

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL SÃO GABRIEL
Bacharelado em Sistemas de Informação

Arthur Pimenta Braga
Frederico Henrique Pires da Silva
Gabriel de Oliveira Fávero
Pedro Henrique Faria Andrade
Roselmiriam Rodrigues dos Santos

TRABALHO PRÁTICO 2

Belo Horizonte
2022

1) Descrição do Software

Para a realização dos testes previstos no trabalho prático foi escolhido o software “Agência de Turismo”. Esse é um programa na linguagem Java, desenvolvido por integrantes do grupo durante a disciplina de “Programação Orientada a Objetos”, do terceiro período do curso.

O software consiste em um recurso para gerir uma agência de turismo fictícia. Ele permite adicionar clientes e produtos nos dados do programa, vincular um produto a um cliente, localizar e listar produtos e clientes.

As entidades do programa são a Agência e o Cliente. Existem diferentes tipos de clientes, em que cada tipo possui um desconto, pontuação e pontuação mínima distintos. Os principais produtos dessa agência de turismo são a hospedagem, a compra de passagens aéreas e a realização de passeios.

Toda a estrutura foi organizada em diversas classes e existe uma estrutura de dados em .txt. Para os arquivos em texto serem lidos com sucesso tanto nos testes quanto no código, é preciso abrir o código a partir da pasta “Code” para ela ser a pasta raiz do projeto

2) Framework para criação dos casos de teste

O framework escolhido para a realização dos casos de teste foi o *JUnit*. Ele é um software open-source, destinado para a linguagem Java e que tem como função a automação de testes unitários de software.

Os testes possuem o intuito de otimizar o processo de identificação de erros no sistema. Dessa forma, é possível identificar diversos itens, como o incremento de valores incorretos, cálculos e cenários inválidos, instâncias com valores nulos e vazios e a inicialização de variáveis.

Para a realização dos testes, teve-se como objetivo cobrir o máximo possível do código. Foram pensados diversos cenários distintos para cada método, tendo em vista todas as condições e resultados que cada método poderia ter como retorno.

Imagem 1 - Exemplo de caso de teste utilizando JUnit

```
@Test
@DisplayName("Para nome e cpf preenchidos, adicionarCliente deve retornar o novo cliente")
void test06() {
    var output : Cliente = Agencia.adicionarCliente( nome: "José Silva", cpf: "123.123.123-01");
    var expected : Cliente = Agencia.localizarCliente( cpf: "123.123.123-01");

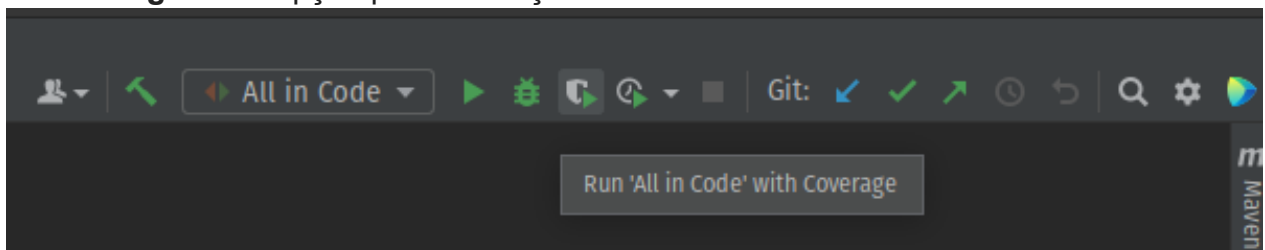
    assertNotNull(expected);
    assertEquals(expected.getCpf(), output.getCpf());
    assertEquals(expected.getNome(), output.getNome());
}
```

Fonte: Elaborado pelo grupo

3) Software utilizado para o cálculo da cobertura de testes

Tendo em vista que o *IntelliJ IDEA* foi o ambiente de desenvolvimento selecionado para desenvolvimento do software, a análise e o cálculo da cobertura de testes foi feita a partir de um recurso próprio disponibilizado e recomendado pela IDE. Para executar a cobertura, é preciso selecionar a opção “Run [Teste] with Coverage’ ” localizada no menu horizontal da IDEA. Com isso, são realizados cálculos para determinar a quantidade de classes, métodos e linhas no código.

Imagem 2 - Opção para execução da análise de cobertura mencionada.

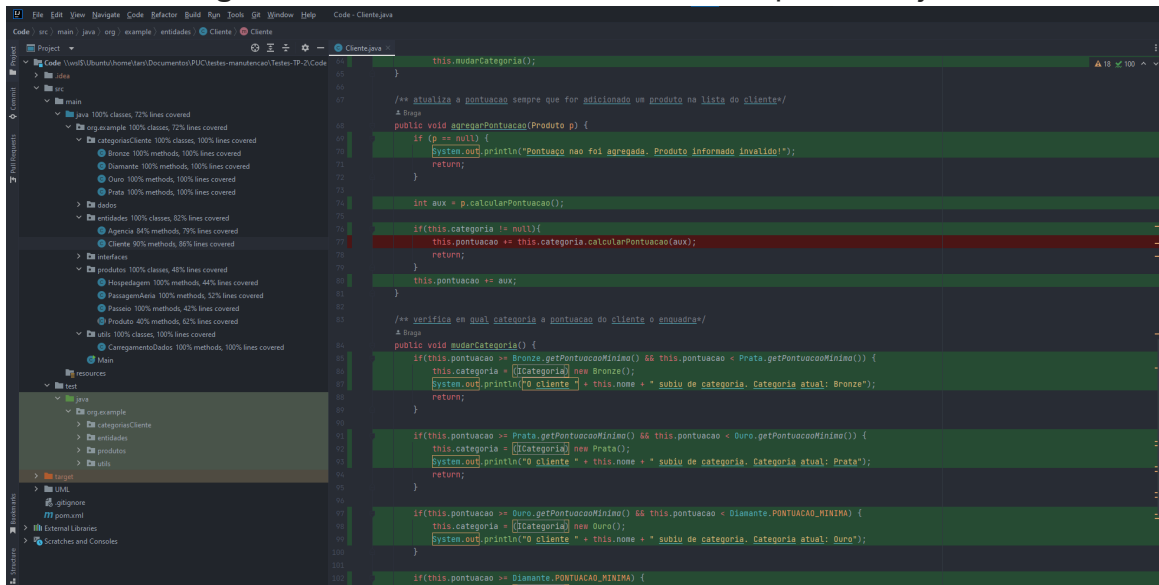


Fonte: Elaborado pelo grupo

A documentação do IntelliJ não apresenta nenhuma métrica ou cálculo explícito de como o resultado de cobertura é atingido. Contudo, após uma breve análise do grupo, foi constatado que para o cálculo é realizada uma divisão e esse resultado é multiplicado por 100 para chegar à porcentagem. Segue abaixo um exemplo do cálculo de cobertura do software por linhas.

$$Cobertura = \frac{Linhas\ testadas}{Linhas\ totais} \times 100$$

Imagem 3 - Análise de cobertura realizada pelo IntelliJ IDEA

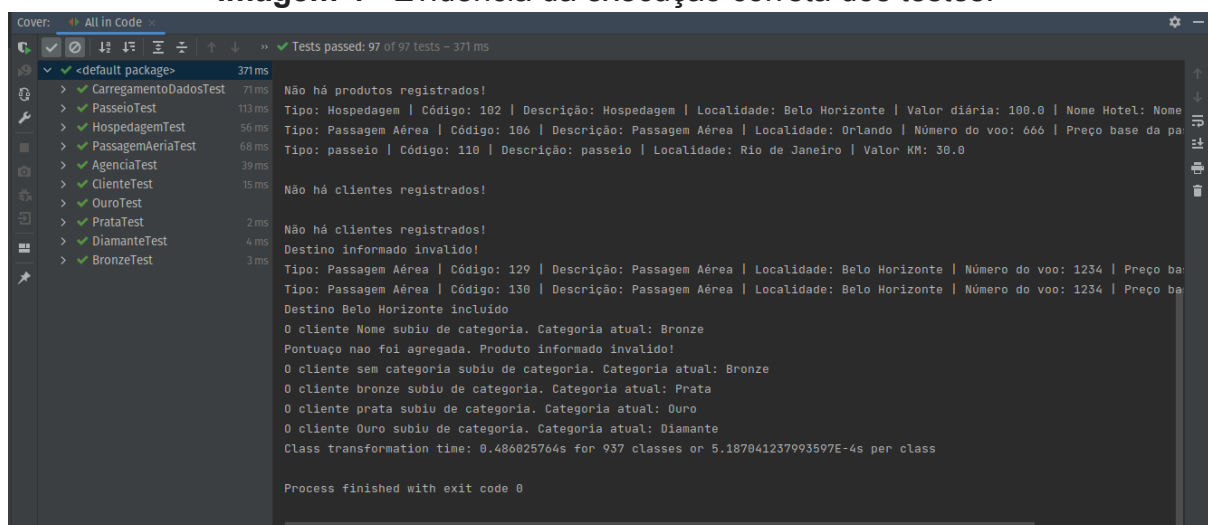


Fonte: [Documentação da JetBrains](#)

4) Casos de Teste e Resultados

Para a análise e cálculo da cobertura de testes, foram realizados 97 casos de teste. Ao todo, houve cobertura em 11/11 classes, 82/91 métodos e 237/326 linhas. A classe 'main' não foi considerada para a realização de testes e para o cálculo de cobertura pelo fato de somente invocar métodos de classes previamente testadas. Além disso, os métodos presentes são constituídos somente de exibição/captura de informações no terminal, assim não havendo o que se testar.

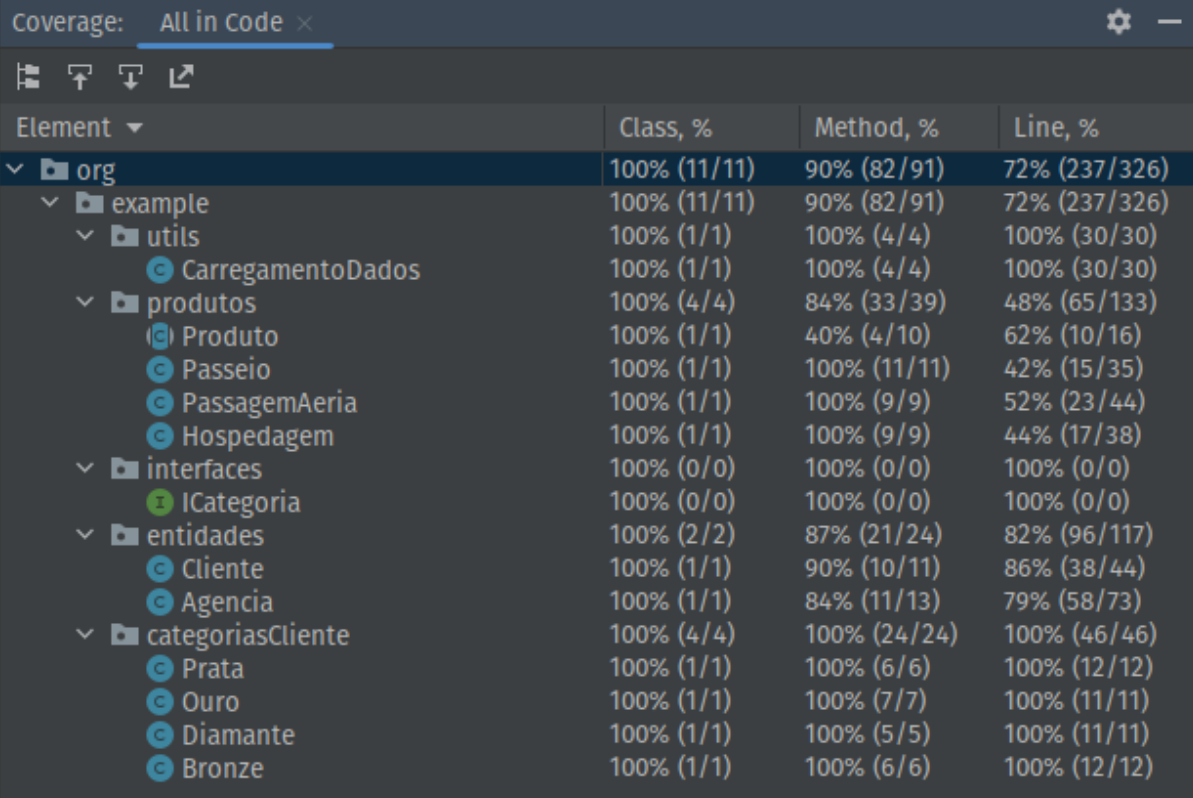
Imagem 4 - Evidência da execução correta dos testes.



Fonte: Elaborado pelo grupo

Com isso, é possível verificar que a taxa de cobertura final foi de 72%, com 237 das 326 linhas testadas. Além disso, 100% das classes e 90% dos métodos tiveram no mínimo um teste os englobando. É importante destacar que a presença dos testes não garantem que a aplicação está livre de erros, porém auxilia imensamente na entrega de um programa menos suscetível a erros para o cliente final.

Imagem 5 - Taxa de cobertura final



Element ▾	Class, %	Method, %	Line, %
▼ org	100% (11/11)	90% (82/91)	72% (237/326)
▼ example	100% (11/11)	90% (82/91)	72% (237/326)
▼ utils	100% (1/1)	100% (4/4)	100% (30/30)
CarregamentoDados	100% (1/1)	100% (4/4)	100% (30/30)
▼ produtos	100% (4/4)	84% (33/39)	48% (65/133)
Produto	100% (1/1)	40% (4/10)	62% (10/16)
Passeio	100% (1/1)	100% (11/11)	42% (15/35)
PassagemAeria	100% (1/1)	100% (9/9)	52% (23/44)
Hospedagem	100% (1/1)	100% (9/9)	44% (17/38)
▼ interfaces	100% (0/0)	100% (0/0)	100% (0/0)
ICategoria	100% (0/0)	100% (0/0)	100% (0/0)
▼ entidades	100% (2/2)	87% (21/24)	82% (96/117)
Cliente	100% (1/1)	90% (10/11)	86% (38/44)
Agencia	100% (1/1)	84% (11/13)	79% (58/73)
▼ categoriasCliente	100% (4/4)	100% (24/24)	100% (46/46)
Prata	100% (1/1)	100% (6/6)	100% (12/12)
Ouro	100% (1/1)	100% (7/7)	100% (11/11)
Diamante	100% (1/1)	100% (5/5)	100% (11/11)
Bronze	100% (1/1)	100% (6/6)	100% (12/12)

Fonte: Elaborado pelo grupo