

TRATAMENTO DE EXCEÇÕES

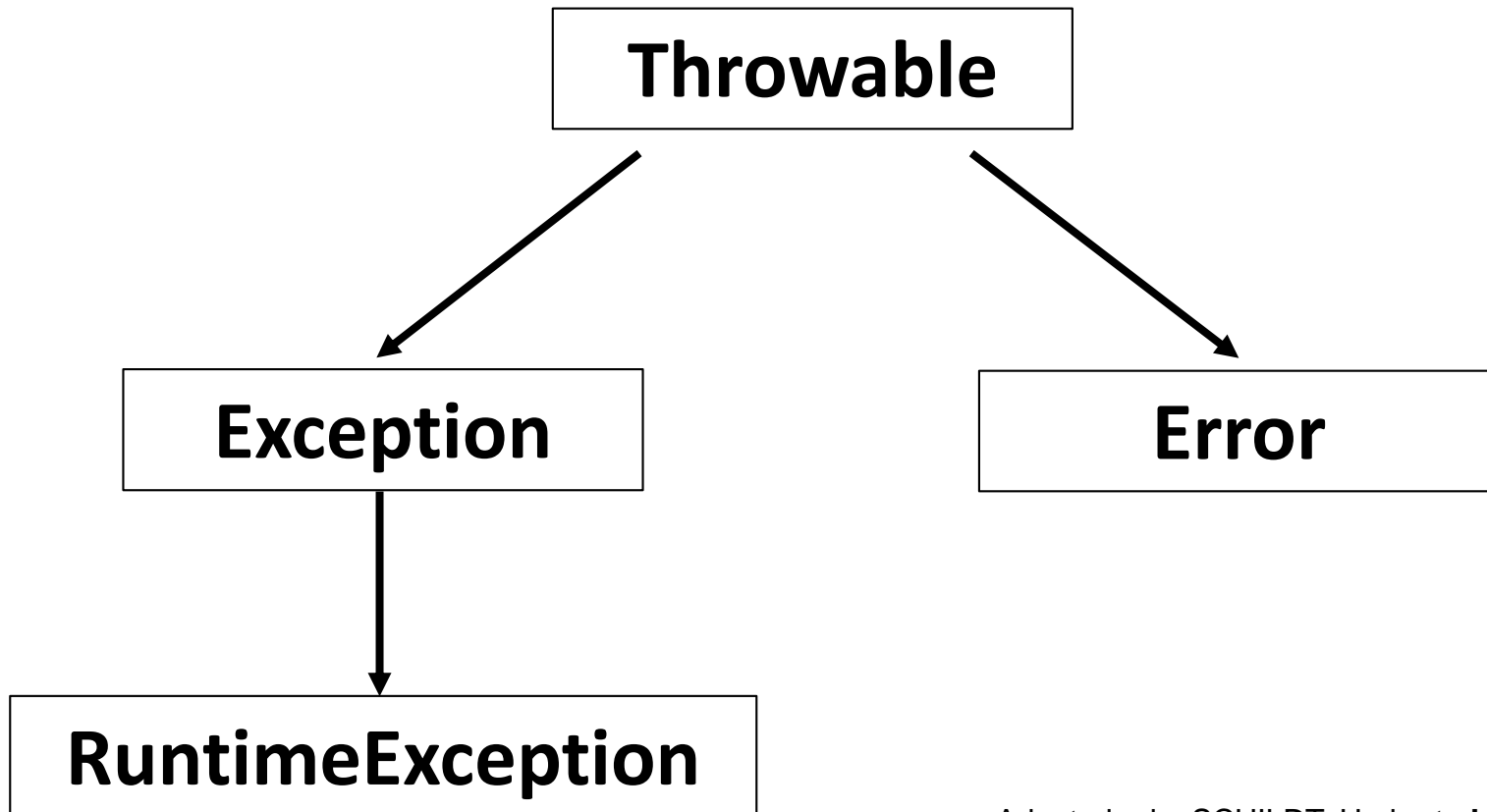
Exceções

- Exceção é uma condição anormal que surge em uma sequencia de código durante a execução. Ou seja: é um erro que acontece no tempo de execução. ^[1]
- Uma exceção Java descreve um erro uma condição excepcional (erro) que ocorreu em uma parte do código. Neste caso, um objeto representando esta exceção é criado e inserido no método que causou o erro. ^[1]
- Este método pode optar por manipular sozinho a exceção ou passa-la adiante. De qualquer maneira, em algum ponto a exceção é capturada e processada. ^[1]

Exceções

- Em Java, todas as exceções são representadas por classes e todas as classes de exceções são derivadas de uma classe chamada **Throwable**. ^[1]
- Logo, quando uma exceção ocorre em um programa, um objeto de algum tipo de classe de exceção é gerado. Há duas subclasses diretas do Throwable: **Exception e Error**. ^[1]
- As exceções do tipo Error estão relacionadas a erros na própria JVM e fogem ao controle do programador. Erros que resultam da atividade do programa são representadas por Exception. ^[1]

Hierarquia de Exceções



Adaptado de: SCHILDT, Herbert. **Java: a referência completa**. 1 ed. Porto Alegre: Bookman, 2014.

Exceção Checked e Unchecked

Exceção Verificada (checked): Representam condições inválidas em áreas fora do controle imediato do programa (problemas de entradas do usuário inválidas, banco de dados, falhas de rede, arquivos ausentes). São subclasses de Exception. Um método é obrigado a estabelecer uma política para todas as exceções verificadas lançadas por sua implementação (ou passar a exceção verificada mais acima na pilha, ou manipulá-lo de alguma forma).

Exceção Não Verificada (unchecked): Representam defeitos no programa. Em geral, refletem erros na lógica do programa e não podem ser recuperados em tempo de execução. São subclasses de RuntimeException. Um método não é obrigado a estabelecer uma política para as exceções não verificadas lançadas pela execução.

Adaptado de: <https://www.devmedia.com.br/checked-exceptions-versus-unchecked-exceptions-trabalhando-com-excecoes/29626>

Classe Exception

- Em geral, os programas devem tratar exceções do tipo Exception. Uma subclasse importante de Exception é a RuntimeException que é usada para representar vários tipos comuns de erros de tempo de execução.^[1]
- Exemplos de erros que resultam da atividade dos programa (exceptions) ^[1]
 - Divisão por Zero
 - Exceder limites de arrays
 - Erros relacionados a Arquivos.



Tratamento de Erros

- As instruções do programa que devem ser monitoradas em busca de exceção, ficarão dentro de um bloco try. Se ocorrer uma exceção ela será **lançada**.^[1]
- O código poderá capturar essa exceção usando catch e trata-la. Exceções geradas pelo sistema são lançadas automaticamente pelo Java.^[1]
- Uma exceção lançada para fora de um método, deve ser especificada como tal por uma clausula throws. Qualquer código que deve ser executado ao sair de um bloco try deve ser inserido em um bloco finally.^[1]

3 Condições que Geram Exceções

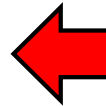
Exceção	Contexto	Programador
Máquina Virtual	Em resposta a um erro interno	Não tem controle
Padrão	Erros no código do programa	Deve tratar essas exceções
Manualmente	Lançada pelo programador	Utilizam a instrução Throw

Adaptado de: SCHILDT, Hebert. **Java para iniciantes: crie, compile e execute programas Java rapidamente**. 6 ed. Porto Alegre: Bookman, 2015.

Exceções Não Capturadas

```
package TratamentoErros;  
public class Matematica{  
    public static void Main (String args[]){  
        int num = 0  
        int res = 15 / num;  
    }  
}
```

Mas nenhum manipulador de exceção foi fornecido, portanto, a exceção é capturada pelo manipulador padrão fornecido pelo sistema de execução Java. Qualquer exceção que não seja capturada pelo seu programa acabará sendo processada pelo manipulador padrão. [2]



O programa ao lado causa um erro bastante comum: a divisão por zero, que não é possível. [2]

Ao perceber esse erro, o programa cria um objeto de exceção e em seguida, lança a exceção, fazendo com que a execução da classe matemática pare. [2]

Uma vez que a exceção é lançada, ela deve ser capturada por um manipulador de exceções e solucionada imediatamente. [2]

Stack Trace

```
package TratamentoErros;  
public class Matematica{  
    public static void Main (String args[]){  
        int num = 0  
        int res = 15 / num;  
    }  
}
```

O manipulador padrão vai descrever a exceção e imprimir um **Stack Trace** (detalhamento da exceção lançada), a partir do ponto em que a exceção ocorreu e encerra o programa. [2]

```
"C:\Program Files\Java\jdk1.8.0_201\bin\java.exe" ...  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at TratamentoErros.Matematica.main(Matematica.java:8)
```

O tipo de exceção gerada é uma subclasse de Exception, chamada ArithmeticException, que descreve que tipo de erro ocorreu. O Java fornece diversos tipos integrados de exceção, que combinam com os vários tipos de erros de execução que podem ser gerados. [2]

Tratamento de Erros

- Um dos principais benefícios do tratamento de exceções é que ele permite que seu programa responda a um erro e então continue a ser executado.^[1]
- Por exemplo: No caso da divisão por erro, uma `AithmeticException` será gerada. No programa, a exceção é tratada pode ser tratada pelo relato do erro e a execução continua.^[1]
- Tentar dividir por zero não causaria um erro abrupto de tempo de execução, resultando no encerramento do programa. Em vez disso, a situação é tratada, permitindo que a execução do programa continue.^[1]

Tratamento de Erros

- Usuários podem ficar confusos se o programa parar de funcionar e imprimir um stack trace sempre que um erro acontecer. [2]
- Para solucionar um erro de execução, simplesmente insira o código a ser monitorado dentro de um bloco try. [2]
- Após o bloco try, inclua um bloco catch que especifica o tipo de exceção que você deseja capturar. [2] Cada bloco catch é tratador de exceção e trata o tipo de exceção indicado por seu argumento – que declara o tipo de exceção que o tratador pode lidar e deve ter o nome de uma classe herdada da classe Throwable. [3]

Bloco Try Catch

```
try{  
    // código  
}  
catch
```

Em um tratador de exceção, envolvemos linhas de código que podem lançar uma exceção dentro do bloco try. É possível colocar cada linha de código que pode lançar uma exceção do método dentro de um bloco try. ^[3]

A associação de tratadores de exceção com um bloco try é feita fornecendo um ou mais blocos catch diretamente depois do bloco try (nenhum código pode estar entre o final do bloco try e o começo do primeiro bloco catch). ^[3]

Bloco Try Cacth

```
try{  
    // código  
}  
catch (TipoExceção1 obEx){  
    // tratador  
}  
catch (TipoExceção2 obEx){  
    // tratador  
}
```

As palavras chaves try e catch formam a base o tratamento de exceção. Não é possível ter um catch sem try, pois elas trabalham em conjunto. ^[1]

TipoExceção é o tipo de exceção que ocorreu. Quando uma exceção é lançada, ela é capturada pela instrução catch correspondente que então a processa. ^[1]

Tratamento de Erros

```
try{  
    // código  
}  
catch (TipoExceção1 ob1){  
    // tratador  
}  
catch (TipoExceção2 ob2){  
    // tratador  
}
```

O tipo da exceção determina que instrução catch será executada. ^[1]

Se o tipo de exceção especificado por uma instrução catch coincidir com o da exceção ocorrida, a instrução catch será executada (as outras serão ignoradas). ^[1]

Quando a exceção é capturada, um objeto (ob) recebe seu valor. ^[1]

Exemplo - Exceção Simples

```
public class Matematica {  
  
    public static void main(String[] args) {  
  
        try {  
            int num = 0;  
            int res = 15 / num; ← Tenta fazer uma  
                                divisão por zero  
        }  
        catch (ArithmeticException obj) { ← Captura erros  
                                            de Arimética  
            System.out.println("Não é possível dividir por zero");  
        }  
    }  
}
```


Saída do programa: "Não é possível dividir por zero"

Exemplo - Exceção Simples

Classe Matemática

```
public class
Matematica {

    public void dividir() {
        int num = 0;
        int res = 15 / num;
    } ArithmeticException
}
```



Exceções do código que ficam dentro de um bloco try são monitoradas. Isso inclui exceções que podem ser geradas por um método chamado dentro de um bloco try.^[1]

Classe Main

```
public class Main {
    a exceção é capturada aqui
    public static void main(String[] args) {

        Matematica m = new Matematica();

        try {
            m.dividir();
        } catch (ArithmeticException obj) {
            System.out.println("Não é possível
            dividir por zero");
        } dividir() é chamado dentro de um bloco try.
        A exceção que dividir() gera e não captura, é
        capturada pela instrução catch de main(). Se
        dividir() tivesse capturado a exceção, ela
        nunca teria sido passada para main().[1]
    }
}
```

Exemplo adaptado de: SCHILDT, Hebert. **Java para iniciantes: crie, compile e execute programas Java rapidamente**. 6 ed. Porto Alegre: Bookman, 2015.

É possível associar mais de uma instrução catch a uma instrução try (o que é bem comum). No entanto, cada catch deve capturar um tipo de exceção diferente. Neste exemplo, o programa captura erros tanto de limite de arrays quanto de divisão por zero.^[1]

O array N tem 8 posições. O array d tem 6 posições. Através do laço for, o programa vai passar por cada posição dos arrays dividido a posição x de um pela posição x do outro. Isto é: A cada passagem do for, será dividido o valor que está na posição 0 do array i (4) pelo valor da posição zero do array d (2), depois a posição 1, posição 2 e assim por diante.

Na primeira passagem, não haverá problema algum, pois 4 é divisível por 2. Mas na passagem seguinte do laço, a posição 1 do array i (8) será dividida pela posição 1 do array d (0), levando a um erro de divisão por 0. Será lançada uma ArithmeticException.

Outro problema é que as divisões vão acontecendo, mas há mais índices no array n do que no array d. Ou seja: nas duas últimas passagem do laço, não será possível dividir o valor do índice 6 e 7 de i, pois não há esses índices em d. Com isso, será lançada uma ArrayIndexOutOfBoundsException .

```
public class Excecao {
    public static void main(String[] args) {

        int n [] = {4,8,16,32,64,128,256,512};
        int d [] = {2,0,4,4,0,8};

        for (int i = 0; i < 8; i++){

            try{
                System.out.println(n [i] + "/" + d [i] + " é " + n [i] / d [i]);
            }
            catch (ArithmeticException a){
                System.out.println("Não é possível dividir por zero");
            }
            catch (ArrayIndexOutOfBoundsException b){
                System.out.println("Nenhum elemento foi encontrado");
            }
        }
    }
}
```

Saída do programa:

4/2 é 2
Não é possível dividir por zero
16/4 é 4
32/4 é 8
Não é possível dividir por zero
128/8 é 16
Nenhum elemento foi encontrado
Nenhum elemento foi encontrado

**Só uma instrução
que apresente
correspondência
é executada.
Todos os outros
blocos catch são
ignorados.^[1]**

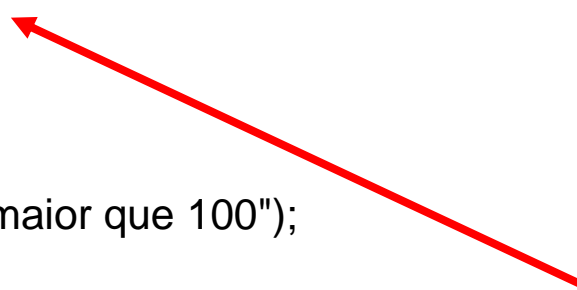
Exemplo adaptado de: SCHILDT, Hebert. **Java para iniciantes: crie, compile e execute programas Java rapidamente**. 6 ed. Porto Alegre: Bookman, 2015.

Lançando Uma Exceção

```
public class Main {  
    public static void main(String[] args) {  
        porcentagem(101);  
    }  
  
    public static void porcentagem(int p){  
  
        try {  
            if (p > 100){  
                throw new IllegalArgumentException();  
            }  
        }  
        catch (IllegalArgumentException a) {  
            System.out.println("O valor não pode ser maior que 100");  
        }  
        finally {  
            System.out.println("Exceção Lançada");  
        }  
    }  
}
```

É possível lançar manualmente uma exceção usando a instrução throw, que será um objeto de uma classe de exceção derivada de Throwable. A exceção é criada com o uso de new na instrução throw - que lança um objeto - logo você deve criar um objeto para lançar.^[1]

Lançamento manual de uma
IllegalArgumentException



Exemplo adaptado de: SCHILDT, Hebert. **Java para iniciantes: crie, compile e execute programas Java rapidamente**. 6 ed. Porto Alegre: Bookman, 2015.

Finally

```
try{  
    // código  
}  
catch (TipoExceção1 obEx){  
    // tratador  
}  
catch (TipoExceção2 obEx){  
    // tratador  
}  
finally{  
    //código de finally  
}
```


Para especificar um bloco de código para ser executado na saída de um bloco try/catch, podemos incluir um bloco Finally no final de uma sequencia try/catch.^[1]

O bloco finally será executado após o bloco try/catch, independente se o bloco tiver encerrado normalmente ou por uma exceção gerada (e mesmo se nenhuma instrução catch for compatível com a exceção). O último bloco executado será o definido por finally.^[1]

Lançando Uma Exceção

```
public class Dirigir {  
    public void checarIdade(int idade) {  
  
        if (idade < 18) {  
            throw new ArithmeticException("Você não pode dirigir");  
        }  
        else {  
            System.out.println("Você pode dirigir");  
        }  
    }  
  
    public static void main(String[] args) {  
        Dirigir d = new Dirigir();  
        d.verificarIdade(15);  
    }  
}
```

**A instrução throw
permite que você crie
um erro personalizado,
usada junto com um
tipo de exceção.**



```
Exception in thread "main" java.lang.ArithmeticException: Você não pode dirigir  
    at TratamentoErros.Dirigir.verificarIdade(Dirigir.java:8)  
    at TratamentoErros.Dirigir.main(Dirigir.java:17)
```

Exemplo adaptado de: https://www.w3schools.com/java/java_try_catch.asp

Exceções em Java

Java define vários outros tipos de exceções relacionadas às suas várias bibliotecas de classes. A seguir, são apresentada uma breve lista de algumas exceções sem verificação do Java.

Exceção	Descrição
ArithmeticException	Dividindo por zero
ArrayIndexOutOfBoundsException	Índice de um array não existe.
ClassCastException	Alterar um objeto String para Double
NullPointerException	Chamar um objeto não disponível
IllegalArgumentException	Argumento ilegal para chamar método.

Adaptado de: <https://study.com/academy/lesson/exceptions-in-java-definition-example.html>

Referências Bibliográficas e Sites Consultados

[1] SCHILDT, Hebert. **Java para iniciantes: crie, compile e execute programas Java rapidamente.** 6 ed. Porto Alegre: Bookman, 2015.

[2] SCHILDT, Herbert. **Java: a referência completa.** 1 ed. Porto Alegre: Bookman, 2014.

[3] GALLARDO, R.; KANNAN, S.; ZACKHOUR, S. B. **Tutorial Java.** 5 ed. Rio de Janeiro: Alta Books, 2015.

https://www.dcc.ufrj.br/~fabiom/java20121/excecoes_jonathan.pdf

https://www.tutorialspoint.com/java/java_builtin_exceptions.htm

<https://study.com/academy/lesson/exceptions-in-java-definition-example.html>

<https://www.devmedia.com.br/checked-exceptions-versus-unchecked-exceptions-trabalhando-com-excecoes/29626>

<https://www.devmedia.com.br/entendendo-java-exceptions/29812>

https://www.w3schools.com/java/java_try_catch.asp

**Material desenvolvido pelo
Prof. Rafael da Silva Polato**
rafael.polato@etec.sp.gov.br

Divisão de Turma
Prof. Leandro Bordignon
leandro.bordignon01@etec.sp.gov.br