



Programação 00

8ª aula

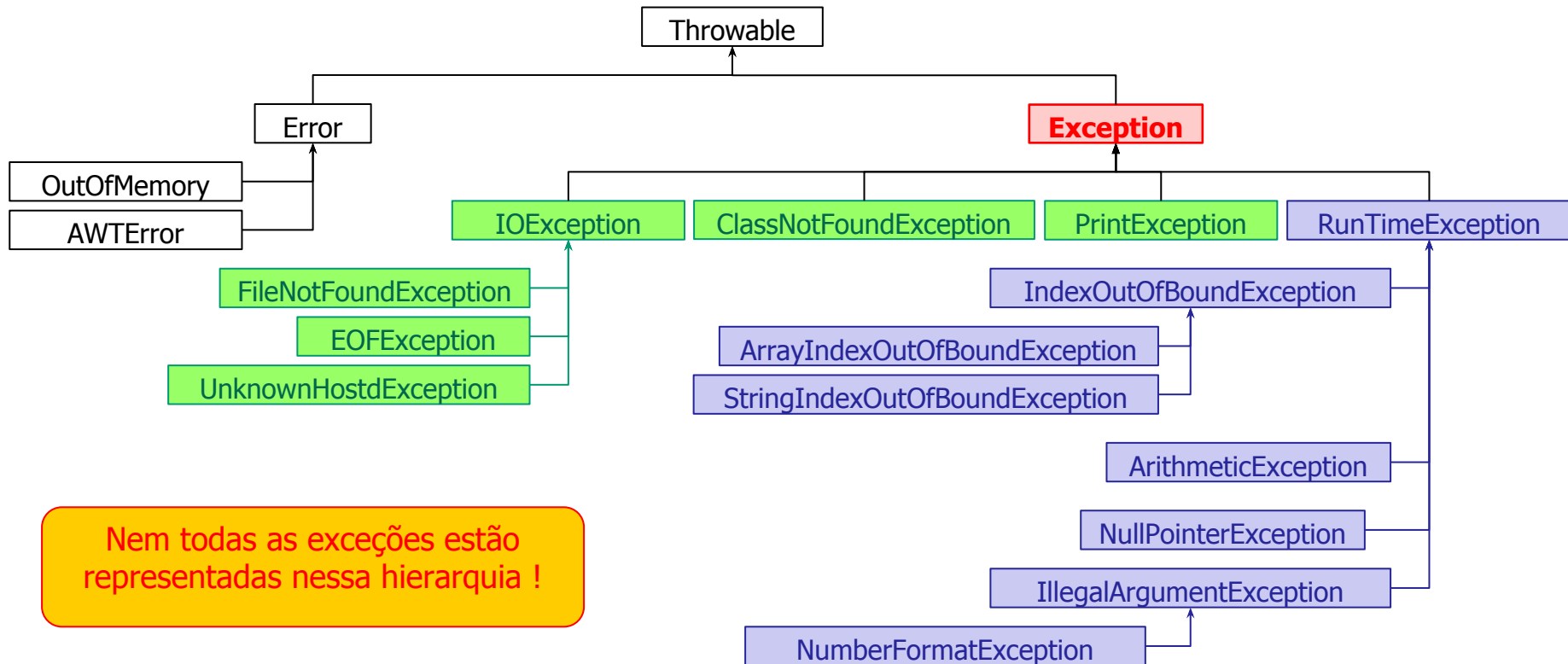
Prof. Douglas Oliveira

douglas.oliveira@prof.infnet.edu.br

- Como tratar, em Java, as seguintes situações?
 - ✓ Divisão por zero
 - ✓ Erro na conversão de tipos (por exemplo, converter uma string que só contém letras em um número inteiro)
 - ✓ Erro na abertura de um arquivo
 - ✓ Erro na transmissão de um arquivo via rede
 - ✓ Erro de impressão
 - ✓ Acesso a um vetor com índice inválido
 - ✓ etc...
- Todas essas situações em Java são chamadas de **exceções** e existe um mecanismo específico para tratá-las chamado de **tratamento de exceções**.

Hierarquia de Exceções

- As exceções em Java estão organizadas em uma **hierarquia de classes**.



Hierarquia de Exceções

- Um **Error** ocorre devido a problemas no SO, na JVM ou no hardware.
 - ✓ Nesse caso, o melhor a fazer é deixar a JVM encerrar o programa.
- A classe **Exception** é a classe **mãe** de todas as exceções que nossos programas podem tratar.

Hierarquia de Exceções

- Ela está sub-dividida em **dois** ramos:
 - a) **RuntimeException:**
 - Ocorrem devido a um erro de programação: divisão por zero, índice inválido do vetor, acesso a objeto nulo, etc.
 - Também são chamadas de exceções **não verificadas** (*unchecked*).
 - b) **Demais exceções:**
 - Ocorrem devido a um erro no programa causado por fatores externos: erro na abertura de um arquivo, erro na impressão, etc.
 - Também são chamadas de exceções **verificadas** (*checked*).

- Quando executamos o programa abaixo:

```
1 public class DividePorZero {  
2     public static void main(String args[]) {  
3         System.out.println(3/0);  
4         System.out.println("imprime");  
5     }  
6 }
```

- Observamos a seguinte mensagem:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at DividePorZero.main(DividePorZero.java:3)
```

- Como o nosso programa não está tratando essa exceção (divisão por zero), o **tratador padrão do Java** executa as seguintes tarefas:
 1. Imprime o nome da exceção e a mensagem de erro;
 2. Imprime a pilha de execução (sequência de chamadas dos métodos);
 3. Termina o programa.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at DividePorZero.main(DividePorZero.java:3)
```

Esse é o nome da
exceção que ocorreu.

Essa é a pilha de execução, ou seja, a
sequência de chamadas de métodos.

Essa é a descrição da
exceção, ou seja, a
mensagem de erro.

Tratamento de Exceções

- O tratamento de exceções é um mecanismo que permite que o programa define **como as situações inesperadas serão tratadas**.
- Existem três comandos relacionados ao tratamento de exceções:
 1. Blocos **try...catch...finally**
 2. Comando **throws**
 3. Comando **throw**

Tratamento de Exceções

1. Bloco try...catch

```
try {
```

Se ocorrer uma exceção nesse bloco, então a execução é automaticamente desviada para o bloco **catch**.

```
// Código que pode gerar uma exceção
```

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

```
// Código que será executado quando ocorrer a exceção
```

```
}
```

No **catch** devemos definir a exceção a ser tratada. Quando definimos uma exceção estamos tratando também todas as suas subclasses.

A variável e referencia a exceção que ocorreu. Com ela é possível acessar as informações sobre essa exceção.

- A classe `Exception` implementa alguns métodos que podemos executar em nossos programas:
- Dentre eles está o método:
 - ✓ `getMessage()`: retorna a mensagem de erro armazenada na exceção. Nem toda exceção possui mensagem de erro (nesse caso o método retorna `null`).

□ Exemplo 1: tratando uma única exceção.

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
    System.out.printf("%d / %d = %d\n", a, b, c);  
} catch (Exception e) {  
    System.out.printf("Erro: %s\n", e.getMessage());  
}
```

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exemplo 1: tratando uma única exceção.

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
    System.out.printf("%d / %d = %d\n", a, b, c);  
} catch (Exception e) {  
    System.out.printf("Erro: %s\n", e.getMessage());  
}
```

Exceção 1

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exemplo 1: tratando uma única exceção.

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
    System.out.printf("%d / %d = %d\n", a, b, c);  
  
} catch (Exception e) {  
    System.out.printf("Erro: %s\n", e.getMessage());  
}
```

Exceção 2

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exemplo 1: tratando uma única exceção.

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
  
    System.out.printf("%d / %d = %d\n", a, b, c);  
} catch (Exception e) {  
    System.out.printf("Erro: %s\n", e.getMessage());  
}
```

Exceção 3

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Tratamento de Exceções

- É possível tratar **várias exceções** associando **vários *catch*'s** ao mesmo *try*.
- Nesse caso, a **ordem** dos tratadores é importante: eles devem estar ordenados das subclasses para a superclasse.

```
try {  
    // Código a ser tratado  
} catch (ArithmeticException e3) {  
    System.out.printf("Erro de aritmetica: %s\n", e3.getMessage());  
} catch (IOException e2) {  
    System.out.printf("Erro de E/S: %s\n", e2.getMessage());  
} catch (Exception e1) {  
    System.out.printf("Erro desconhecido: %s\n", e1.getMessage());  
}
```

Primeiro o catch da exceção mais específica.

Por último o catch da exceção mais geral.

Tratamento de Exceções

- Cada *catch* indica o *tipo* de exceção que vai tratar:

```
try {  
    // Código a ser tratado  
  
} catch (ArithmeticException e3) {  
    System.out.printf("Erro de aritmetica: %s\n", e3.getMessage());  
}  
catch (IOException e2) {  
    System.out.printf("Erro de E/S: %s\n", e2.getMessage());  
}  
catch (Exception e1) {  
    System.out.printf("Erro desconhecido: %s\n", e1.getMessage());  
}
```

Se ocorrer um erro de aritmética,
esse código será executado.

Para qualquer outro erro, esse código
será executado.

Se ocorrer um erro de E/S
ou de qualquer de suas
subclasses, esse código será
executado.

Exemplo 2: tratando **várias** exceções.

```
int a, b, c;  
Scanner t = new Scanner(System.in);  
try {  
    a = t.nextInt();  
    b = t.nextInt();  
    c = a / b;  
    System.out.printf("%d / %d = %d\n", a, b, c);  
} catch(InputMismatchException e) {  
    System.out.printf("Entrada inválida\n");  
} catch(ArithmeticException e) {  
    System.out.printf("Erro: %s\n", e.getMessage());  
}
```

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exemplo 2: tratando várias exceções.

```
int a, b, c;
Scanner t = new Scanner(System.in);
try {
    a = t.nextInt();
    b = t.nextInt();
    c = a / b;
    System.out.printf("%d / %d = %d\n", a, b, c);

} catch (InputMismatchException e) {
    System.out.printf("Entrada inválida\n");
} catch (ArithmeticException e) {
    System.out.printf("Erro: %s\n", e.getMessage());
}
```

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exceção 1

Exemplo 2: tratando **várias** exceções.

```
int a, b, c;
Scanner t = new Scanner(System.in);
try {
    a = t.nextInt();
    b = t.nextInt();
    c = a / b;
    System.out.printf("%d / %d = %d\n", a, b, c);

} catch (InputMismatchException e) {
    System.out.printf("Entrada inválida\n");
} catch (ArithmeticException e) {
    System.out.printf("Erro: %s\n", e.getMessage());
}
```

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

Exceção 2



Tratamento de Exceções

- Exemplo 2: tratando **várias** exceções.

```
int a, b, c;
Scanner t = new Scanner(System.in);
```

```
try {
    a = t.nextInt();
    b = t.nextInt();
}
```

Exceção 3

```
c = a / b;
```

```
System.out.printf("%d / %d = %d\n", a, b, c);
```

```
} catch (InputMismatchException e) {  
    System.out.printf("Entrada inválida\n");
```

```

} catch (ArithmeticException e) {
    System.out.printf("Erro: %s\n", e.getMessage());
}

```

Que exceções podem acontecer ?

1. Usuário digitar um número inválido para **a**
2. Usuário digitar um número inválido para **b**
3. Usuário digitar ZERO para **b**

- O que acontece quando ocorre uma exceção ?
 - 1) O método cria um objeto do tipo **Exception** e o envia para a JVM:
 - ✓ Esse processo é chamado de "disparar uma exceção" (*throw an exception*)
 - ✓ O objeto **Exception** criado contém **todas** as informações sobre o erro: seu tipo, o local onde ocorreu, uma mensagem de descrição, a pilha de chamadas, etc.
 - 2) A JVM procura um bloco **try..catch** para tratar a exceção **no método que gerou a exceção**. Se encontrar, desvia a execução para o **catch**.

- O que acontece quando ocorre uma exceção ?
 - 3) Se não encontrar, procura um bloco `try..catch` para tratar a exceção na `pilha de execução`, ou seja, nos métodos que chamaram o método que gerou a exceção. Se encontrar, desvia a execução para o `primeiro catch` que encontrar.
 - 4) Se não encontrou nenhum tratador na pilha de execução, desvia para o `tratador padrão` da JVM que `interrompe` a execução do programa.

□ Exemplo 1: bloco `try..catch` no método que gerou a exceção.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

Esse tratador vai ser ativado porque *Exception* captura qualquer exceção.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        try {  
            x = 3 / 0;  
        } catch (Exception e) {  
            System.out.println("Erro !");  
        }  
    }  
    public void metodoB() {  
        metodoC();  
    }  
    public void metodoA() {  
        metodoB();  
    }  
}
```

Exemplo 2: bloco `try..catch` no método que gerou a exceção.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

Esse tratador vai ser ativado porque *RuntimeException* é superclasse de *ArithmeticException*.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        try {  
            x = 3 / 0;  
        } catch (EOFException e) {  
            System.out.println("EOF!");  
        } catch (RuntimeException e) {  
            System.out.println("Erro !");  
        }  
    }  
    public void metodoB() {  
        metodoC();  
    }  
    public void metodoA() {  
        metodoB();  
    }  
}
```


❑ Exemplo 3: bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

Como o metodoC que gerou a exceção não define um tratador, a JVM procura o tratador no método chamador, ou seja, o metodoB.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        try {  
            metodoC();  
        } catch (Exception e) {  
            System.out.println("Erro !");  
        }  
    }  
    public void metodoA() {  
        metodoB();  
    }  
}
```

Exemplo 4: bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

Se o metodoB (chamador) também não define um tratador, a JVM continua procurando por toda a pilha de execução.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        metodoC();  
    }  
    public void metodoA() {  
        try {  
            metodoB();  
        } catch (Exception e) {  
            System.out.println("Erro !");  
        }  
    }  
}
```

Exemplo 5: bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

O tratador *`NullPointerException`* não vai ser ativado porque essa classe não é superclasse de *`ArithmeticException`*.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        try {  
            metodoC();  
        } catch (NullPointerException e) {  
            System.out.println("NULO !");  
        }  
    }  
    public void metodoA() {  
        try {  
            metodoB();  
        } catch (Exception e) {  
            System.out.println("Erro !");  
        }  
    }  
}
```

Exemplo 6: bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        try {  
            ex.metodoA();  
        } catch (RuntimeException e) {  
            System.out.println("Erro !");  
        }  
    }  
}
```

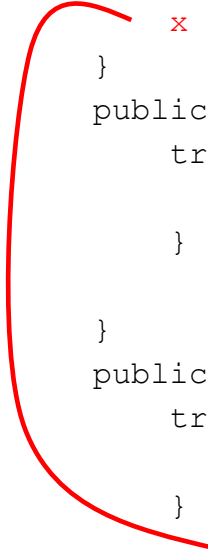
Os tratadores *NullPointerException* e *EOFException* não vão ser ativados porque essas classes não são superclasses de *ArithmeticException*.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        try {  
            metodoC();  
        } catch (NullPointerException e) {  
            System.out.println("NULO !");  
        }  
    }  
    public void metodoA() {  
        try {  
            metodoB();  
        } catch (EOFException e) {  
            System.out.println("EOF !");  
        }  
    }  
}
```

Exemplo 7: bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        try {  
            ex.metodoA();  
        } catch (Exception e) {  
            System.out.println("Erro !");  
        }  
    }  
}
```

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        try {  
            metodoC();  
        } catch (NullPointerException e) {  
            System.out.println("NULO !");  
        }  
    }  
    public void metodoA() {  
        try {  
            metodoB();  
        } catch (Exception e) {  
            System.out.println("Erro!");  
        }  
    }  
}
```



Exemplo 8: sem bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
        ex.metodoA();  
    }  
}
```

O programa vai ser interrompido pela JVM e será apresentada a mensagem de erro.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        x = 3 / 0;  
    }  
    public void metodoB() {  
        try {  
            metodoC();  
        } catch (NullPointerException e) {  
            System.out.println("NULO !");  
        }  
    }  
    public void metodoA() {  
        try {  
            metodoB();  
        } catch (EOFException e) {  
            System.out.println("EOF !");  
        }  
    }  
}
```

Exemplo 9: sem bloco `try..catch` na pilha de execução.

```
public class TesteExcecao {  
    public static void main(String[] args) {  
        Exemplo ex = new Exemplo();  
  
        ex.metodoA();  
    }  
}
```

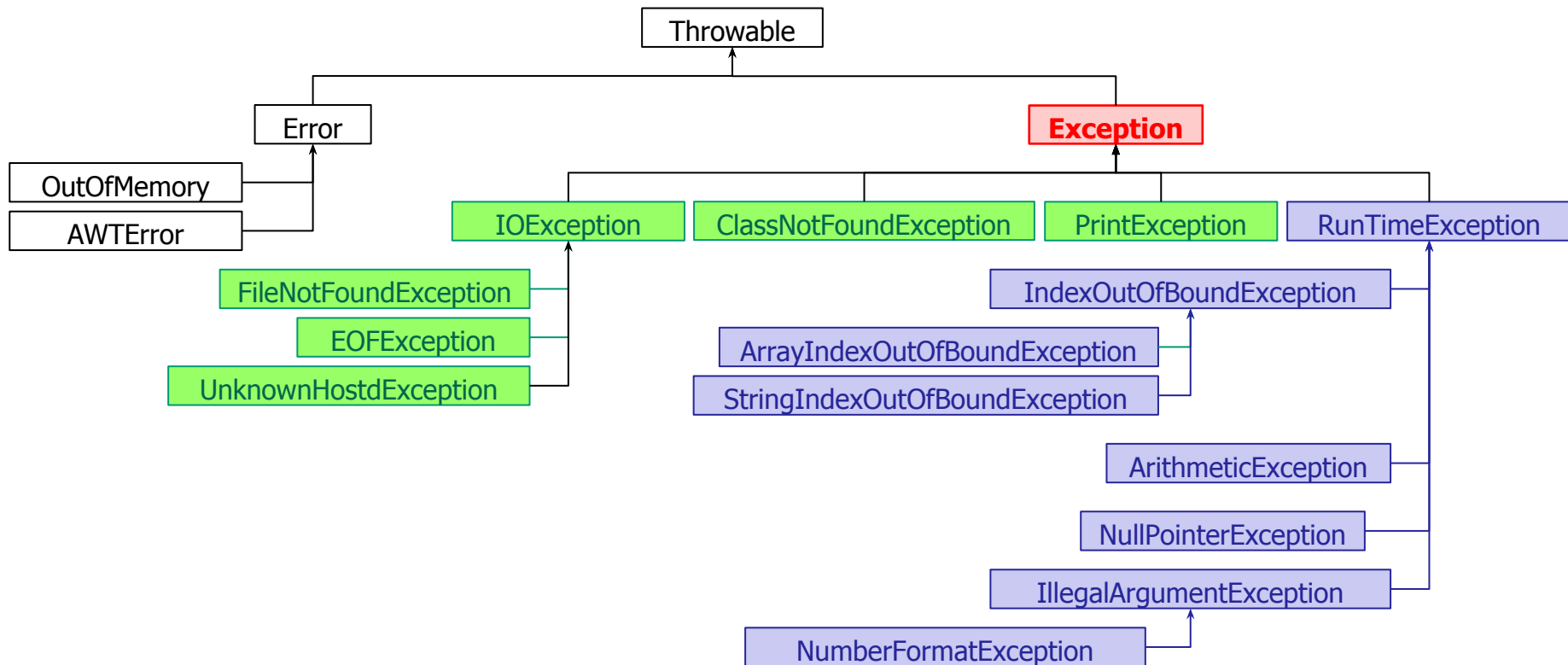
O programa vai ser interrompido pela JVM e será apresentada a mensagem de erro.

```
public class Exemplo {  
    public void metodoC() {  
        int x;  
        // Vai gerar ArithmeticException  
        try {  
            x = 3 / 0;  
        } catch (EOFException e) {  
            System.out.println("EOF!");  
        }  
    }  
    public void metodoB() {  
        metodoC();  
    }  
    public void metodoA() {  
        metodoB();  
    }  
}
```

- Exercício 39: crie um programa que leia números inteiros positivos e imprima o somatório desses números. O programa deve parar de ler quando o usuário digitar um número menor ou igual a zero. Faça o tratamento de exceção para o caso do usuário não digitar um número quando solicitado.

Hierarquia de Exceções

- As exceções do Java são classificadas como **checked** ou **unchecked**.



- Para as exceções *checked* (verificadas), o Java nos obriga a:
 - 1) Tratar a exceções no método onde ela pode ocorrer.
 - ✓ Nesse caso, implementamos o bloco *try...catch* visto anteriormente.
 - OU
 - 2) Avisar que estamos cientes de que aquela exceção pode ocorrer, mas *não desejamos tratá-la*.
 - ✓ Nesse caso, usamos o comando *throws*.

2. Comando `throws`

```
public class ImprimeArquivo {  
    public static void main(String[] args)  
    {  
        FileReader fr = new FileReader("arquivo.txt");  
        BufferedReader f = new BufferedReader(fr);  
        String linha;  
        linha = f.readLine();  
        while (linha != null) {  
            System.out.println(linha);  
            linha = f.readLine();  
        }  
        f.close();  
    }  
}
```

Esse trecho de programa lê e imprime um arquivo texto. Esse código pode gerar exceções *checked* do tipo **FileNotFoundException** ou **IOException**. **O Java não compila esse código !**

2. Comando `throws`

```
public class ImprimeArquivo {  
    public static void main(String[] args) throws  
                                FileNotFoundException, IOException{  
        FileReader fr = new FileReader("arquivo.txt");  
        BufferedReader f = new BufferedReader(fr);  
        String linha;  
        linha = f.readLine();  
        while (linha != null) {  
            System.out.println(linha);  
            linha = f.readLine();  
        }  
        f.close();  
    }  
}
```

Para avisar ao compilador que **não desejamos tratar esses erros**, temos que usar o comando **throws**. Assim, a compilação é realizada.

- ❑ Como tratar situações de erro que são específicas dos nossos programas ?
- ❑ Exemplo: que valores seriam inválidos na construção dos objetos abaixo?

```
public Retangulo(int x, int y, int largura, int altura)
public Circulo(int x, int y, int raio)
public Triangulo(int x1, int y1,
                  int x2, int y2,
                  int x3, int y3)
```

Tratamento de Exceções

- ❑ Em muitos casos precisamos **sinalizar** a ocorrência de um erro.
- ❑ Exemplo: que valores seriam inválidos na construção dos objetos abaixo?

```
public Retangulo(int x, int y, int largura, int altura)
```

Largura e altura devem ser maiores que zero.

```
public Circulo(int x, int y, int raio)
```

Raio deve ser maior que zero.

```
public Triangulo(int x1, int y1,  
                 int x2, int y2,  
                 int x3, int y3)
```

Os pontos **(x1, y1)**, **(x2, y2)** e **(x3, y3)** devem formar um triângulo.

- Como podemos tratar a situação onde **largura** ou **altura** é menor ou igual a zero? Resposta: **criando** e **disparando** uma exceção!

```
public class Retangulo {  
    private int x, y, largura, altura;  
    public Retangulo(int x, int y, int largura, int altura)  
    {  
        this.x = x;  
        this.y = y;  
        this.largura = largura;  
        this.altura = altura;  
    }  
}
```

3. Comando `throw`

- É usado para `disparar` ou `lançar` uma exceção que `nós mesmos criamos!`
- Como?
 1. Criamos um objeto da classe `Exception` ou de uma de suas subclasses com o operador `new` e o construtor:
 - ✓ `Exception(String mensagemErro)`: construtor que permite criar uma exceção e armazenar nesse objeto uma mensagem de erro.
 2. Disparamos a exceção com o comando `throw`;
 3. Se necessário, declaramos que o método irá disparar a exceção com o comando `throws`.

- Exemplo: criação do objeto **Exception** e disparo da exceção (note que também é necessário o uso do comando **throws**).

```
public class Retangulo {  
    private int x, y, largura, altura;  
    public Retangulo(int x, int y, int largura, int altura)    throws Exception  
    {  
        if (largura <= 0 || altura <= 0) {  
            Exception ex = new Exception("Retângulo deve ter largura e  
                altura maior que zero");  
            throw ex;  
        }  
        this.x = x;  
        this.y = y;  
        this.largura = largura;  
        this.altura = altura;  
    }  
}
```

Se **largura** ou **altura** forem menores ou iguais a zero a execução do construtor será interrompida e a exceção será disparada.

- No método onde o Retangulo é criado podemos tratar essa exceção com **try...catch**:

```
public class EditorGrafico {  
    public static void main(String[] args) {  
        Retangulo r;  
        try {  
            r = new Retangulo(0,0, -10, 20);  
            r.desenhar();  
        } catch(Exception e) {  
            System.out.printf("Erro: %s\n", e.getMessage());  
        }  
    }  
}
```

Em caso de erro na criação do Retângulo, imprime a mensagem de erro e encerra o programa.

- ❑ Para indicar o uso de parâmetros inválidos ou ilegais é possível usar a exceção `IllegalArgumentException` ao invés de `Exception`.
- ❑ Nesse caso, **não** é necessário usar o comando `throws`.

```
public class Retangulo {  
    private int x, y, largura, altura;  
    public Retangulo(int x, int y, int largura, int altura)  
    {  
        if (largura <= 0 || altura <= 0)  
            throw new IllegalArgumentException("Retângulo deve ter largura  
                                                e altura maior que zero");  
  
        this.x = x;  
        this.y = y;  
        this.largura = largura;  
        this.altura = altura;  
    }  
}
```

- Exercício 40: altere o exercício da classe OperacaoMatematica para:
 - a) Gere uma exceção caso seja feita uma divisão por zero ou raiz de número negativo.
 - b) Trate a exceção gerada pelo método executar().
 - c) Implemente um tratador para a entrada dos operandos, que emite uma mensagem de erro caso o usuário digite um número inválido.
- Importante: em Java, a divisão de números reais (float ou double) por zero NÃO gera uma exceção! Operações ilegais com números reais geram valores **indefinidos** chamados **NaN** (Not a Number) ou **Infinity**.

Tratamento de Exceções

1. Bloco try...catch..finally

```
try {
```

```
    // Código a ser tratado
```

```
}
```

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

```
    System.out.println("Erro: %s\n", e.getMessage());
```

```
}
```

```
finally {
```

```
    // Esse código será sempre executado, independente
```

```
    // se houve exceção ou não
```

```
}
```

Se ocorrer uma exceção nesse bloco, então a execução é automaticamente desviada para o bloco **catch**.

A variável **e** referencia a exceção que ocorreu. Com ela é possível acessar informações sobre essa exceção.

finally não é obrigatório.
Deve ser usado para instruções de "limpeza"

Tratamento de Exceções

- ❑ A classe `Exception` implementa alguns métodos que podemos executar em nossos programas:
- ❑ Dentre eles está o método:
 - ✓ `Exception(String mensagemErro)`: construtor que permite criar uma exceção e armazenar nesse objeto uma mensagem de erro.
 - ✓ `getMessage()`: retorna a mensagem de erro armazenada na exceção. Nem toda exceção possui mensagem de erro (nesse caso o método retorna `null`).
 - ✓ `printStackTrace()`: imprime a pilha de execução no mesmo formato da JVM.