



Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0084 Ano: 2021
Professor: Ismayle de Sousa Santos

Aula
11 e
12

Sistemas de Informações e Banco de Dados

Exceções e Tratamento de Exceções em Java



rfbrkh3



ismaylesantos@great.ufc.br



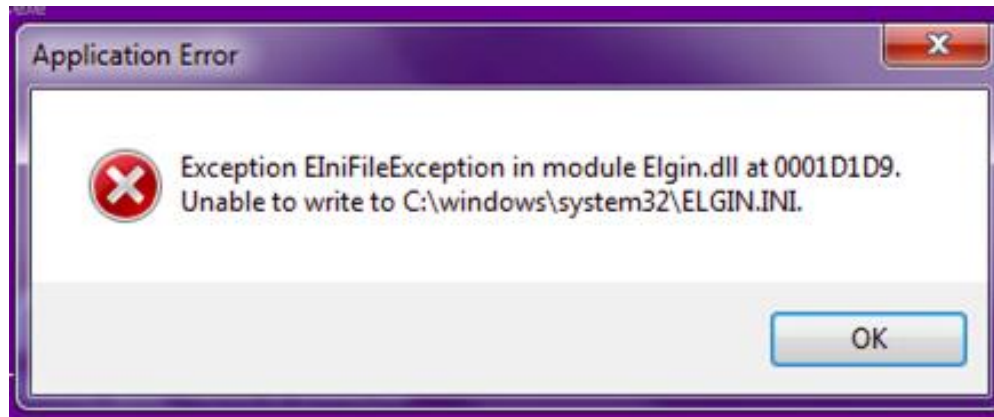
@IsmayleSantos

Hoje aprenderemos sobre ...

- Visão geral sobre tratamento de exceções em Java
 - O que são exceções?
 - Como tratar as exceções?
 - Como criar uma exceção?
 - O que é um *Error*?
 - Tratamento de erros
-

O que são Exceções?

- Exceções são os erros imprevistos durante a execução programas de computador
 - Ou ainda, são **comportamentos inesperados** que o programa assume
- As exceções **podem ser provenientes erros de lógica ou acesso a recursos que talvez não estejam disponíveis**



O que são Exceções?

- Exemplos de exceções provenientes de **erros de lógica** são:
 - Tentar manipular um objeto que está com o valor nulo
 - Dividir um número por zero (Java não permite!)
 - Tentar manipular um tipo de dado como se fosse outro
 - Tentar utilizar um método ou classe não existentes

java.lang.ArithmeticException: / by zero

O que são Exceções?

- Exemplo de exceções provenientes de **motivos externos**:
 - Tentar abrir um arquivo que não existe
 - Tentar fazer consulta a um banco de dados que não está disponível
 - Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita
 - Tentar conectar em servidor inexistente
 - *Timeout* de comunicação com outro software

Note que nem sempre o que causa uma exceção é um problema no código-fonte

Qual a Diferença entre Erro e Exceção?

- A diferença entre erro e exceção é que um **erro** é causado devido à falta de recursos do sistema e uma **exceção** é causada por causa de seu código
 - **Erro** é uma condição crítica que **não pode ser tratada** pelo código do programa
 - Exemplo: Falta de memória pode impedir o programa de continuar
 - **Exceção** é a situação excepcional que **pode ser tratada** pelo código do programa
 - Exemplo: Se o programa aceitar uma entrada incorreta
-

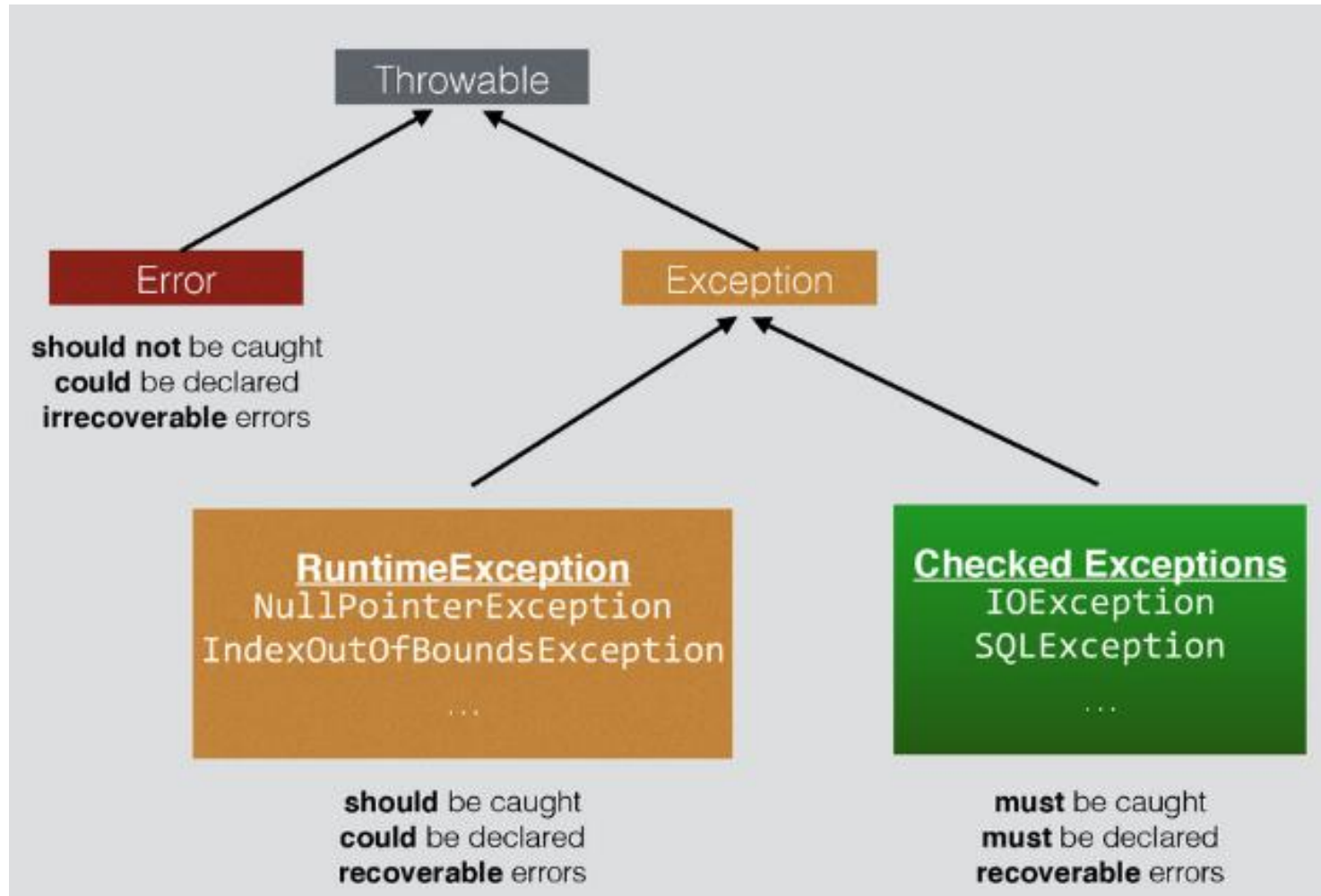
Checked e Unchecked Exceptions

- **Throwable** é a superclasse de todos os erros e exceções na linguagem Java
 - **RuntimeException** é a superclasse dessas exceções que podem ser lançadas durante a operação normal da Java Virtual Machine
 - Não é obrigatório o tratamento dessa exceção, você pode tratar se sentir que é necessário
 - **Checked Exceptions (exceções verificadas)** são exceções que você é obrigado a tratá-la, seja com um bloco try-catch ou mesmo com um throws
-

Checked e Unchecked Exceptions

- **Checked exceptions** são utilizadas para erros recuperáveis enquanto que **Unchecked exceptions** são utilizadas para erros irre recuperáveis
 - Exemplo:
 - NullPointerException é irre recuperável
 - Todas as classes que herdam da classe Exceptions, mas não da classe RuntimeException são **exceções verificadas**
-

Qual a Diferença entre Erro e Exceção?



Como Tratar as Exceções?



Uma maneira de tentar contornar esses imprevistos é realizar o tratamento dos locais no código que podem vir a lançar possíveis exceções

Tratamento de Exceções

- A possibilidade de tratar uma exceção nos fornece uma maneira **mais segura de controlar o fluxo de um programa**, evitando o excesso de estruturas condicionais, o deixando assim com melhor desempenho
- Tratar um exceção também nos permite fazer um código **mais robusto** (à prova de erros de execução)

Tratamento de erro é importante, mas se ele obscurece a lógica, ele está errado!

Como Tratar as Exceções?

- Se for tratar no mesmo método
 - Usar **try-catch**
 - Se for tratar em outro método
 - O método usa **throw** para lançar a exceção
 - O bloco **try-catch** ficará encarregado de tratar as exceções
-

O que é o Throw?

- **Especifica as exceções que o método lança**
- Aparece depois da lista de parâmetros do método e antes do corpo do método
- Contém uma lista das exceções separadas por vírgulas que o método lançará se ocorrer um problema
- Essas exceções podem ser lançadas por instruções no corpo do método ou por métodos chamados no corpo

```
public void metodoComExceçãoVerificada () throws FileNotFoundException {
```

Como usar o Bloco Try-Catch?

- `try{ ... }`
 - Neste bloco são introduzidas todas as linhas de código que podem vir a lançar uma exceção
 - Toda a execução dentro de um `try` pode ser interrompida a qualquer momento por uma exceção
 - `catch(tipo_excessao e) { ... }`
 - Neste bloco é descrita a ação que ocorrerá quando a exceção for capturada
 - O bloco `catch` tem que deixar o programa em um estado consistente
-

Como usar o Bloco Try-Catch?

- Se ocorrer uma exceção:
 - O código restante do bloco try será pulado e o controle vai para o bloco catch
 - Se ocorrer uma exceção em um bloco try:
 - O bloco catch captura a exceção
 - O bloco catch que será executado é o bloco catch cujo o tipo é correspondente ao tipo da exceção que ocorreu
 - Todo bloco **catch** exibe uma mensagem de erro e pede para o usuário tentar novamente
-

Sintaxe do Bloco Try-Catch

```
1  try
2  {
3      //trecho de código que pode vir a lançar uma exceção
4  }
5  catch(tipo_excecao_1 e)
6  {
7      //ação a ser tomada
8  }
9  catch(tipo_excecao_2 e)
10 {
11     //ação a ser tomada
12 }
13 catch(tipo_excecao_n e)
14 {
15     //ação a ser tomada
16 }
```

Exemplo de Tratamento de Exceção

- Prefira Exceções a Códigos de Retorno

```
try {  
    deletePage(page);  
    registry.deleteReference(page.name);  
    configKeys.deleteKey(page.name.makeKey());  
}  
catch (Exception e) {  
    logger.log(e.getMessage());  
}
```

O código para deletar a página e o código de tratamento de erro agora estão separados
Está muito **mais claro** o que cada um faz!

O que é Try-Catch-Finally?

- O bloco **try-catch-finally** lida com alguns ou todos os erros que podem ocorrer em um bloco de código, **enquanto ainda executa o código**
 - O bloco **finally** representa o trecho de código que será executado independente do fato de ter ocorrido a exceção ou não
 - Programas que obtêm certos tipos de recursos devem retorná-los ao sistema explicitamente para evitar os vazamentos de recursos
 - Exemplos: arquivos, conexões de banco de dados, conexões de rede, etc...
-

Sintaxe do Bloco Try-Catch

```
try {  
}  
catch(tipo de exceção 1) {  
}  
catch(tipo de exceção 2) {  
}  
finally {  
}
```

Vale ressaltar que o bloco finally é opcional!

Exemplo de try-catch-finally

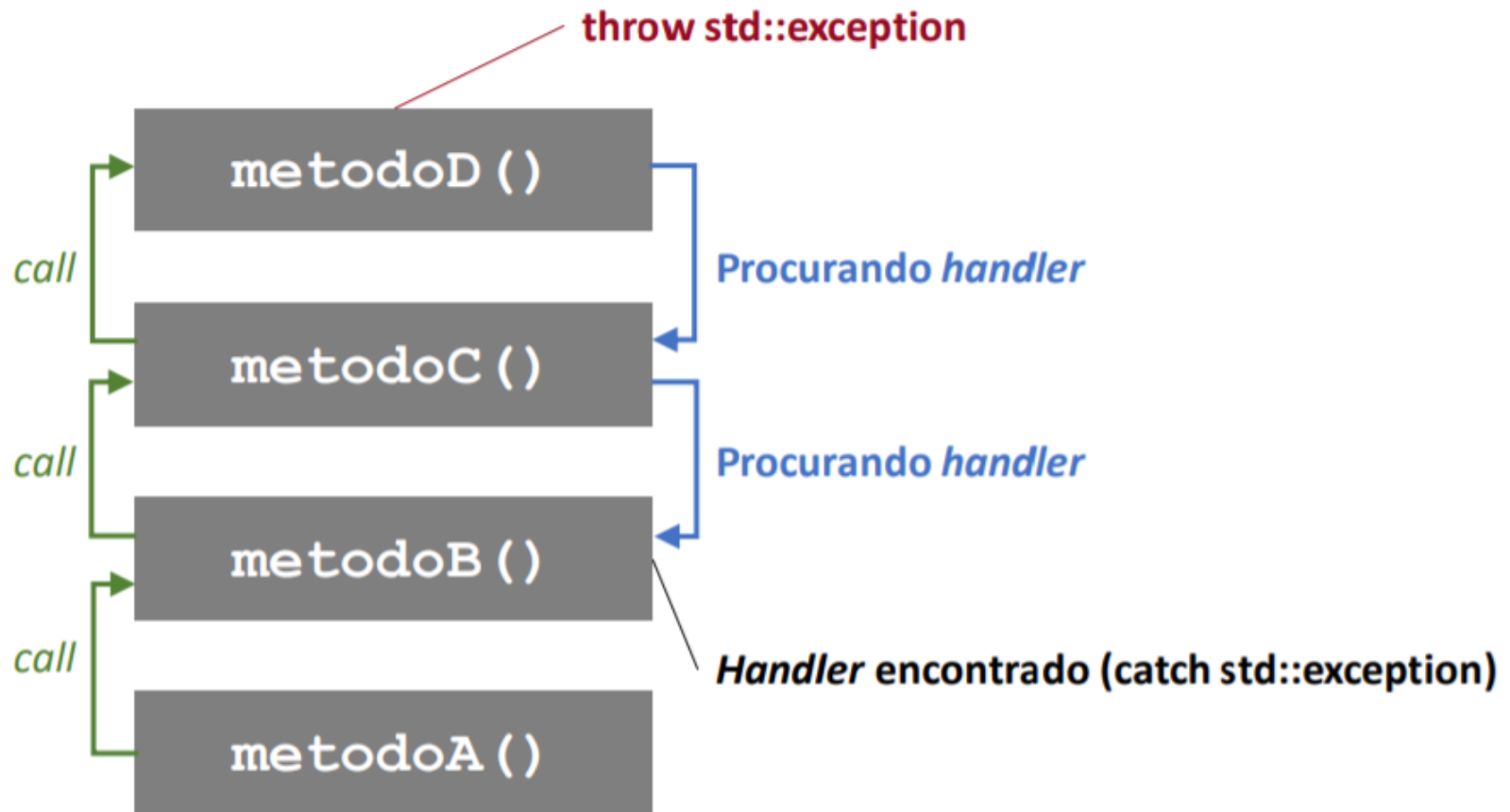
```
//bloco try-catch-finally
try {
    metodoComExceçãoNullPoint();
} catch (NullPointerException e) {
    // Tratamento da Exceção
    System.out.println("Exceção foi tratada");
} finally {
    System.out.println("Sempre é executado");
}
```

Vale ressaltar que o bloco finally é opcional!

Vamos ver exemplos!



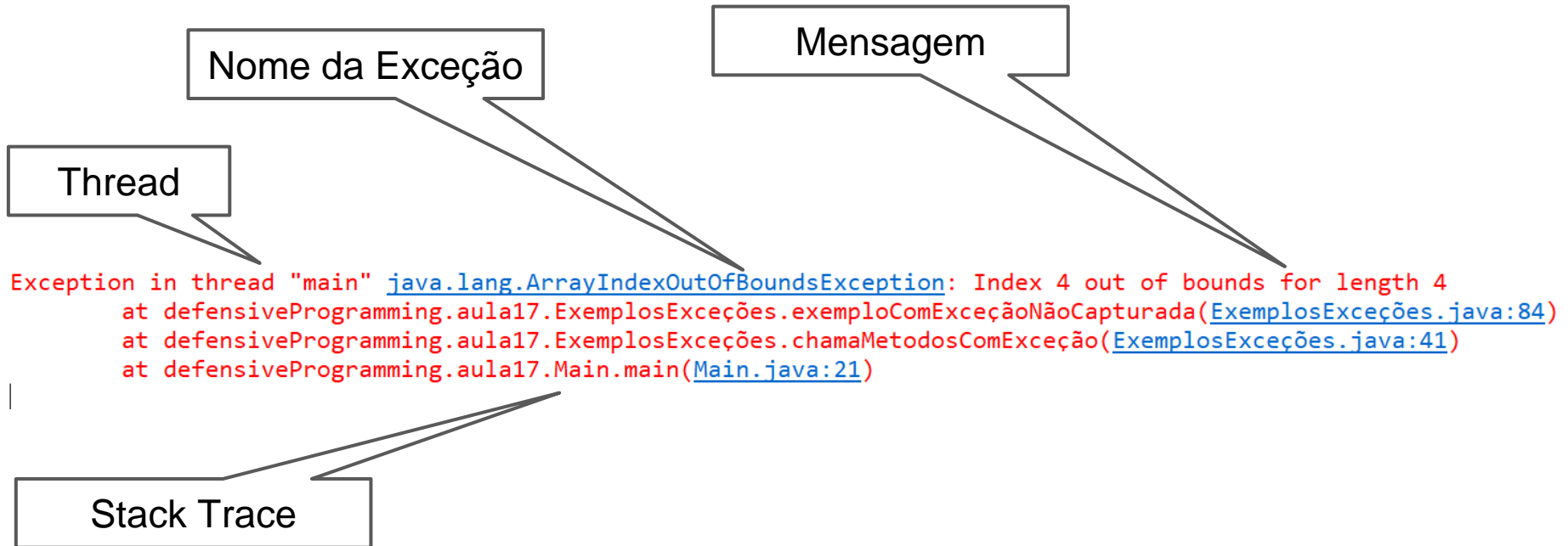
Pilha de Exceção



Informações de uma Exceção

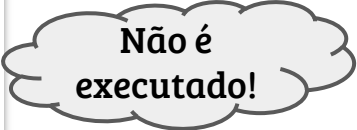
- Uma exceção contém as seguintes informações
 - Nome
 - Da classe da exceção
 - Thread
 - Onde a exceção foi lançada
 - Mensagem
 - Mensagem da exceção (fornecida durante a criação da exceção)
 - Stack Trace
 - Rastro de pilha de chamados
-

Exemplo de Informações de uma Exceção



O que Acontece se a Exceção não é Capturada?

```
public void exemploComExceçãoNãoCapturada() {  
    int[] vetor = new int[4];  
  
    try {  
        int i = vetor[4]; // vai disparar exceção  
  
        System.out.println("Comandos dentro do bloco try");  
    } catch (NullPointerException e) {  
        System.out.println("Exceção capturada");  
    }  
  
    System.out.println("Resto do Programa");  
}
```



Não é executado!

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException](#): Index 4 out of bounds for length 4
at defensiveProgramming.aula17.ExemplosExceções.exemploComExceçãoNãoCapturada([ExemplosExceções.java:76](#))
at defensiveProgramming.aula17.ExemplosExceções.chamaMetodosComExceção([ExemplosExceções.java:38](#))
at defensiveProgramming.aula17.Main.main([Main.java:20](#))

Vamos ver exemplos!



Como Criar uma Exceção?

- **Defina classes de exceção** em termos das necessidades do método que invocou a classe
 - Especialmente no caso de tratamento de exceções de APIs/bibliotecas de terceiros
 - Assim, mudar de API/biblioteca vai impactar menos o seu código
 - Você pode fazer isso estendendo alguma das classes de exceção
 - Ótima opção para encapsular exceções de bibliotecas de terceiros
-

Como Criar uma Exceção?

```
public class SemLetraBException extends Exception {  
    @Override  
    public String getMessage(){  
        return "Não existe letra B em sua frase";  
    }  
}
```

```
public class TesteExcecao {  
    public static void main(String args[]) throws SemLetraBException  
    {  
        String frase = "Sou um teste!";  
        if(!frase.contains("b") || !frase.contains("B"))  
            throw new SemLetraBException();  
    }  
}
```

Como Criar uma Exceção?

- **Forneça Contexto com as Exceções**
 - Cada exceção disparada deve fornecer contexto suficiente para determinar a fonte e o local do erro
 - Crie mensagens de erro significativas
 - Se você está utilizando log registre essas informações no log no bloco catch
-

Métodos da Classe Throwable

- **printStackTrace**
 - Envia para o fluxo de erro padrão o rastreamento da pilha
 - Útil para o processo de teste e depuração
- **getStackTrace**
 - Recupera informações sobre o rastreamento de pilha que podem ser impressas por printStackTrace
- **getMessage**
 - Retorna a string descritiva armazenada em uma exceção

Você pode sobrescrever esses métodos ao criar seu tipo de Exceção!

Vamos fazer um exemplo de exceção!



**Agora vamos para as dicas
e boas práticas =)**



Exception NullPointerException

- **NullPointerException** é uma unchecked exception, logo, ela não precisa ser tratada e o compilador não acusa erro em tempo de compilação
 - Como evitar
 - Não retorne NULL
 - Passar null para outros métodos é pior ainda!
 - Trabalhe com a ideia de que passagem de null como parâmetro é proibido
-

Exception NullPointerException

- NullPointerException é uma exceção que indica que a aplicação tentou usar uma referência de um objeto que estava com valor nulo
- Normalmente são bugs da aplicação que poderiam ter sido evitados caso o desenvolvedor tomasse mais cuidado na hora de programar

```
List<Employee> employees = getEmployees();  
if (employees != null) {  
    for(Employee e : employees) {  
        totalPay += e.getPay();  
    }  
}
```



Exception NullPointerException

- Quando você opta por não retornar null, você minimiza as chances de NullPointerExceptions

```
List<Employee> employees = getEmployees();  
    for(Employee e : employees) {  
        totalPay += e.getPay();  
    }
```

```
public List<Employee> getEmployees() {  
    if( .. there are no employees .. )  
        return Collections.emptyList();  
}
```

Exception IndexOutOfBoundsException

- A IndexOutOfBoundsException é uma exception unchecked que é lançada ao tentar acessar uma posição inexistente de algum Array, List, Matriz, String, ou um Vetor
- **Exemplo:** A exceção abaixo, está informando que o usuário está tentando o acessar o índice 4 de um Array com apenas 1 posição

```
java.lang.IndexOutOfBoundsException: Index: 4, Size: 1
```

Validação das Entradas

- Para lidar com **entradas ruins/inválidas**
 - Verificar os valores de todos os dados de fontes externas
 - Exemplo: verificar se valores numéricos de um arquivo ou entrada de usuário estão dentro dos limites esperados
 - **Verificar os valores de todos os parâmetros** de entradas de métodos
-

Validação das Entradas

- Uma vez detectado um parâmetro inválido, o que fazer?

e se num = -2 ?

```
public static int fatorial(int num) {  
    if(num == 0)  
        return 1;  
    else  
        return num * fatorial(num-1);  
}
```

Exemplos de Validação das Entradas

- O valor é numérico? É uma String?
 - Aceita valores negativos? Só maior que algum valor X?
 - Número de casas decimais importa?
 - Quantidade é válida?
 - Index do array é válido?
-

Dicas para usar Exceções

- Use exceções para notificar outras partes do software sobre erros que não deveriam ser ignorados
 - O benefício da exceção é sinalizar uma condição de erro de forma que ele não pode ser ignorado
 - Em outras abordagens de tratamento de erro, o erro pode passar despercebido pelo código base
 - Exemplo: nas técnicas de substituições a entrada inválida é substituída por uma válida
-

Dicas para usar Exceções

- Lance exceções somente para condições que são verdadeiramente excepcionais
 - Exceções são usadas para eventos não frequentes e que nunca deveriam acontecer
 - Não use exceções para “passar a bola”
 - Prefira tratar os erros localmente, se possível
 - Sem lançar exceção para métodos superiores na hierarquia de chamadas
 - Facilita a manutenção
-

Dicas para usar Exceções

- Inclua na mensagem da exceção todas as informações que levaram a exceção
 - A mensagem tem que ter a informação necessária para o entendimento do porque a exceção foi lançada
 - Exemplo: Se a exceção foi devido a erro de indexação de um array , inclua na mensagem os valores inferior e superior do array, além do valor do index inválido

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
at defensiveProgramming.aula17.ExemplosExceções.exemploComExceçãoNãoCapturada(ExemplosExceções.java:76)
at defensiveProgramming.aula17.ExemplosExceções.chamaMetodosComExceção(ExemplosExceções.java:38)
at defensiveProgramming.aula17.Main.main(Main.java:20)
```

Dicas para usar Exceções

- **Evite blocos de catch vazios**
 - Se o bloco catch está vazio
 - O bloco try está errado, levantando uma exceção sem motivo OU e por não tratar a exceção
 - Se for o caso de não tratar o erro, pelo menos registre o log em algum arquivo

```
//Sugerido - tratar a exceção no bloco catch ou pelo menos registrar no log
try {
    //Código
    throw new Exception();
} catch (Exception e) {
    //Salvar no arquivo de Log a exceção
    Logger logger = Logger.getAnonymousLogger();
    logger.log(Level.SEVERE, "exceção grave");
}
```

Dicas para usar Exceções

- **Conheça as exceções que seu código e bibliotecas usadas estão lançando**
 - Falhar em tratar exceções lançadas pelo seu código e bibliotecas usadas pode resultar em crash da sua aplicação



Dicas para usar Exceções

- Considere desenvolver um tratamento de exceção centralizado
 - Fornece um repositório central
 - Para os tipos de exceção tratados
 - Como cada exceção deve ser tratada
 - Formatação das mensagens de exceção
 - Desvantagem
 - Acoplamento do software com a classe que faz o tratamento global das exceções
-

O que é Tratamento de Erros?

- É o processo de capturar erros que podem ocorrer durante a execução do software e tomar alguma ação
 - Representam situações anormais que poderiam acontecer na JVM



Tratamentos de Erros

- Existem várias maneiras de lidar com erros
 - Retornar um valor neutro
 - Substituir pelo próximo dado válido
 - Substituir pelo valor válido mais próximo
 - Salvar o log em um arquivo
 - Retornar um código de erro
 - Exibir mensagem de erro
 - Shut Down
-

Obrigado!

Por hoje é só pessoal...

Dúvidas?



rfbrkh3



ismaylesantos@great.ufc.br



@IsmayleSantos
