Programação para Web I
Tratamento de Exceções

Profa. Silvia Bertagnolli

Exceção

Uma exceção significa “condição excepcional” e é uma ocorrência que altera o fluxo normal da execução do programa.

Exceções: problemas mais comuns

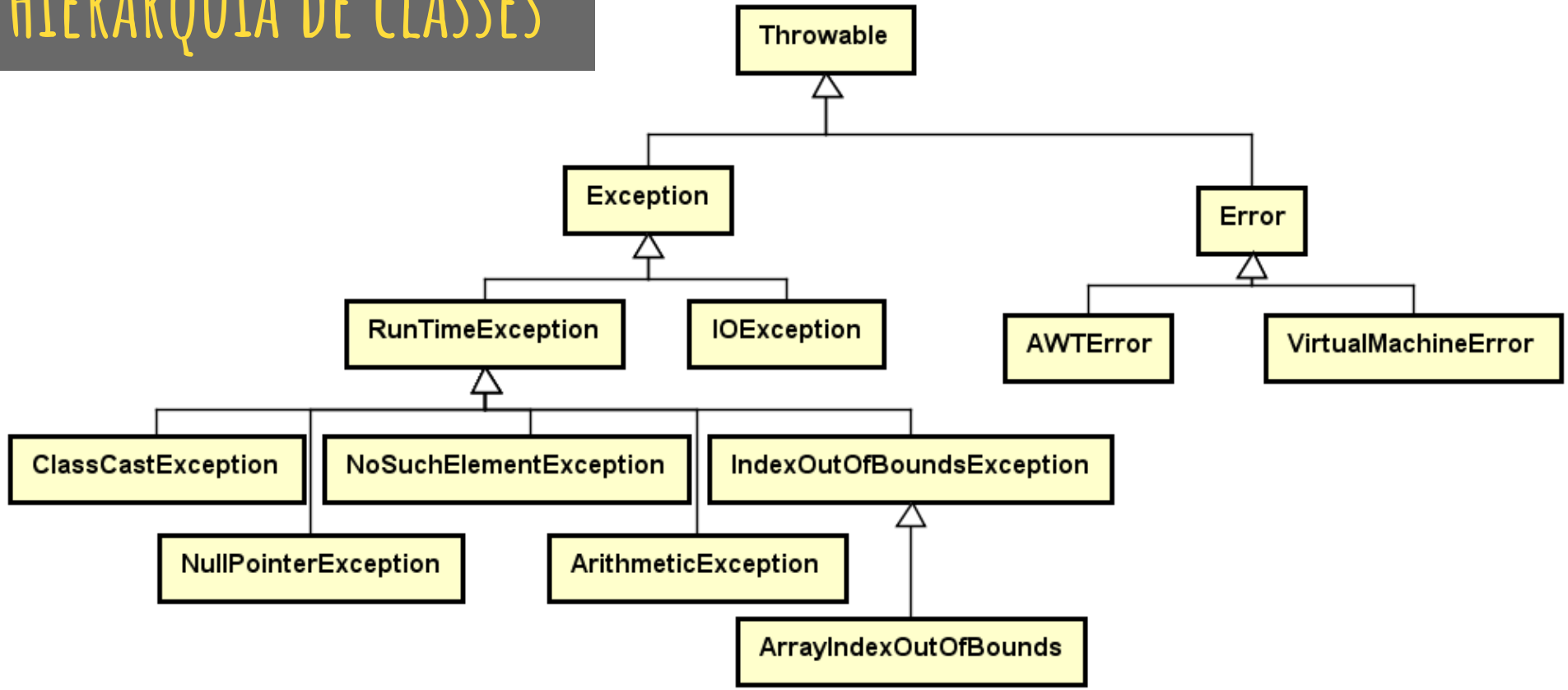
Os problemas mais comuns são:

- falha na aquisição de um recurso (new, open..)
- tentativa de fazer consulta em um banco de dados que está indisponível
- outras condições inválidas (manipular objetos nulos, lista vazia, overflow..)

Algumas exceções:

- NullPointerException
- ArrayIndexOutOfBounds
- FileNotFoundException

Hierarquia de classes



Tipos de Exceções: Error

Error = classe base que descreve os erros não esperados ou erros que não devem ser tratados em circunstâncias normais

Indica um problema grave ocorrido em tempo de execução, tal como: VirtualMachineError, OutOfMemoryError

Erros são de difícil recuperação, se não for impossível

Este tipo de problema é raro de acontecer e não pode ser tratado pelo programador

Tipos de Exceções: Exception

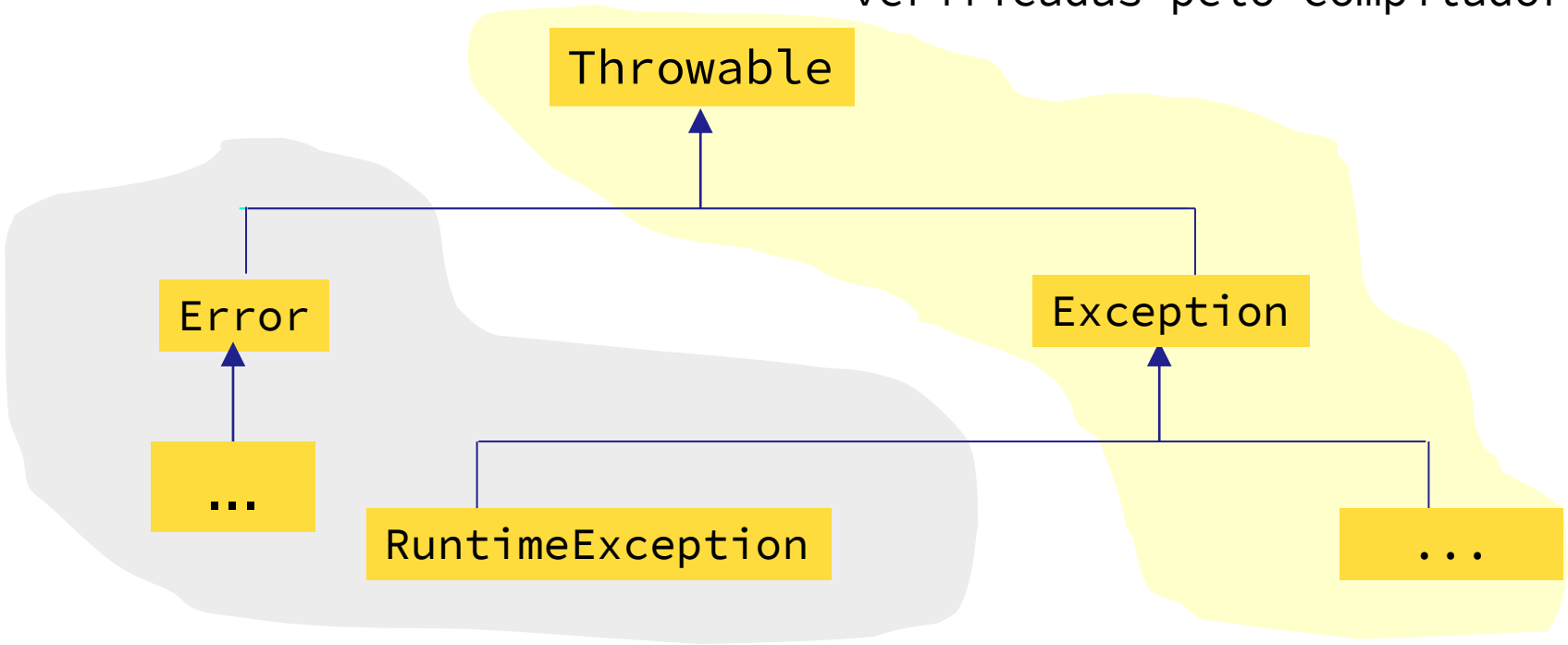
Exception = classe base que descreve uma condição anormal que deve ser tratada pelo programa

Durante a compilação, as exceções são divididas em dois grupos (i) exceções verificadas, e (ii) exceções não verificadas

Exceções verificadas pelo compilador devem ser tratadas, pois, caso contrário, a classe não é compilada

Hierarquia de Classes

verificadas pelo compilador



Não são verificadas pelo compilador

Tipos de Exceções: Exception

Exceções não verificadas pelo compilador (subclasses de RuntimeException) - exceções que podem ocorrem em tempo de execução

Exemplos: ArithmeticException, IndexOutOfBoundsException, entre outras

Tipos de Exceções: Exception

Exceções verificadas pelo compilador - exceções cuja ocorrência é sinalizada em tempo de compilação

Exemplos: IOException, SQLException, entre outras

Instruções para T. E.

Instruções para Tratar/Manipular Exceções

try - identifica um bloco de comandos que **pode** disparar uma exceção

catch - **captura** as exceções e implementa os tratadores de exceções

finally - usado para códigos de liberação de recursos, código que **sempre** executa

throw - usado para **causar** uma exceção

throws - usado para **propagar** uma exceção causada em um método

TRY/CATCH

TRY

Bloco try indica a região do programa que deve ser monitorada pelo sistema de tratamento de exceções

Dicas:

Incluir dentro do bloco try o código que **pode** gerar uma exceção

O bloco try é imediatamente seguido por zero ou mais blocos catch

Capturar exceções: catch

Um comando catch tem as seguintes funções:

- capturar um (determinado) tipo de exceção

- implementar um tratador para aquele tipo de exceção

SEMPRE colocar uma mensagem ou associar alguma informação no bloco catch, porque uma exceção pode ocorrer e você não perceber

Exemplo

```
1  import java.util.Scanner;
2
3  public class Exemplo1 {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          System.out.print("Digite um número: ");
7          int num = scanner.nextInt();
8          System.out.println(num);
9          System.out.println(10/num);
10         scanner.close();
11     }
12 }
```

O que acontece se o usuário informar a letra “a”? E se ele informar o número zero (0)?
Então gera as exceções: InputMismatchException e ArithmeticException

Como tratar a(s) exceção(ões) no código abaixo?

```
1  import java.util.Scanner;
2
3  public class Exemplo1 {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          System.out.print("Digite um número: ");
7          int num = scanner.nextInt();
8          System.out.println(num);
9          System.out.println(10/num);
10         scanner.close();
11     }
12 }
```

Execução das Exceções

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class Exemplo1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("INÍCIO");
        try {
          System.out.print("Digite um número: ");
          int num = scanner.nextInt();
          System.out.println(num);
          System.out.println("Depois da impressão do número");
          System.out.println(10/num);
          System.out.println("Depois da divisão do número");
          scanner.close();
        }catch(InputMismatchException e) {
            System.out.println("Entrada inválida");
        }catch(ArithmeticException e) {
            System.out.println("Divisão inválida");
        }
        System.out.println("FIM");
    }
}
```

Se entrada “a” saída:

```
INÍCIO
Digite um
número: a
Entrada inválida
FIM
```

Linhas executadas

```
1  import java.util.InputMismatchException;
2  import java.util.Scanner;
3
4  public class Exemplo1 {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("INÍCIO");
8          try {
9            System.out.print("Digite um número: ");
10           int num = scanner.nextInt();
11           System.out.println(num);
12           System.out.println("Depois da impressão do número");
13           System.out.println(10/num);
14           System.out.println("Depois da divisão do número");
15           scanner.close();
16         }catch(InputMismatchException e) {
17             System.out.println("Entrada inválida");
18         }catch(ArithmeticException e) {
19             System.out.println("Divisão inválida");
20         }
21         System.out.println("FIM");
22     }
23 }
```

Execução das Exceções

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class Exemplo1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("INÍCIO");
        try {
          System.out.print("Digite um número: ");
          int num = scanner.nextInt();
          System.out.println(num);
          System.out.println("Depois da impressão do número");
          System.out.println(10/num);
          System.out.println("Depois da divisão do número");
          scanner.close();
        }catch(InputMismatchException e) {
            System.out.println("Entrada inválida");
        }catch(ArithmeticException e) {
            System.out.println("Divisão inválida");
        }
        System.out.println("FIM");
    }
}
```

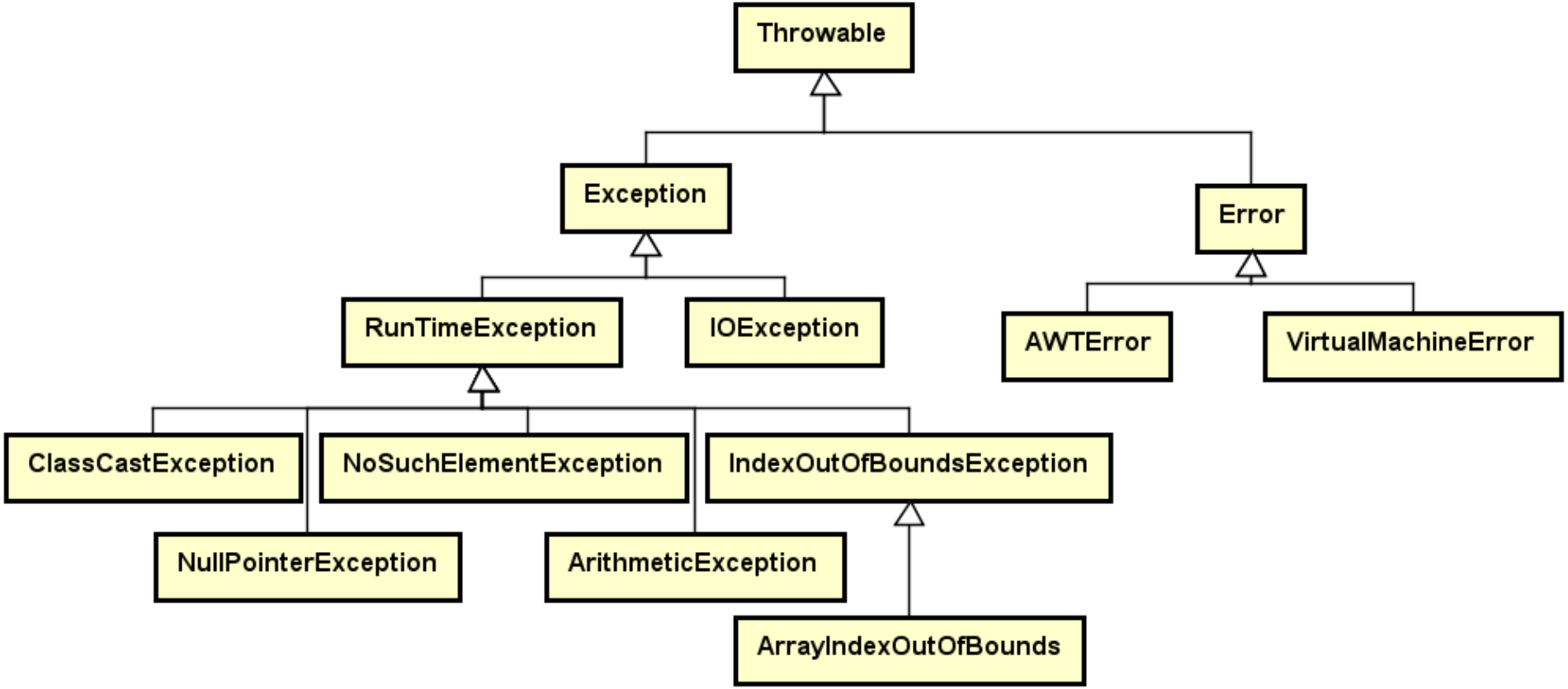
Se entrada “0” saída:

```
INÍCIO
Digite um número: 0
0
Depois da impressão do número
Divisão inválida
FIM
```

Linhas executadas

```
1  import java.util.InputMismatchException;
2  import java.util.Scanner;
3
4  public class Exemplo1 {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("INÍCIO");
8          try {
9            System.out.print("Digite um número: ");
10           int num = scanner.nextInt();
11           System.out.println(num);
12           System.out.println("Depois da impressão do número");
13           System.out.println(10/num);
14           System.out.println("Depois da divisão do número");
15           scanner.close();
16         }catch(InputMismatchException e) {
17             System.out.println("Entrada inválida");
18         }catch(ArithmeticException e) {
19             System.out.println("Divisão inválida");
20         }
21         System.out.println("FIM");
22     }
23 }
```

Hierarquia de Classes


Throwable
Exception
Error
RunTimeException
IOException
AWTError
VirtualMachineError
ClassCastException
NoSuchElementException
IndexOutOfBoundsException
NullPointerException
ArithmeticException
ArrayIndexOutOfBounds
powered by Astah

TRY/CATCH: SINTAXE X HIERARQUIA DE CLASSES

```
try{
    // código que pode gerar exceção
}
catch(Exceção e1){
        //tratamento exceção 1
}
catch(Exceção e2){
        //tratamento exceção 2
}
```

1o colocar blocos catch das subclasses

O último catch deve ser o da superclasse

Capturando Qualquer Exceção

Para capturar qualquer exceção basta fazer um catch que capture a classe Exception

Vantagem: mais rápido

Desvantagem: construção muito vaga, porque não se sabe ao certo qual foi a exceção gerada

```
catch (Exception e) {

    //...

}
```

Métodos Classe Exception

`String getMessage()`

Retorna mensagem passada pelo construtor

`String toString()`

Retorna nome da exceção e mensagem

`void printStackTrace()`

imprime detalhes sobre exceção

FINALLY

FINALLY

Após o último bloco catch pode ser definido um bloco finally (opcional)

Fornece o código, que é sempre executado independente de uma exceção ocorrer

O bloco finally é ideal para códigos de liberação de recursos

Se não houver blocos catch seguindo o bloco try, o bloco finally é requerido

FINALLY: SINTAXE

```
try{
   // código que pode gerar exceção
}catch(Exceção e1){
         //tratamento exceção 1
}catch(Exceção e2){
        //tratamento exceção 2
}finally{
   // sempre executa
}
```

Execução com finally

```
4  public class Exemplo1 {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("INÍCIO");
8          try {
9            System.out.print("Digite um número: ");
10           int num = scanner.nextInt();
11           System.out.println(num);
12           System.out.println("Depois da impressão do número");
13           System.out.println(10/num);
14           System.out.println("Depois da divisão do número");
15         }catch(InputMismatchException e) {
16             System.out.println("Entrada inválida");
17         }catch(ArithmeticException e) {
18             System.out.println("Divisão inválida");
19         }finally {
20             scanner.close();
21             System.out.println("NO FINALLY");
22         }
23         System.out.println("FIM");
24     }
25 }
```

Execução com finally

```
4  public class Exemplo1 {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("INÍCIO");
8          try {
9            System.out.print("Digite um número: ");
10           int num = scanner.nextInt();
11           System.out.println(num);
12           System.out.println("Depois da impressão do número");
13           System.out.println(10/num);
14           System.out.println("Depois da divisão do número");
15         }catch(InputMismatchException e) {
16             System.out.println("Entrada inválida");
17         }catch(ArithmeticException e) {
18             System.out.println("Divisão inválida");
19         }finally {
20             scanner.close();
21             System.out.println("NO FINALLY");
22         }
23         System.out.println("FIM");
24     }
25 }
```

Note que:
A instrução scanner.close() deve sempre fechar o recurso, independente de ocorrer exceção ou não, então ela deve ficar no bloco finally, pois isso garante que ela será SEMPRE executada

Se entrada “1” saída:

```
INÍCIO
Digite um número: 1
1
Depois da impressão do número
10
Depois da divisão do número
NO FINALLY
FIM
```

```java
public class Exemplo1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("INÍCIO");
        try {
          System.out.print("Digite um número: ");
          int num = scanner.nextInt();
          System.out.println(num);
          System.out.println("Depois da impressão do número");
          System.out.println(10/num);
          System.out.println("Depois da divisão do número");
        }catch(InputMismatchException e) {
            System.out.println("Entrada inválida");
        }catch(ArithmeticException e) {
            System.out.println("Divisão inválida");
        }finally {
            scanner.close();
            System.out.println("NO FINALLY");
        }
        System.out.println("FIM");
    }
}
```

```java
public class Exemplo1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("INÍCIO");
        try {
          System.out.print("Digite um número: ");
          int num = scanner.nextInt();
          System.out.println(num);
          System.out.println("Depois da impressão do número");
          System.out.println(10/num);
          System.out.println("Depois da divisão do número");
        }catch(InputMismatchException e) {
            System.out.println("Entrada inválida");
        }catch(ArithmeticException e) {
            System.out.println("Divisão inválida");
        }finally {
            scanner.close();
            System.out.println("NO FINALLY");
        }
        System.out.println("FIM");
    }
}
```

Se entrada “a” saída:

```
INÍCIO
Digite um número: a
Entrada inválida
NO FINALLY
FIM
```

Se entrada “0” saída:

```
INÍCIO
Digite um número: 0
0
Depois da impressão do número
Divisão inválida
NO FINALLY
FIM
```

```
4  public class Exemplo1 {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          System.out.println("INÍCIO");
8          try {
9            System.out.print("Digite um número: ");
10           int num = scanner.nextInt();
11           System.out.println(num);
12           System.out.println("Depois da impressão do número");
13           System.out.println(10/num);
14           System.out.println("Depois da divisão do número");
15         }catch(InputMismatchException e) {
16             System.out.println("Entrada inválida");
17         }catch(ArithmeticException e) {
18             System.out.println("Divisão inválida");
19         }finally {
20             scanner.close();
21             System.out.println("NO FINALLY");
22         }
23         System.out.println("FIM");
24     }
25 }
```

Execução das Exceções: passos

O controle do programa deixa o bloco **try**

Os blocos **catch** são pesquisados em ordem a procura do tratador apropriado

Se um tipo de exceção disparada corresponder ao tipo de parâmetros em um dos blocos **catch**, o código desse bloco é executado

Se nenhuma exceção for disparada pelo bloco **try** os tratadores **catch** são ignorados

Se um bloco **finally** aparece após o último **catch**, ele é executado independentemente de uma exceção ter sido disparada

Como tratar as exceções verificadas pelo compilador?

Quais são as exceções que devem ser tratadas ?

Passo 1: localizar na documentação do Java a classe que deseja utilizar

Passo 2: localizar o método que deseja utilizar no programa

Passo 3: ativar a documentação do método

Passo 4: verificar se o método propaga alguma exceção (instrução **throws** na assinatura do método)

Obs.:

Caso o método possua a instrução throws você deve tratar a exceção que aparece, caso contrário não precisa

Como tratar a exceção verificada pelo compilador (linha 13)?

```
1  import java.io.File;
2  import javax.swing.JOptionPane;
3
4  public class Exemplo2 {
5      public static void main(String[] args) {
6          String nomeArq = JOptionPane.showInputDialog("Informe o nome do arquivo:");
7          String menu = "1 - Criar arquivo\n2 - Excluir arquivo\nInforme uma opção:";
8          String op = JOptionPane.showInputDialog(menu);
9          int opcao = Integer.parseInt(op);
10         File f= new File(nomeArq);
11         switch (opcao){// converte String em int
12             case 1:// cria arquivo
13                     if (f.createNewFile()==true)
14                         System.out.println("Arq. criado");
15                     else
16                         System.out.println("Arq. já existe");
17                     break;
18         }
19     }
20 }
```

```java
public class Exemplo2 {
    public static void main(String[] args) {
        String nomeArq = JOptionPane.showInputDialog("Informe o nome do arquivo:");
        String menu = "1 - Criar arquivo\n2 - Excluir arquivo\nInforme uma opção:";
        String op = JOptionPane.showInputDialog(menu);
        int opcao = Integer.parseInt(op);
        File f= new File(nomeArq);
        switch (opcao){// converte String em int
            case 1:// cria arquivo
            try {
                if (f.createNewFile()==true)
                    System.out.println("Arq. criado");
                else
                    System.out.println("Arq. já existe");
            } catch (IOException e) {
                System.out.println("Não conseguiu criar o arquivo!");
                e.printStackTrace();
            }
            break;
        }
    }
}
```

Exceção identificada na documentação da classe

THROWS

Através da cláusula **throws** é possível listar, na declaração de um método, as exceções que podem ser disparadas por este

A sintaxe para usar essa instrução é dada por:

```
<modificadores> <tipo_retorno> <nome_método>
    (<lista_parâmetros>) throws <nome_exceção1>
                                ,... <nome_exceçãoN>{
        // código do método
}
```

THROWS

throws declara que o método pode **provocar** exceções

Considerando que:

- a unidade de execução pode detectar problemas mas geralmente não sabe como tratá-lo

- a unidade requisitante não pode detectar problemas mas geralmente sabe como tratá-los

- é conveniente saber quais são as exceções que um método pode disparar para providenciar um tratador

- Uma declaração throws é obrigatória em métodos e construtores que deixam de capturar uma ou mais exceções que ocorrem em seu interior

```
public void m1() throws Excecao1, Excecao2 {...}
```

THROWS: EXEMPLO

```
2  public class Exemplo3{
3      public static void m1() throws Exception{
4            System.out.println("Executou m1()!" );
5            //.....
6       }
7       public  static void main(String args[]){
8              try{
9                   m1();
10             }catch(Exception e){
11                   System.out.println("Erro!" );
12             }
13      }
14 }
```

THROW

Comando throw:

- dispara explicitamente um tipo de exceção

- pré-definido ou definido pelo usuário

É possível que o tratador catch que capturou uma exceção decida que não é capaz de processá-la, neste caso o tratador pode dispará-la novamente usando a instrução throw

THROW

Sintaxe da instrução throw

```
throw new <nome classe de exceção>();
```

Para forçar a ocorrência de uma exceção, utiliza-se a palavra reservada **throw** (no singular)

Disparar uma exceção: throw

throws (plural)
propaga exceção

throw (singular)
Causa/gera a exceção

```
2  public class Exemplo3{
3      public static void m1() throws Exception{
4          throw new Exception();
5      }
6      public  static void main(String args[]){
7          try{
8              m1();
9          }catch(Exception e){
10             System.out.println("Erro!" );
11         }
12     }
13 }
```

Exemplo todas instruções

```
2  public class Exemplo3{
3      public static void m1() throws Exception{
4          System.out.println("ANTES throw()");
5          throw new Exception();
6      }
7      public  static void main(String args[]){
8              System.out.println("INÍCIO");
9              try{
10                 System.out.println("ANTES m1()");
11                 m1();
12                 System.out.println("DEPOIS m1()");
13             }catch(Exception e){
14                   System.out.println("NO CATCH!" );
15             }finally {
16                 System.out.println("NO FINALLY!" );
17             }
18     }
19 }
```

```
INÍCIO
ANTES m1()
ANTES throw()
NO CATCH!
NO FINALLY!
```

Exceções: resumo

O tratamento de exceções não pode ser usado para substituir testes

Em um mesmo método procure agrupar todas as exceções que ocorrem em um mesmo bloco try-catch

Quando capturar uma exceção sempre exibir uma mensagem indicando o erro que ocorreu **NUNCA USAR SÓ**: catch(Exception e){}

Procure não propagar as exceções, quanto mais perto da instrução que gerou a exceção, mais preciso será o tratamento do erro