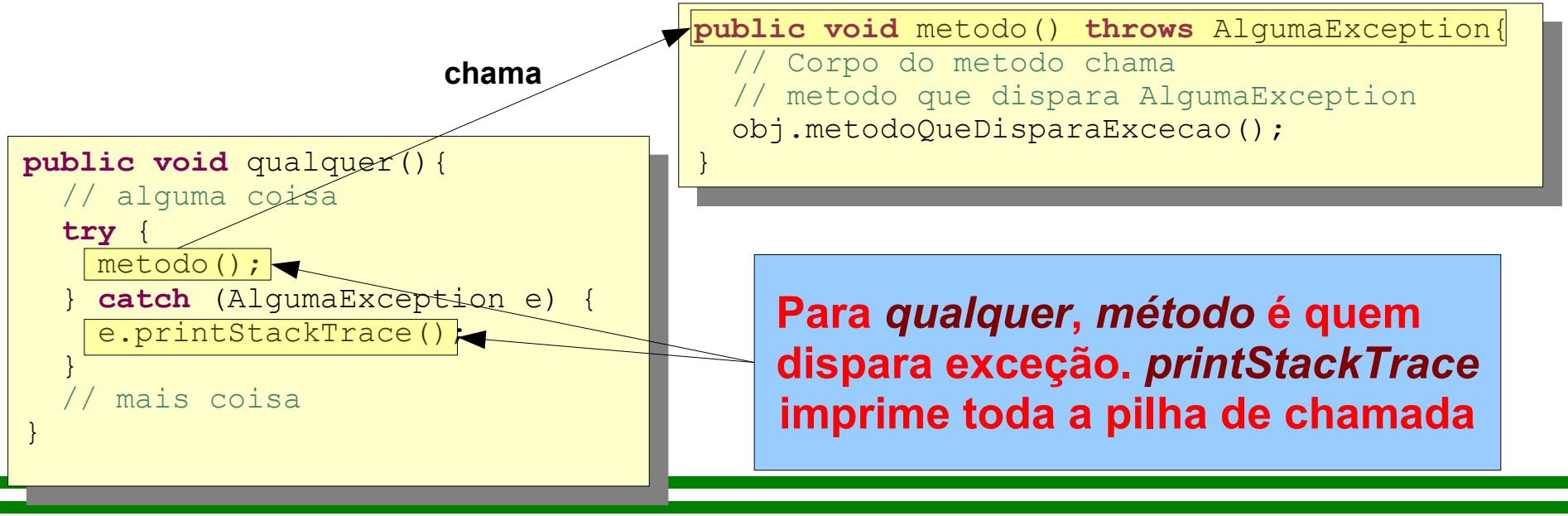
Um método pode **repassar** uma exceção

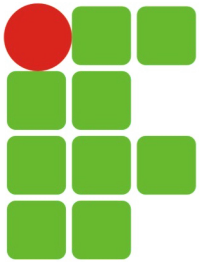
Ele chama um método que dispara uma exceção, mas não quer tratar.

Ele pode repassar a exceção

Basta colocar a cláusula **throws** na assinatura do método

Para quem chama, é o método que dispara a exceção

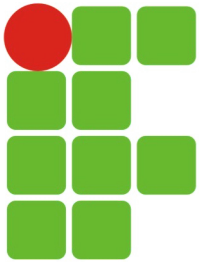




Definindo exceções

Defina uma classe que herde de Exception

```
public class ContatoNaoEncontradoException extends Exception {

  public ContatoNaoEncontradoException()
  {
    super("Contato não encontrado");
  }

}
```

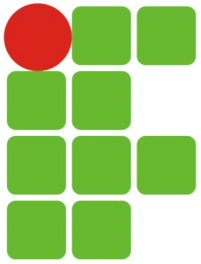


Definindo exceções (1)

No método que dispara a exceção:

- Coloque a cláusula **throws**
- Crie o objeto da classe de exceção
- Dispare (throw) a exceção
  - Onde a exceção será disparada depende de cada método

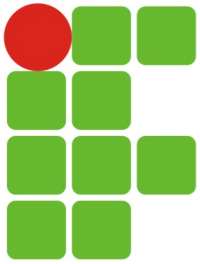
```
public Contato buscar(String nome)throws ContatoNaoEncontradoException{
  // Laco para procurar contato pelo nome
  for (int i = 0 ; i < quantidade ; i++)
    if (contatos[i].nome().equals(nome))
      return contatos[i];

  // Se sair do laço e não tiver retornado,
  // o contato não está cadastrado.
  // Deve disparar exceção
  throw new ContatoNaoEncontradoException();
}
```



Definindo exceções (2)

Exceção é tratada na interface com usuário

```
private void buscarContato() {
   System.out.print("Digite o nome: ");
   String nome = sc.nextLine();
   try {
      Contato contato = agenda.buscar(nome);
      System.out.println(contato);
   } catch (ContatoNaoEncontradoException e) {
      System.out.println("ERRO!!!!");
      System.out.println(e.getMessage());
      System.out.println("\nDigite [ENTER] para continuar....");
      sc.nextLine();
   }
}
```



Exercício

Identifique as Exceções do pacote java.lang

Faça uma calculadora com as quatro operações básicas (somar, subtrair, multiplicar e dividir), verificando se a entrada de números é válida ou não. No caso da divisão, verificar a divisão de um número por zero e dispara a exceção DivisaoPorZeroException. Use try...catch...finally

No programa do Banco, implementar a exceção:

SaldoInsuficienteException

Esta exceção deverá ser disparada pelo método sacar