

**SSC0620 – Engenharia de Software – 1º semestre 2020**

**Profa. Dra. Lina Garcés**

**Estagiário PAE: Gustavo Avellar**

**Colaboradora: Carolina Arenas**

## **Exercício 07**

# **Planejamento de testes unitários com o framework JUnit**

**Alunos:**

**Felipe Barbosa de Oliveira 10276928**

**Paulo Katsuyuki Muraishi Kamimura 10277040**

**Data: 21/05/2020**

<b>1. Abordagem de desenvolvimento</b>	<b>3</b>
<b>2. Planejamento dos testes</b>	<b>3</b>
<b>2.1 Classe Estudante</b>	<b>3</b>
<b>2.2. Classe Disciplina</b>	<b>7</b>
<b>4. Relatar os resultados dos testes</b>	<b>10</b>
<b>5. Causas dos erros e suas devidas soluções</b>	<b>13</b>

## **1. Abordagem de desenvolvimento**

Uma vez o programa não é um projeto muito grande, a abordagem escolhida para o desenvolvimento do programa será primeiro realizar a codificação do programa e depois os testes. Sabendo que existe alguns requisitos e formatações específicas que precisam ser seguidas, por exemplo o formato do código da disciplina, será implementado uma primeira versão da lógica do programa que restringe o formato dos dados e cumpra todos os requisitos e funções necessárias.

A parte da programação inicial se encerrará no momento em que o código estiver compilando e todas as funções desejadas implementadas. Vale ressaltar que, após a finalização da programação inicial não significa que o programa tem seu devido funcionamento idealizado ou todos os requisitos estão sendo cumpridos, então dessa forma o próximo passo será realizar os testes com base nos requisitos do programa.

Foi utilizado Visual Studio Code para a realização desse exercício, a versão do JUnit foi a disponibilizado no Edisciplinas. A versão do JDK do java foi openJDK 11

## **2. Planejamento dos testes**

Para melhor organizar esse documento, a parte do planejamento dos testes será dividido em classes e métodos. No geral, os testes a serem realizado serão elaborados com base nos requisitos descritos no enunciado do exercício e também nas consultas necessárias implementadas. Todas vez que precisava realizar um assertEquals com variáveis booleanas, o valor de referência/comparação era sempre "true", assim, quando o algoritmo falhava o resultado esperado seria uma "failed".

Todas as implementações dos testes foram comentadas. Por inviabilidade de apresentar os códigos neste documento será anexado os códigos dos testes.

### **2.1 Classe Estudante**

A fim de testar a classe estudante, foi criado algumas instâncias de alunos e foi realizado algumas de notas. O cenário utilizado para tal simulação foi a seguinte:

Estudante	Nome	nUSP	Notas
estudanteNUSP_01	Teste01	10277040	-
			-
			-
estudanteNUSP_02	Teste02	123456789	-
			-
			-
estudanteNUSP_03	Teste03	1234	-
			-
			-
estudanteNOTAS	Miguel	1000004	6,00
			-1,00
			12,00

Para cada novo estudante, o programa deve registrar o número USP que contém somente números e comprimento de 7 caracteres. Assim será implementado os seguintes testes que verificará algumas tentativas de adicionar um aluno com os números USP a seguir:

Caso	Entrada	Resultado esperado
Caso Teste 01	10277040	✓
Caso Teste 02	123456789	✗
Caso Teste 03	123	✓

Como o parâmetro do nUSP do construtor de estudante é um int ele já não permite a inserção de letras e outros caracteres, assim, é só preciso verificar se ele tem um comprimento de até 7 caracteres. O teste consiste em tentar retornar o nUSP a partir de um estudante já instanciado e ele verifica se o valor equivale ao valor esperado.

All7

Failed7

EstudanteTest

> casoTeste1

Failed

0.01s

> casoTeste2

Failed

0s

> casoTeste3

Failed

0s

Verificou-se então uma inconsistência, **um erro**. Inicialmente ao tentar inserir um nUSP proibido no construtor o código retornava uma exceção, porém isso

induzia o programa a crashar e interromper sua execução, afetando os resultados dos testes.

```
public Estudante(int nUSP, String nome){
    if(nUSP > 9999999){
        throw new IllegalArgumentException("nUSP deve ter no maximo 7 digitos, porem foi encontrado: "+ nUSP);
        //throw new IllegalArgumentException("nUSP deve ter no maximo 7 digitos, porem foi encontrado: "+ nUSP);
    }
    this.nome = nome;
    this.nUSP = nUSP;
}
```

Dessa forma, foi realizada uma nova forma de validação de nUSP a partir de uma função auxiliar (Vale ressaltar que esse problema de chamar uma exceção também estava presente no construtor da disciplina e portanto o tal erro foi corrigido da mesma forma).

A **solução** portanto foi criar um “construtor” auxiliar para realizar a verificação dos parâmetros e assim, toda vez que alguém precise criar um novo estudante é utilizado a função auxiliar e não o construtor original.

```
private Estudante(int nUSP, String nome){
    this.nome = nome;
    this.nUSP = nUSP;
}

public static Estudante newEstudante(int nUSP, String nome){
    if (nUSP > 99999999) {
        return null; // or throw an Exception - it is how you want
    }
    return new Estudante(nUSP, nome);
}
```

Os novos testes então agora batem com o resultado esperado apresentado na tabela.

All 7	Failed 3	Passed 4
EstudanteTest		
> casoTeste1	Passed	0s
> casoTeste2	Failed	0.02s
> casoTeste3	Passed	0s

As notas para as P1, P2, e P3. As notas para cada prova devem ser números no intervalo [0,10]. Assim será testado a adição de notas não permitidas como notas negativas ou notas maiores que 10.

Caso	Entrada	Resultado esperado
Caso Teste 04	P1 = 6	✓
Caso Teste 05	P2 = -1	✗
Caso Teste 06	P3 = 12	✗

O teste consiste em tentar inserir uma nota e verificar se essa inserção foi realizada com sucesso.

> casoTeste4	Passed	0s
> casoTeste5	Failed	0.02s
> casoTeste6	Failed	0s

Assim, o resultado do teste bateu com o esperado e portanto o código de inserção de nota está funcionando corretamente.

A média final das provas (MFP) de cada aluno utiliza a média aritmética  $(P1+P2+P3)/3$  será testada na classe de Teste “DisciplinaTeste.java” em que será possível realizar a consulta da média de algum estudante específico.

Por fim, o último Teste será verificar se o método getNumero() consegue retornar o nUSP corretamente. Obteve sucesso.

Caso	Parâmetro	Resultado esperado
Caso Teste 07	Verifica se a função retorna o nUSP corretamente	10277040

> casoTeste7	Passed	0s	
--------------	--------	----	--

## 2.2. Classe Disciplina

O código da disciplina deve ter no máximo 7 caracteres de comprimento, sendo que esse código deve começar sempre com as letras “SSC” e os quatro últimos devem conter somente números. (ex. SSC0620).

Para esse requisito será elaborado 3 casos de teste. O primeiro com um tamanho um comprimento maior que 7 caracteres, o segundo caso apesar de ter tamanho 7 possui as iniciais diferentes de “SSC”e, por fim, o terceiro caso uma tentativa de inserir um valor correto.

O critérios utilizados foram os códigos da disciplina inseridos em uma nova instância Disciplina.

Caso	Entrada	Resultado esperado
Caso Teste 01	SSC123456789	✗
Caso Teste 02	ABC4567	✗
Caso Teste 03	SSC0620	✓

All 3	Failed 2	Passed 1
DisciplinaTest		
> casoTeste01	Failed	0.01s
> casoTeste02	Failed	0s
> casoTeste03	Passed	0s

Os testes obtiveram sucesso, vale ressaltar que um problema percebido nos testes de estudante já foi corrigido em disciplina, então esse teste foi realizado após a implementação da solução.

O programa deve permitir adicionar novos estudantes na disciplina assim foi implementada uma lista(Arraylist) de estudantes em cada disciplina. Uma vez que será necessário analisar as consultas de alunos matriculados em uma disciplina um teste para verificar a correta adição de um novo estudante não será necessário por que ao verificar corretamente qualquer consulta a devida inserção está sendo indiretamente verificada.











Foi criado uma situação na classe de teste com as seguintes estradas:

Estudante	Nome	nUSP	Notas	Médias
alunoAprovado1	Aprovado	1000001	10,00	9,10
			9,50	
			7,80	
alunoAprovado2	Aprovado	1000002	4,00	5,00
			5,00	
			6,00	
alunoAprovado3	Aprovado	1000003	5,00	5,00
			5,00	
			5,00	
alunoReprovado1	Reprovado	1000004	2,00	2,33
			3,00	
			2,00	
alunoReprovado2	Reprovado	1000005	4,10	4,43
			5,30	
			3,90	

Assim essa é a nossa “base de dados” para fazer a realização de todos os casos de testes seguintes. O nomes dos estudantes foram padronizados como “Aprovado” e “Reprovado” para facilitar a verificação de alunos aprovados, uma vez sabendo que na inserção os alunos com médias maiores que 5 seu nome é “Aprovado”.



Caso	Parâmetro	Resultado esperado
<b>Caso Teste 04</b>	Verificar quantidade de alunos matriculados na disciplina disciplinaGlobal	5,00
<b>Caso Teste 05</b>	Verificar quantidade de alunos aprovados na disciplina disciplinaGlobal	3,00
<b>Caso Teste 06</b>	Verificar quantidade de alunos reprovados na disciplina disciplinaGlobal	2,00
<b>Caso Teste 07</b>	Nota 1 alunoReprovado2 -Nota valor quebrado(float)	4,10
<b>Caso Teste 08</b>	Nota 2 alunoReprovado2 -Nota valor quebrado(float)	5,30
<b>Caso Teste 09</b>	Nota 3 alunoReprovado2 -Nota valor quebrado(float)	3,90
<b>Caso Teste 10</b>	Verificar média final de alunoAprovado1	9,10
<b>Caso Teste 11</b>	Verificar ordenação descendentes dos alunos	-
<b>Caso Teste 12</b>	Verificar se a lista de aprovados contém somente alunos com média > 5	-
<b>Caso Teste 13</b>	Verificar se a lista de reprovados contém somente alunos com média < 5	-

> casoTeste04	Passed	0s	
> casoTeste05	Passed	0s	
> casoTeste06	Passed	0s	
> casoTeste07	Passed	0s	
> casoTeste08	Passed	0s	
> casoTeste09	Passed	0s	
> casoTeste10	Passed	0s	
> casoTeste11	Passed	0s	
> casoTeste12	Passed	0.01s	
> casoTeste13	Passed	0.01s	

Obteve sucesso em todas as consultas.

#### 4. Relatar os resultados dos testes

##### Estudante














All	7	Failed	3	Passed	4
EstudanteTest					
>	casoTeste1	Passed	0s		
>	casoTeste2	Failed	0.02s		
>	casoTeste3	Passed	0s		
>	casoTeste4	Passed	0s		
>	casoTeste5	Failed	0.01s		
>	casoTeste6	Failed	0s		
>	casoTeste7	Passed	0.01s		

- Teste 1: testa se o nUSP tem no máximo 7 dígitos. A primeira versão do código **continha erros**, foi **corrigido** então e posteriormente realizado novos testes(**Sucesso pois o nUSP cadastrado é válido**);
- Teste 2: testa se o nUSP tem no máximo 7 dígitos. A primeira versão do código **continha erros**, foi **corrigido** então e posteriormente realizado novos testes(**Sucesso pois o nUSP cadastrado é inválido**);
- Teste 3: testa se o nUSP tem no máximo 7 dígitos. A primeira versão do código **continha erros**, foi **corrigido** então e posteriormente realizado novos testes(**Sucesso pois o nUSP cadastrado é válido**);
- Teste 4: testa se a nota 1 cadastrada está entre 0 e 10. Obteve **sucesso** pois a nota cadastrada era **válida**.
- Teste 5: testa se a nota 2 cadastrada está entre 0 e 10. Obteve **sucesso** pois a nota cadastrada era **inválida**.
- Teste 6: testa se a nota 3 cadastrada está entre 0 e 10. Obteve **sucesso** pois a nota cadastrada era **inválida**.
- Teste 7: testa se o código retorna o nUSP corretamente. Obteve **sucesso**.

### Disciplina

Os testes 1 e 2 obtiveram resultado de falha, **como esperado(Sucesso)**. Foram criados para analisar como o programa reagiria para uma entrada não permitida, e foi percebido que o programa não permitia que códigos inválidos fossem registrados.

Os testes de 3 a 13 foram realizados para avaliar o funcionamento normal do programa.

DisciplinaTest			
> casoTeste01	Failed	0.01s	
> casoTeste02	Failed	0s	
> casoTeste03	Passed	0s	
> casoTeste04	Passed	0s	
> casoTeste05	Passed	0s	
> casoTeste06	Passed	0s	
> casoTeste07	Passed	0s	
> casoTeste08	Passed	0s	
> casoTeste09	Passed	0s	
> casoTeste10	Passed	0s	
> casoTeste11	Passed	0s	
> casoTeste12	Passed	0.01s	
> casoTeste13	Passed	0.01s	

- Teste 3: testa se o programa permite a inserção de um código válido, e foi avaliado que o programa permite, obtendo **sucesso**;
- Teste 4: testa se o programa calcula a quantidade de alunos matriculados de maneira correta, e o **resultado obtido foi igual** ao número de alunos cadastrados;
- Teste 5: testa se o programa calcula a quantidade de alunos aprovados de maneira correta, e o **resultado obtido foi igual** ao número de alunos cadastrados com média maior ou igual a 5;
- Teste 6: testa se o programa calcula a quantidade de alunos reprovados de maneira correta, e o **resultado obtido foi igual** ao número de alunos cadastrados com média menor que 5;
- Teste 7: testa se o programa retorna a nota 1 de um aluno corretamente, obtendo um **resultado igual** ao cadastrado;
- Teste 8: testa se o programa retorna a nota 2 de um aluno corretamente, obtendo um **resultado igual** ao cadastrado;
- Teste 9: testa se o programa retorna a nota 3 de um aluno corretamente, obtendo um **resultado igual** ao cadastrado;
- Teste 10: testa se o programa calcula a média de um aluno corretamente. O resultado obtido inicialmente não era exatamente igual, pois o Java estava tendo um erro de aproximação (o que deveria ser 9.1 estava 9.099). Com isso, colocamos uma margem de erro de 0.01 na função assert, fazendo com que o **teste fosse sucedido**;
- Teste 11: testa se o programa ordenou os alunos de maneira decrescente corretamente. O teste percorre a lista de alunos e checka se o número USP de um aluno é maior que o anterior. Se sim, quer dizer que houve alguma falha. O **teste obteve sucesso**;
- Teste 12: testa se a lista de aprovados calculada realmente só possui alunos com média maior ou igual a 5. O teste percorre cada aluno e checka sua média. O **teste foi sucedido**;
- Teste 13: testa se a lista de reprovados calculada realmente só possui alunos com média menor que 5. O teste percorre cada aluno e checka sua média. O **teste foi sucedido**.

## **5. Causas dos erros e suas devidas soluções**

### **Estudante**

Os erros encontrados foram relacionados aos testes 1, 2 e 3. O erro consiste em colocar um `IllegalArgumentException` no construtor do `Estudante`, induzindo ao programa e inclusive aos teste interromper sua execução(inclusive dos testes). Caso o número tivesse mais de 7 dígitos, era lançada uma `IllegalArgumentException`, de modo que o código é parado. Para corrigir tal problema, foi criado uma função construtor auxiliar para realizar a verificação do nUSP, caso seja inserido um nUSP proibido o novo “Estudante” não é instanciado com sucesso.

### **Disciplina**

Para os testes 1 2, e 3 foi implementada a mesma solução do construtor de estudante, assim criado um construtor auxiliar com checagem da String do código da disciplina para impedir a criação de uma inválida.

Os outros testes não eram suscetíveis à falhas, pois simplesmente chamam funções do estudante.