

Relatório de Projeto

Ferramentas de Teste

José Adeljan Marinho da Silva - 077.073.694-75

Projeto - Banco





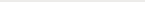
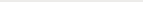


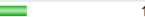
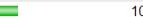
Com base no projeto anexado a este post, realizar as seguintes tarefas:

1. Escrever testes em JUnit e encontrar todos os bugs que existem na implementação atual;
2. Garantir 100% de cobertura de branch;
3. Matar todos os mutantes gerados pela ferramenta PIT.

O projeto proposto envolve a realização de testes em JUnit para avaliar e melhorar a qualidade da implementação atual do código-fonte de uma aplicação de Banco. Para atender a esse objetivo, foi requisitada a criação testes que abranjam todos os casos de uso possíveis, incluindo cenários normais e excepcionais. Além disso, é necessário realizar testes que atinjam 100% de cobertura de branch, ou seja, todos os caminhos possíveis no código devem ser percorridos pelos testes. Por fim, para testar a qualidade dos testes, a ferramenta PIT é usada para gerar mutantes e os testes devem ser elaborados de forma a matar todos esses mutantes.

Os testes implementados atingiram 100% de cobertura de *branch* para as três classes. Inicialmente os testes foram realizados utilizando a classe “Fachada”, que instancia as outras classes presentes do código-fonte. Com essa classe, foi atingida a cobertura de 100% de *branches* e foram realizados testes individualmente para os outros métodos das classes “Clientes” e “Conta” não cobertos na classe “Fachada”. O relatório JaCoCo de cobertura de branches pode ser visto na imagem abaixo.

app

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
br.ufpe.cin.residencia.banco.conta		99%		100%	1	67	2	154	1	52	0	9
br.ufpe.cin.residencia.banco.excecoes		85%		n/a	1	7	2	14	1	7	1	7
br.ufpe.cin.residencia.banco		0%		n/a	2	2	2	2	2	2	1	1
br.ufpe.cin.residencia.banco.cliente		100%		100%	0	39	0	80	0	28	0	5
br.ufpe.cin.residencia.banco.fachada		100%		100%	0	17	0	36	0	14	0	1
Total	13 of 967	98%	0 of 58	100%	4	132	6	286	4	103	2	23

A realização dos testes revelou *bugs* relativos às funcionalidades do código-fonte, como métodos não-implementados e métodos implementados erroneamente. Um exemplo é o método “existe” dos repositórios RCMMap e RCArray. Esta função tinha um retorno “false” e, dessa forma, os testes nunca conseguiam alcançar 100% de cobertura de linha, pois, não correspondia a todos os casos de condição. Por esse motivo, os métodos de cadastro dos repositórios RCMMap e RCArray nunca funcionavam, pois os testes não alcançavam as branches dependentes do método “existe”. Outro exemplo é o método “descadastrar” da classe “CadastroClientes” que não estava implementado, assim, o teste sempre levantava a exceção “ClientelInexistente” para qualquer um dos casos de teste que utilizassem o cliente descadastrado a partir deste método.

Na classe Fachada, no método `cadastrar(ContaAbstrata c)`, há uma verificação para evitar que o cliente (`cli`) associado à conta (`c`) seja nulo, o que é bom para prevenir uma `ClientelInexistenteException`. No entanto, seria prudente adicionar uma verificação para tratar a possibilidade de `c` em si ser nulo, evitando assim um potencial null pointer exception.

Na classe ContaBonificada, o campo `bonus` é privado, mas não há um método getter público para acessar seu valor. Isso pode ser limitante se for necessário recuperar o valor do bônus de fora da classe.

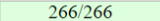

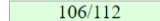
Na classe Poupança, o método `renderJuros` calcula e credita os juros na conta, mas não lida com a situação em que o cálculo dos juros resulta em um valor negativo. Seria apropriado adicionar uma validação para garantir que o saldo da conta nunca seja definido como negativo.

Na classe ContaImposto, ao calcular o imposto no método `debitar`, podem ocorrer problemas de precisão devido ao uso da aritmética de ponto flutuante. É uma boa prática utilizar `BigDecimal` para cálculos financeiros a fim de evitar problemas de precisão com números de ponto flutuante. Além disso, o método `existe` não está implementado e sempre retorna `false`, o que deve ser corrigido para verificar a existência de um número de conta no repositório. Para outras mudanças nos métodos do código do Banco, é possível consultar o GitHub.

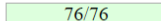
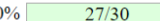
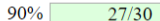
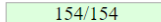
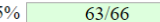
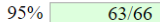
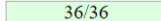
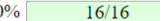
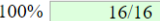
Relativo aos mutantes gerados pela ferramenta PIT, foi possível matar aproximadamente 95% dos gerados. A imagem abaixo ilustra o quadro geral de geração/eliminação de mutantes. Para a implementação deste código, foram gerados um total de 112 mutantes, dos quais 106 foram mortos. Adiante serão discutidos os resultados para cada uma das classes que sofreram as mutações.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
14	100% 	95% 	95% 

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
br.ufpe.cin.residencia.banco.cliente	4	100% 	90% 	90% 
br.ufpe.cin.residencia.banco.conta	9	100% 	95% 	95% 
br.ufpe.cin.residencia.banco.fachada	1	100% 	100% 	100% 

Para o pacote “Fachada”, 100% dos mutantes foram mortos pelos testes, como representado na imagem abaixo.

Pit Test Coverage Report

Package Summary

br.ufpe.cin.residencia.banco.fachada

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	100% <div>36/36</div>	100% <div>16/16</div>	100% <div>16/16</div>

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
Fachada.java	100% <div>36/36</div>	100% <div>16/16</div>	100% <div>16/16</div>

Para o pacote “Cliente”, 90% dos mutantes foram mortos, como representado na imagem abaixo.

Pit Test Coverage Report

Package Summary

br.ufpe.cin.residencia.banco.cliente

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
4	100% <div>76/76</div>	90% <div>27/30</div>	90% <div>27/30</div>

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
CadastroClientes.java	100% <div>13/13</div>	100% <div>5/5</div>	100% <div>5/5</div>
Cliente.java	100% <div>14/14</div>	100% <div>3/3</div>	100% <div>3/3</div>
RepositorioClientesArray.java	100% <div>30/30</div>	81% <div>13/16</div>	81% <div>13/16</div>
RepositorioClientesMap.java	100% <div>19/19</div>	100% <div>6/6</div>	100% <div>6/6</div>

A classe com mutantes sobreviventes foi a “RepositórioClientesArray” onde as mutações implementadas envolvem a troca de sinais no método de remover clientes. Esta mutação não conseguiu ser abrangida pelos testes. Não consegui identificar se é uma falha dos testes em si ou se é necessário uma melhor implementação do código. As mutações sobreviventes são apresentadas na imagem abaixo.

Mutations

23	1. negated conditional → KILLED
32	1. negated conditional → KILLED
32	2. replaced boolean return with true for br/ufpe/cin/residencia/banco/cliente/RepositorioClientesArray::existe → KILLED
38	1. Replaced integer addition with subtraction → KILLED
44	1. negated conditional → KILLED
45	1. replaced return value with null for br/ufpe/cin/residencia/banco/cliente/RepositorioClientesArray::procurar → KILLED
53	1. changed conditional boundary → KILLED
53	2. negated conditional → KILLED
54	1. negated conditional → KILLED
55	1. replaced int return with 0 for br/ufpe/cin/residencia/banco/cliente/RepositorioClientesArray::procurarIndice → KILLED
58	1. replaced int return with 0 for br/ufpe/cin/residencia/banco/cliente/RepositorioClientesArray::procurarIndice → KILLED
64	1. negated conditional → KILLED
66	1. Replaced integer subtraction with addition → SURVIVED
67	1. Replaced integer subtraction with addition → SURVIVED
68	1. Replaced integer subtraction with addition → KILLED
76	1. replaced return value with Collections.emptyList for br/ufpe/cin/residencia/banco/cliente/RepositorioClientesArray::listar → SURVIVED

Para o pacote “Conta”, 95% dos mutantes foram mortos, como representado na imagem abaixo. Similar ao pacote “Cliente”, no pacote “Conta” as mutações não abrangidas envolvem a classe “RepositorioContasArray”, sendo estas mutações semelhantes ao caso do pacote “Cliente”. Neste caso também não consegui identificar se é necessária mudança na lógica de programação ou se foi defeito dos testes implementados.

Pit Test Coverage Report

Package Summary

br.ufpe.cin.residencia.banco.conta

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
9	100% 154/154	95% 63/66	95% 63/66

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
CadastroContas.java	100% 23/23	100% 9/9	100% 9/9
Conta.java	100% 8/8	100% 4/4	100% 4/4
ContaAbstrata.java	100% 27/27	100% 14/14	100% 14/14
ContaBonificada.java	100% 11/11	100% 5/5	100% 5/5
ContaEspecial.java	100% 8/8	100% 2/2	100% 2/2
ContaImposto.java	100% 10/10	83% 5/6	83% 5/6
Poupanca.java	100% 7/7	100% 2/2	100% 2/2
RepositorioContasArray.java	100% 42/42	89% 16/18	89% 16/18
RepositorioContasMap.java	100% 18/18	100% 6/6	100% 6/6

Em relação à classe “ContaImposto”, o mutante gerado no método “debitar” não foi morto. Semelhante aos casos anteriores, também não consegui identificar se é necessária mudança na lógica da implementação ou se foi defeito dos testes implementados. As mutações sobreviventes são apresentadas na imagem abaixo.

Mutations

17	1. Replaced double multiplication with division → KILLED
18	1. Replaced double addition with subtraction → KILLED
19	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
20	1. Replaced double subtraction with addition → KILLED 2. removed call to br/ufpe/cin/residencia/banco/conta/ContaImposto::setSaldo → KILLED

Quanto à experiência em testes, posso afirmar que foi bastante edificante, uma vez que a área de testes é de grande interesse pessoal. No entanto, enfrentei desafios significativos durante a execução deste projeto, principalmente devido ao uso da linguagem de programação Java, com a qual não tinha uma ampla familiaridade. Além disso, não possuo formação prévia em programação, e meu primeiro contato com Java ocorreu durante o curso de residência. Apesar de ter conseguido identificar muitos dos problemas no código, minha falta de experiência em Java limitou minha capacidade de implementar algumas das lógicas necessárias para corrigir esses problemas. Não obstante, essa experiência como testador foi valiosa e se assemelhou às situações de testes reais com as quais trabalhamos no curso até então.