



Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0236
Professor: Ismayle de Sousa Santos

Aula 19

Técnica de Programação II

Tratamento de Exceção



qpg4p5x



ismaylesantos@great.ufc.br



@IsmayleSantos

Tratamento de Exceção

- Uma das estratégias de Programação Defensiva
- Importante porque
 - Entradas podem ser incorretas/inválidas
 - Dispositivos podem falhar
 - etc



Tratamento de Exceção

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

Tratamento de Exceção

- Prefira Exceções a Códigos de Retorno

```
if (deletePage(page) == E_OK) {  
    if (registry.deleteReference(page.name) == E_OK) {  
        if (configKeys.deleteKey(page.name.makeKey()) == E_OK){  
            logger.log("page deleted");  
        } else {  
            logger.log("configKey not deleted");  
        }  
    } else {  
        logger.log("deleteReference from registry failed");  
    }  
} else {  
    logger.log("delete failed");  
    return E_ERROR;  
}
```



Tratamento de Exceção

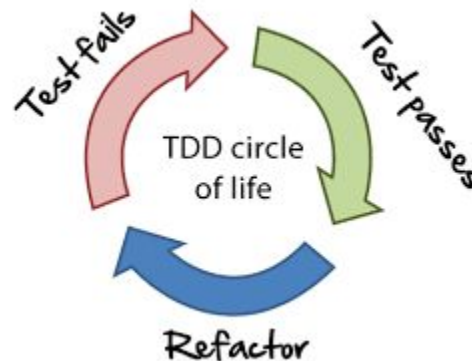
- Prefira Exceções a Códigos de Retorno

```
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!

Tratamento de Exceção

- Escreva seu bloco **try-catch-finally** primeiro
 - Toda a execução dentro de um try pode ser interrompida a qualquer momento por uma exceção
 - O bloco catch tem que deixar o programa em um estado consistente
 - Usando TDD
 - Tente primeiro escrever testes que forcem a exceção



Tratamento de Exceção

- Escreva seus bloco **try-catch-finally** primeiro
 - Exemplo

```
public List<String> recuperarTexto (String fileName){
    FileInputStream stream;
    try {
        stream = new FileInputStream(fileName);
        stream.close();
    } catch (FileNotFoundException e) {
        Logger logger = Logger.getAnonymousLogger();
        logger.log(Level.SEVERE, "Arquivo não encontrado");
    } catch (IOException e) {
        Logger logger = Logger.getAnonymousLogger();
        logger.log(Level.SEVERE, "Erro no acesso do arquivo");
    }
    return new ArrayList<String>();
}
```

Testando Exceção com JUnit

- No Junit 5 podemos especificar testes que esperam por uma exceção
 - **assertDoesNotThrow** (Executable executable)
 - Verifica se o código passado por parâmetro não lança nenhuma exceção
 - **assertThrows**(Class<T> expectedType, Executable executable)
 - Verifica se o código passado por parâmetro lança uma exceção do tipo expectedType
-

Testando Exceção JUnit

- No Junit podemos especificar testes que esperam por uma exceção
 - Exemplo

```
public List<String> recuperarTexto (String fileName) throws IOException{
    FileInputStream stream;
    try {
        stream = new FileInputStream(fileName); // pode causar FileNotFoundException
        stream.close(); // pode causar IOException
    } catch (FileNotFoundException e) {
        Logger logger = Logger.getAnonymousLogger();
        logger.log(Level.SEVERE, "Arquivo não encontrado");
    }
    return new ArrayList<String>();
}
```

```
@Test
void testRecuperarTextoArquivoNãoExiste() {
    assertDoesNotThrow(() -> exemploExceções2.recuperarTexto("arquivo inválido"));
}
```

Testando Exceção JUnit

- No Junit podemos especificar testes que esperam por uma exceção
 - Exemplo

```
@Test
void testParserErradoLançaExceção() throws FileNotFoundException {
    Exception exception = assertThrows(NumberFormatException.class, () -> {
        Integer.parseInt("One");
    });

    assertEquals("For input string: \"One\"", exception.getMessage());
}
```

Tratamento de 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
-

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
-

Informações de uma exceção

- Exemplo

Nome da Exceção

Mensagem

Thread

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException](#): Index 4 out of bounds for length 4
at defensiveProgramming.aula17.ExemplosExceções.exemploComExceçãoNãoCapturada([ExemplosExceções.java:84](#))
at defensiveProgramming.aula17.ExemplosExceções.chamaMetodosComExceção([ExemplosExceções.java:41](#))
at defensiveProgramming.aula17.Main.main([Main.java:21](#))

Stack Trace

Tratamento de 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
 - Código também fica mais limpo
-

Criando Exceções em Java

- Você pode fazer isso estendendo alguma das classes de exceção
 - Ótima opção para encapsular exceções de bibliotecas de terceiros

```
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();  
    }  
}
```

Criando Exceções em Java

- Outro Exemplo

```
class ExcecaoImpar extends Exception {  
    private int x;  
    public ExcecaoImpar() { }  
    public ExcecaoImpar(String msg) {  
        super(msg);  
    }  
    public ExcecaoImpar(int x) {  
        this.x = x;  
    }  
    public String toString() {  
        return "O número " + x + " é ímpar!";  
    }  
}
```

```
public class Teste {  
    public static void imprimePar(int num)  
        throws ExcecaoImpar {  
        if ((num % 2) == 0)  
            System.out.println(num);  
        else  
            throw new ExcecaoImpar(num);  
    }  
    public static void main(String[] args) {  
        try {  
            imprimePar(2);  
            imprimePar(3);  
        } catch (ExcecaoImpar e) {  
            System.out.println(e);  
        }  
    }  
}
```

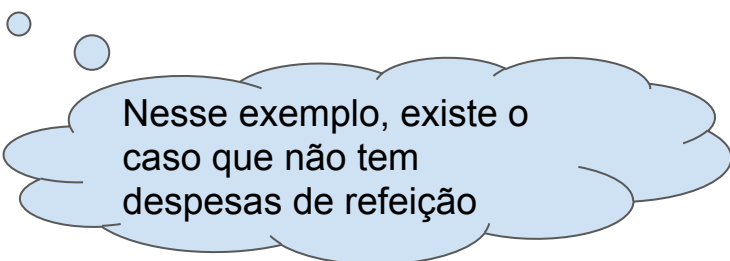

Tratamento de Exceção

- Defina o fluxo normal

- Special Case Pattern

- Criar uma classe ou configurar um objeto que trata um caso especial
- Evita que o código cliente tenha que lidar com comportamento excepcional

```
try {  
    MealExpenses expenses = expenseReportDAO  
                             .getMeals(employee.getID());  
    m_total += expenses.getTotal();  
} catch (MealExpensesNotFound e) {  
    m_total += getMealPerDiem();  
}
```



Nesse exemplo, existe o caso que não tem despesas de refeição

Tratamento de Exceção

- Defina o fluxo normal

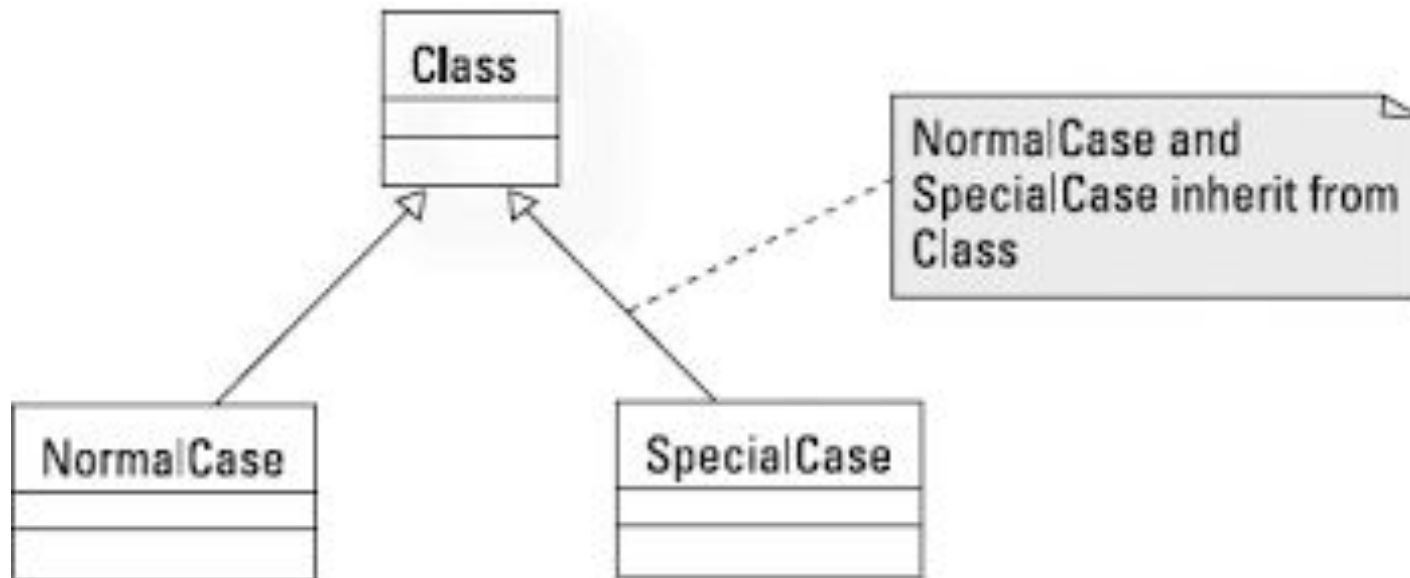
- Nesse exemplo, expenseReportDAO sempre retorna um objeto do tipo MealExpense

```
MealExpenses expenses = expenseReportDAO  
                        .getMeals(employee.getID());  
m_total += expenses.getTotal();
```

O código fica **mais simples**, não existe mais o caso exceptional

Special Case Pattern

- Diz respeito a uma subclasse que fornece um comportamento especial para casos particulares



Special Case Pattern

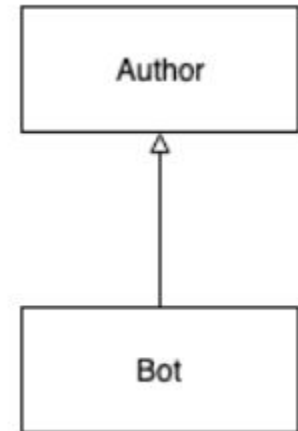
- **Exemplo**
 - Suponha uma aplicação de blog com as funcionalidades
 - create blog post (title, text, authorId, date)
 - listar autores (name, last name, picture)
 - Então o PO solicita uma nova funcionalidade
 - Alguns artigos vão ser criados automaticamente por bots.
 - **Problema**
 - Cada artigo precisa de um authorId e o bot não é um author
-

Special Case Pattern

- Exemplo
 - Uma solução inicial (ruim)
 - Tornar authorId opcional
 - modificamos uma regra importante do domínio (todo artigo tem um autor)
 - vamos precisar verificar se o authorId é null em várias partes do código
 - Outra solução (ruim)
 - Usar um código especial para bot
 - Sem significado para o domínio
 - Implicaria em verificações do código em várias partes
 - ``if ($authorId === BOT_ID)``
-

Special Case Pattern

- Exemplo
 - Usando o Special Case Pattern
 - Criamos uma classe chamada bot que é subclasse de autor
 - Podemos criar no banco de dados um usuário bot com id predefinido
 - Não precisamos mais verificar null
 - todos registros são autores



Evitamos que um caso especial aumentasse a complexidade do código

Tratamento de Exceção

- Não retorne null

- Quando retornamos nulo, precisamos verificar nos métodos clientes do nosso método
 - E uma falta de verificação pode levar a NullPointerException
- Pense em retornar um Objeto Especial (Special Case Pattern)
- Se é uma API de terceiro que retorna nulo, encapsulo o método de forma a lançar uma exceção ou retornar um objeto de caso especial.

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

Tratamento de Exceção

- Não retorne null

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

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

Assim, você minimiza as chances de
NullPointerException e o código fica **mais limpo**

Tratamento de Exceção

- Não passe null

- Passar null para outros métodos é pior ainda!
- Trabalhe com a ideia de que passagem de null como parâmetro é proibido
 - Durante o desenvolvimento você pode utilizar Assertions

```
public double xProjection(Point p1, Point p2) {  
    return (p2.x - p1.x) * 1.5;  
}
```

```
public double xProjection(Point p1, Point p2) {  
    if (p1 == null || p2 == null) {  
        throw IllegalArgumentException ("Invalid argument  
                                     for MetricsCalculator.xProjection");  
    }  
    return (p2.x - p1.x) * 1.5;  
}
```

Tratamento de Exceção

- **Extraia blocks e try/catch**
 - Colocar tudo junto no código principal, mistura o tratamento de erros com o processamento normal
 - Extraia o corpo de try/catch para métodos específicos para isso
-

Tratamento de Exceção

- Extraia blocks e try/catch

focado só no
tratamento de erro

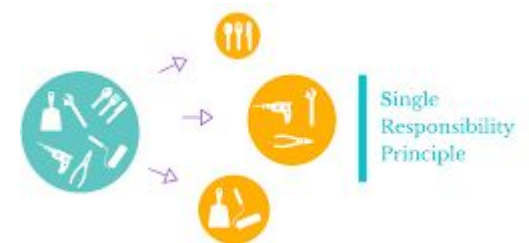
focado só no
processamento de
deletar uma página

```
public void delete(Page page) {  
    try {  
        deletePageAndAllReferences(page);  
    } catch (Exception e) {  
        logError(e);  
    }  
}  
  
private void deletePageAndAllReferences(Page page) throws Exception {  
    deletePage(page);  
    registry.deleteReference(page.name);  
    configKeys.deleteKey(page.name.makeKey());  
}  
  
private void logError(Exception e) {  
    logger.log(e.getMessage());  
}
```

Essa separação de preocupações torna o código fácil de entender e modificar

Tratamento de Exceção

- Tratamento de Erro é UMA coisa
 - Métodos que tratam de erros deveriam fazer só isso
 - Ou seja, esses métodos devem começar com try e não deve ter nada depois dos blocos catch/finally



Tratamento de Exceção

- Lembrando
 - Tratamento de Exceções é Importante, mas
 - É preciso colocá-lo de forma independente da lógica principal
-

Atividade Projeto Final (APF-2)

- **Deadline: 21/02/2021**
 - **Formato**
 - Em grupo, mas temos os requisitos por aluno para pontuação máxima:
 - Mínimo de 10 testes unitários
 - Mínimo de 3 testes funcionais automatizados
 - **Anexar Vídeo**
 - **Anexar informações solicitadas em um arquivo .txt ou .pdf**
 - Quantos casos de testes foram implementados
 - Quais classes com os casos de testes
 - Qual ferramenta usada
-

Obrigado!

Por hoje é só pessoal...

Dúvidas?



qpg4p5x



ismaylesantos@great.ufc.br



@IsmayleSantos
