Relatório do trabalho de Qualidade e Teste de Software - 2025.2 (Parte 2)

Membros: Daniel Fontoura, Daniele Pimenta, Irhael Chagas e João Gabriel Otogali

1. Introdução

Dando continuidade ao trabalho prático, para esta segunda entrega, foi proposto diversas tarefas com o intuito de aplicar o que foi aprendido em sala de aula.

Para facilitar a identificação, quando necessário, cada tópico terá um subtópico separado para cada integrante do grupo, detalhando o que foi feito por cada um.

2. Ferramentas utilizadas

Nesta segunda entrega, acrescentamos as ferramentas:

- PIT (Mutation Testing)
- SonarQube
- System Lambda (Teste de Sistema)

3. Casos de Teste com Mockito

Nesta etapa, foi preciso melhorar e aumentar os casos de teste unitários, isolando dependências. Para isso, utilizamos o Mockito, cujo objetivo é isolar comportamentos específicos e simular dependências a fim de garantir que o método sob teste funcione corretamente sem depender das outras classes reais. Com isso, é possível fazer simulações de cenários reais de execução.

3.1 - Daniel Fontoura

Além dos cenários de teste já criados na primeira entrega, foi incluído também os cenários a seguir, a fim de explorar a maior quantidade possível de casos. A abordagem explorada foi incluir **valores de fronteira**:

ID do cenário	Método	Descrição	Resultado esperado
1.3.2-S	`dadosTransacao`	Depósito Premium (Soma Excede).	Valor inválido
1.3.3-S	`dadosTransacao`	Depósito Diamond (Soma Excede).	Valor inválido
1.4.1	`dadosTransacao`	Tentar realizar depósito com valor zero.	Retornar false
1.4.2	`dadosTransacao`	Tentar realizar depósito com valor negativo.	Retornar false

1.4.3	dadosTransacao	Tentar realizar transferência com valor zero.	Retornar false
2.3.1	valorFatura	Tentar pagar fatura com valor zero.	Retornar false
2.3.2	valorFatura	Tentar pagar o valor exato da fatura.	Retornar true
2.3.3	valorFatura	Tentar aumentar limite com valor exato disponível.	Retornar false (Comportamento de borda do método)
2.4	valorFatura	Valor Fatura com operação desconhecida	Retornar true
4.1	verificarBoleto	Validar criação de boleto com dados válidos.	Retornar true
4.2	verificarBoleto	Validar criação de boleto com valor negativo.	Lançar ValorInvalido
4.3	verificarBoleto	Validar criação de boleto com multa negativa.	Lançar ValorInvalido
4.4	verificarBoleto	Validar criação de boleto com data lógica inválida	Retornar false
4.5	verificarBoleto	Validar criação de boleto com multa não numérica.	Retornar false
4.6	verificarBoleto	Validar criação de boleto com valor zero.	Retornar true
4.7	verificarBoleto	Validar criação de boleto com multa zero.	Retornar true
4.8	verificarBoleto	Validar criação de boleto com texto de data inválido.	Lançar ValorInvalido

3.2 - Daniele Pimenta

Dando prosseguimento ao estudo de Mockito para a classe que utilizei como base para os estudos (Conta.java), foram criados os seguintes testes:

Teste	Objetivo	Cenário	Resultado esperado
pagarBoleto_ComSucesso()	Validar pagamento de boleto com saldo suficiente.	Conta criada com R$500,00 de saldo. Mock de boleto com valor R$200,00, sem multa.	O saldo deve ser reduzido corretamente após o pagamento.
pagarBoleto_SaldoInsuficiente_LancarExcecao()	Garantir que o pagamento não ocorra quando o saldo é insuficiente	Conta com R$100,00 e boleto simulado de R$250,00.	Lançar TransacaoException e não alterar o saldo da conta.
transferir_ComDadosMockados_ComSucesso()	Validar a transferência utilizando mock de método estático (InterfaceUsuario.getDadosTransacao()).	2 contas reais criadas (origem e destino). MockedStatic usado para simular dados da transação. Conta origem com R$500,00. Valor de transferência: R$200,00.	Conta de origem deve ter saldo reduzido; conta de destino deve receber o valor.
pagarBoleto_MultaDeAtraso()	Validar cálculo de multa ao pagar boleto atrasado.	Conta com R$1000 de saldo; boleto mockado com valor R$100 e multa diária R$10; atraso de 2 dias.	Saldo deve diminuir 100 + 20 = 120. Totalizando R$880,00
modificarChavePix_DeveRetornarTrue()	Validar modificação/adição de chave Pix com mocks.	Spy de conta, mock de DadosChavesPix, mock de ChavePix, mock de chamada estática getDadosChavePix().	Método deve retornar true.
depositar_DeveAumentarSaldo()	Validar comportamento de depósito usando mocks.	Conta com saldo 0, mock de DadosTransacao retornando valor 200 para origem e destino iguais.	Saldo final deve ser 200.
agendarTransacao_ComSucesso()	Validar agendamento de transação com mocks estáticos.	Spy de conta; mocks de DadosTransacao, Data, Transacao.criarTransacaoAgendada; addTransacaoAgendadas forçado a true.	Método deve retornar a transação mockada.
getLimiteMaximo_ComCartao_DeveRetornarValor()	Validar retorno de limite máximo.	GerenciamentoCartao recebe cartão mock com limite 1000.	Deve retornar 1000

3.3 - Irhael Chagas

Diferente de outros módulos do sistema que interagem diretamente com o console (exigindo o uso de Mocks para simular a entrada do usuário), a classe VerificadorClientes atua como uma unidade lógica pura de validação. Ela recebe dados brutos e aplica regras de negócio determinísticas.

Por esse motivo, optou-se pela implementação de testes unitários diretos utilizando JUnit 5, garantindo maior fidelidade e simplicidade na verificação das regras, sem a necessidade de simular comportamentos externos.

ID	Método de Teste	Objetivo / Cenário	Resultado Esperado
3.3.1	verificarEndereco	Validação de CEP numérico com 8 dígitos.	Retornar true.
3.3.2	verificarEndereco	Tentativa de validação de CEP contendo letras ou tamanho inválido.	Retornar false (ou tratar exceção interna).
3.3.3	verificarIdade	Teste de Fronteira: Pessoa Física com 17 anos.	Retornar false (Bloqueio).
3.3.4	verificarIdade	Teste de Fronteira: Pessoa Física com 18 anos.	Retornar true (Aprovação).
3.3.5	informacoesClientes	Validação de campos obrigatórios em branco.	Lançar exceção ValorInvalido.
3.3.6	informacoesClientes	Validação de segurança (Senha fraca ou nome com números).	Lançar exceção ValorInvalido.

3.4 - João Otogali

ID	Método de Teste	Objetivo / Cenário	Resultado Esperado
1.1.1	tipoChavePix	Validar entrada válida "EMAIL".	Retornar true.
1.1.2	tipoChavePix	Validar entrada válida "TELEFONE".	Retornar true.

1.1.3	tipoChavePix	Validar entrada inválida "TIPO_INVALIDO".	Retornar false.
1.1.4	tipoChavePix	Robustez: Validar entrada null.	Retornar false.
1.2.1	chavePix	Interação: Usuário digita "0" (Cancelar).	Retornar true.
1.3.1	chavePix	Lógica: Email Válido (Confirmado pelo Mock).	Retornar false (Sucesso).
1.3.2	chavePix	Lógica: Email Inválido (Rejeitado pelo Mock).	Retornar true (Erro).
1.3.3	chavePix	Lógica: Telefone Válido (Confirmado pelo Mock).	Retornar false.
1.3.4	chavePix	Lógica: Chave Aleatória Válida (Tamanho correto).	Retornar false.
1.3.5	chavePix	Lógica: Chave Aleatória Inválida (Tamanho errado).	Retornar true.
1.3.6	chavePix	Fluxo: Tipo de chave desconhecido (Default).	Retornar true.
2.1	chavePix	UX: Confirmação implícita (Usuário digita algo != "0").	Seguir para validação.
3.1	chavePix	Integração Mockada: Validação de Email.	Chamar VerificadorClientes.
3.2	chavePix	Integração Mockada: Validação de CPF/Identificação.	Chamar VerificadorClientes.
4.1	chavePix	Robustez: Chave Aleatória null (Crash Test).	Tratar erro e retornar true.
4.2	chavePix	Bordas: Chave vazia ou tamanho incorreto.	Retornar true.

4.3	chavePix	Robustez: Tipo de Chave null (Switch).	Lançar Exceção Controlada.

4. Casos de Teste de Integração

Os testes de Integração são feitos após os testes unitários e têm o objetivo de verificar a interação entre diferentes módulos ou componentes do sistema, garantindo que os métodos funcionem corretamente quando executados em conjunto.

4.1 Daniel Fontoura

ID	Classe de Teste	Funcionalidade	Descrição do Cenário	Resultado Esperado
INT-01	ContaTransacaoIntegrationTest	Transferência entre Contas	**Fluxo de Transferência:** Realizar transferência entre duas instâncias reais de Conta, validando o débito na origem, crédito no destino e persistência no histórico.	Saldo Origem: -200 Saldo Destino: +200 Histórico atualizado.
INT-02	ContaTransacaoIntegrationTest	Depósito (Auto-Transferência)	**Fluxo de Depósito:** Conta realiza depósito para si mesma. Valida se o sistema reconhece a operação e atualiza o saldoTotalDepositado.	Saldo atualizado e transação registrada com destino = origem.

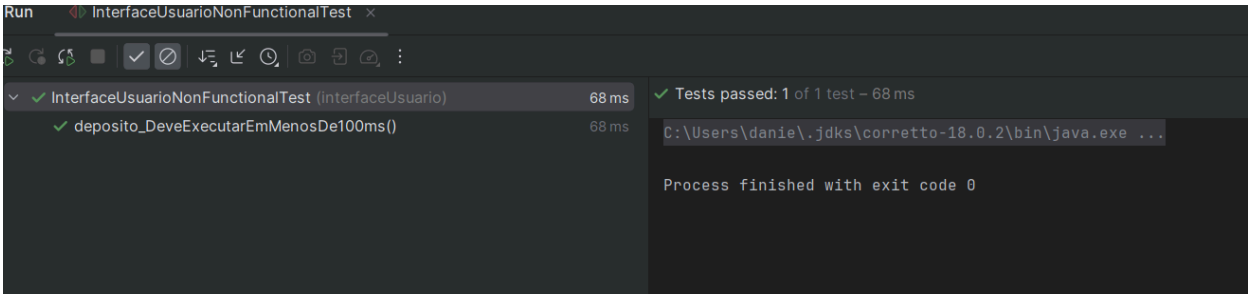
INT-03	ContaTransacaoIntegrationTest	Integridade de Dados	**Criação de Transação:** Verificar se a Conta, ao criar uma Transacao, preenche corretamente os metadados (Data, ID gerado, Valor).	Objeto Transação não nulo com dados consistentes.
INT-04	AgenciaIntegrationTest	Gestão de Clientes (Memória)	**Persistência de Cliente:** Adicionar um Cliente (Mock) na Agencia e recuperá-lo através do método de busca por CPF, validando o armazenamento em memória.	O cliente recuperado deve ser o mesmo objeto inserido.
INT-05	AgenciaIntegrationTest	Gestão de Boletos (Memória)	**Ciclo de Vida do Boleto:** Adicionar um Boleto na lista da Agencia, buscá-lo com sucesso e, após removê-lo, garantir que a busca falhe.	Boleto encontrado após inserção e exceção BuscaException após remoção.
INT-06	IntegracaoMemoriaTest	Criação de Cartão	**Integração Conta-Carteira:** Solicitar à Conta a criação de um cartão e verificar se ele foi corretamente instanciado e armazenado na lista interna da GerenciamentoCartao.	A carteira deve conter 1 cartão com os dados informados.

4.2 Daniele Pimenta

Os testes de integração desenvolvidos por mim tiveram como foco a minha classe escolhida (Conta), que é um dos módulos centrais do domínio bancário.

Teste	Objetivo	Resultado esperado	Resultado obtido
integracao_TransferenciaEntreContas()	Validar fluxo completo de transferência entre contas reais.	Saldos devem refletir corretamente a operação (origem diminui, destino aumenta).	Teste aprovado
ntegracao_CriarEPagarEmprestimo()	Verificar criação e pagamento de empréstimo.	Saldo deve aumentar com a criação e zerar ao quitar o empréstimo.	Teste aprovado
integracao_CriarCartaoStandard()	Validar criação de cartão Standard para ContaStandard.	Cartão do tipo Standard deve ser criado e adicionado à carteira.	Teste aprovado
integracao_CriarCartaoDiamond()	Validar criação de cartão Diamond para ContaDiamond.	Cartão do tipo Diamond deve ser criado corretamente.	Teste aprovado
integracao_CriarCartaoPremium()	Validar criação de cartão Premium para ContaPremium.	Cartão do tipo Premium deve ser criado e adicionado à carteira.	Teste aprovado

- Requisito Não Funcional:



4.3 Irhael Chagas

ID do Teste	Método de Teste	Objetivo da Integração	Resultado Esperado
INT-CLI-01	integracao_ValidacaoEmail_RegexGlobal	Validar se a Regex global de e-mail definida em VerificadorEntrada está sendo respeitada.	Sucesso: O sistema rejeitou corretamente um e-mail fora do padrão.
INT-CLI-02	integracao_ValidacaoTelefone_LimiteGlobal	Verificar se o limite de caracteres para telefone (constante global) é aplicado.	Sucesso: Telefone com tamanho excedente foi barrado.
INT-CLI-03	integracao_LogicaTipoCliente_Incoerencia	Validar a integração entre a escolha do Tipo de Cliente (Enum) e a regra de validação (CPF vs CNPJ).	Sucesso: O sistema identificou a incoerência ao tentar validar CPF para um cliente do tipo Empresa.
INT-CLI-04	integracao_Bug_CpfComLetras	Teste de Defeito: Verificar se a integração com a Regex de identificação permite falhas de segurança.	Falha Detectada (Bug): O sistema aceitou um CPF contendo letras, confirmando uma falha na Regex global. O teste confirmou a existência desse bug
INT-CLI-05	integracao_FluxoCompleto_Sucesso	Validar um fluxo completo onde todas as integrações (Regex, Constantes e Enums) estão corretas.	Sucesso: Cadastro aprovado.

4.4 João Otogali

ID do Teste	Método de Teste	Objetivo da Integração	Resultado Esperado
INT-01	integracao_ValidacaoEmail_Sucesso	Validar se o VerificadorPix utiliza corretamente a Regex de e-mail definida em VerificadorEntrada.	Aceitar e-mail válido (Retornar false).
INT-02	integracao_ValidacaoEmail_Falha	Validar rejeição de e-mail inválido pela Regex integrada.	Rejeitar e-mail sem domínio (Retornar true).
INT-03	integracao_ValidacaoTelefone_Limite	Validar se o limite de caracteres (12 dígitos) definido em VerificadorEntrada é respeitado.	Rejeitar telefones com 13 dígitos.
INT-04	integracao_Bug_ArrayFaltandoIdentificacao	Teste de Configuração: Verificar se o tipo "identificacao" existe no array ENTRADAS_CHAVE_PIX de VerificadorEntrada.	O tipo deve estar cadastrado no array.
INT-05	integracao_Bug_LogicaIdentificacao	Teste de Lógica Cruzada: Verificar se a integração rejeita textos alfanuméricos ("batata") no CPF.	O sistema deve rejeitar a entrada (Retornar true).

5. **Medidas de cada atributo de qualidade (ISO 25010)**

   Considerar escala:
   1: ruim / inexistente
   2: fraco
   3: razoável / aceitável
   4: bom
   5: excelente

Atributo ISO 25010	Nota (1-5)
Adequação Funcional	4 (bom)
Confiabilidade	3 (razoável / aceitável)
Usabilidade	3 (razoável / aceitável)
Eficiência de Desempenho	4 (bom)
Segurança	3 (razoável / aceitável)
Manutenibilidade	4 (bom)
Compatibilidade	4 (bom)
Portabilidade	5 (excelente)

1. **Adequação Funcional (4/5)**
   O sistema implementa bem o domínio bancário: tem criação de contas, inclusive por categorias (Standard, Premium, Diamond), cartões, pix, boletos, faz e paga empréstimos, histórico, notificações, transações agendadas etc. Contudo, há uma forte dependência de entrada de InterfaceUsuario.
2. **Confiabilidade (3/5)**
   O uso de exceções (EmprestimoException, TransacaoException, BuscaException etc), o controle de operações sensíveis como transferências, pagamento de boletos e empréstimos, e o histórico de transações e notificações ajudam a rastrear um acontecimento específico, estes são pontos positivos. Mas há limitações como dependência de variáveis estáticas da InterfaceUsuario, Agência é Singleton com estado global em memória, podendo gerar inconsistências em cenários complexos.
3. **Usabilidade (2/5)**
   O sistema é baseado em menu de console, portanto, não possui interface gráfica, não há acessibilidade, navegação e nem separação clara de camadas de apresentação (GUI/CLI), então, a usabilidade é fraca.
4. **Eficiência de Desempenho (4/5)**
   O sistema é todo em memória, sem banco de dados pesado, usa coleções simples para clientes, boletos e transações etc, sendo suficientes para o escopo da aplicação.

5. **Segurança (3/5)**
   Autenticação simples via senha. Algumas validações protegem o sistema contra operações inválidas, mas em um cenário real, há limitações: senha armazenada em texto puro, sem criptografia, controle de tentativas, logs de segurança. São fatores arriscados.
6. **Manutenibilidade (4/5)**
   O projeto no geral está bem organizado: há pacotes, uso de herança e polimorfismo, separação de responsabilidades razoável entre as classes, foram usadas ferramentas que contribuem com a evolução de um código (SonarQube, JUnit, Mockito, System Lambda etc).
7. **Compatibilidade (4/5)**
   O projeto se integra bem entre seus próprios módulos. A arquitetura orientada a objetos facilita uma futura integração com outras interfaces.
8. **Portabilidade (5/5)**
   Código inteiramente Java puro, sem dependência de sistemas operacionais específicos, sem uso de frameworks pesados e roda em qualquer ambiente com JDK +18.

6. Testes de Sistema

Irhael Sousa das Chagas

Visa garantir que o fluxo de negócio de Cadastro de Cliente resulte no estado correto da aplicação (persistência dos dados na memória simulada do banco).

Para esta etapa, foi simulada a interação de um usuário preenchendo o formulário de cadastro, verificando se o objeto Cliente é corretamente instanciado ou rejeitado conforme as regras de negócio.

SYS-CLI-01 Cadastro com Sucesso (Caminho Feliz):

Ação: Submeter formulário com todos os dados válidos (Pessoa Física).

Resultado: Sucesso: O objeto Cliente foi instanciado e persistido corretamente na memória.

SYS-CLI-02 Bloqueio por Regra de Negócio(Idade)

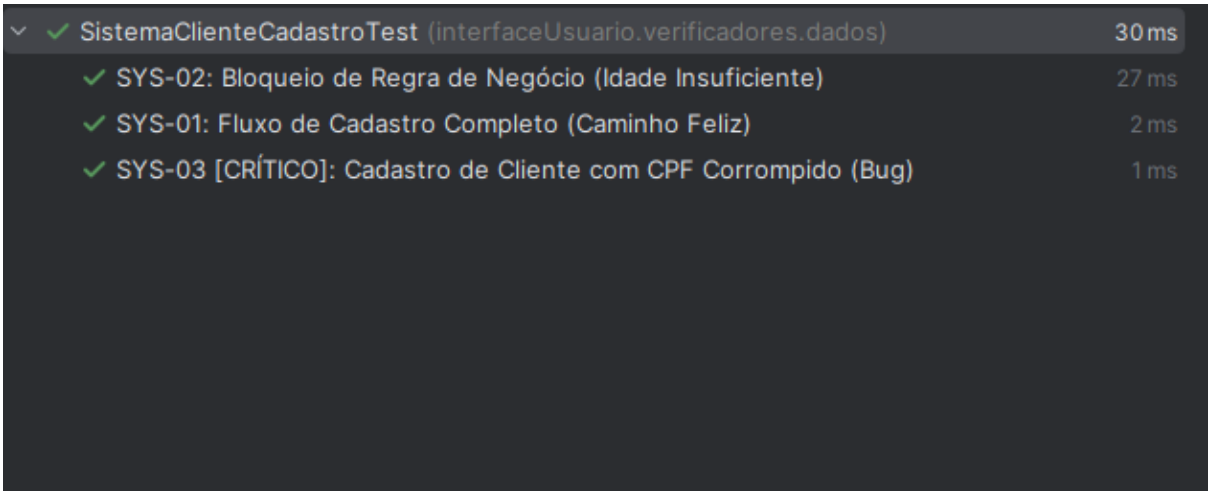
Ação: Tentar cadastrar um cliente menor de idade (ex: 10 anos).

Resultado: Bloqueio: O sistema capturou a exceção e NÃO criou o objeto Cliente, preservando a integridade do estado.

SYS-CLI-03 Impacto de Falha Crítica em relação ao CPF:

Ação: Submeter cadastro com CPF contendo letras ("1234567890a").

Resultado: Falha de Integridade: O sistema permitiu a criação e persistência de um objeto Cliente com dados corrompidos, confirmando que o bug do verificador polui a base de dados.



João Gabriel Otogali

SYS-04: Fluxo Completo de Transferência Pix (Caminho Feliz)

- **Legenda/Objetivo:** Validar a integridade de uma transação financeira completa (End-to-End) sob condições ideais. O teste simula um cliente com saldo suficiente realizando uma transferência para uma chave de e-mail válida, verificando se a validação da chave ocorre com sucesso e se o débito é contabilizado corretamente no saldo da conta.

SYS-05: Bloqueio de Transação por Insuficiência de Fundos

- **Legenda/Objetivo:** Verificar a regra de negócio de proteção de saldo. O teste simula um cenário onde o cliente possui uma chave Pix válida, mas tenta transferir um valor superior ao disponível em conta. O objetivo é garantir que o sistema valide a chave, mas **bloqueie a movimentação financeira**, mantendo o saldo original inalterado.

SYS-06: Validação de Impacto Financeiro de Falha Crítica (Bug do CPF)

- **Legenda/Objetivo:** Comprovar a propagação de falhas da camada unitária para a camada de sistema. O teste simula uma transferência utilizando uma chave de identificação inválida ("batata") detectada nos testes de integração. O objetivo é

demonstrar que a falha na validação permite a efetivação de um débito indevido, gerando **prejuízo financeiro** ao cliente e inconsistência contábil no banco (saída de dinheiro para destino inexistente).

ID	Cenário de Teste	Pré-Condição	Ação Realizada	Resultado Real
SYS-04	Transferência Pix (Caminho Feliz)	Saldo: R$ 1000Chave: Email Válido	Transferir R$ 200	Saldo debitado para R$ 800. Transação realizada.
SYS-05	Transferência sem Saldo	Saldo: R$ 50Chave: Email Válido	Tentar transferir R$ 100	Saldo mantido em R$ 50. Transação bloqueada.
SYS-06	Transferência para CPF Inválido	Saldo: R$ 500Chave: "batata"	Transferir R$ 500	Saldo debitado para R$ 0. Transação realizada.

Daniel Fontoura

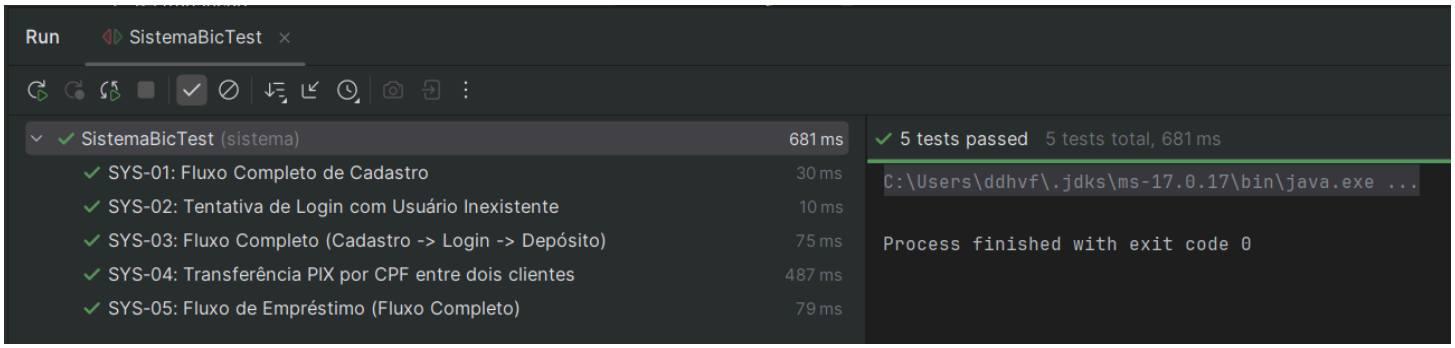
Foco: Simular a jornada do usuário final utilizando a interface de linha de comando (CLI), desde o menu principal até a conclusão da operação. Ferramenta: System Lambda.

ID	Descrição do Fluxo	Passos Simulados (Resumo)	Validação (Assert)
SYS-07	**Cadastro Completo de Cliente**	1. Iniciar Aplicação 2. Escolher "Criar Conta" 3. Preencher formulário (Nome, CPF, Endereço, Senha, etc.)	Verificar mensagem "Bem vindo [Nome]" e encerramento sem erros.

		4. Sair (Logoff)	
SYS-08	**Login com Credenciais Inválidas**	1. Iniciar App 2. Escolher "Acessar Conta" 3. Digitar CPF inexistente 4. Digitar Senha	Verificar presença da mensagem de erro "Cliente nao encontrado".
SYS-09	**Fluxo Financeiro (Depósito)**	1. Cadastrar Cliente 2. Logar 3. Consultar Saldo (0.0) 4. Depositar 500.0 5. Consultar Saldo novamente	Verificar se a saída do console mostra "SALDO >> 500.0" na segunda consulta.
SYS-10	**Transferência PIX (Email/CPF)**	1. Cadastrar Pagador 2. Cadastrar Recebedor 3. Logar Pagador e Depositar 4. Transferir via Chave PIX 5. Consultar Saldo	Verificar se o comprovante "DESTINO DA TRANSACAO" foi impresso e se o saldo foi debitado.

SYS-11	**Ciclo de Empréstimo**	1. Cadastrar e Logar 2. Pedir Empréstimo (ex: 1200 em 12x) 3. Pagar 1ª Parcela	Verificar mensagens "EMPRESTIMO REALIZADO" e "PARCELA PAGA", e saldo final correto.

Resultados:



7. Técnicas Funcional, Estrutural e Baseada em Defeitos

7.1 Técnica Funcional

7.1.1 Daniel Fontoura

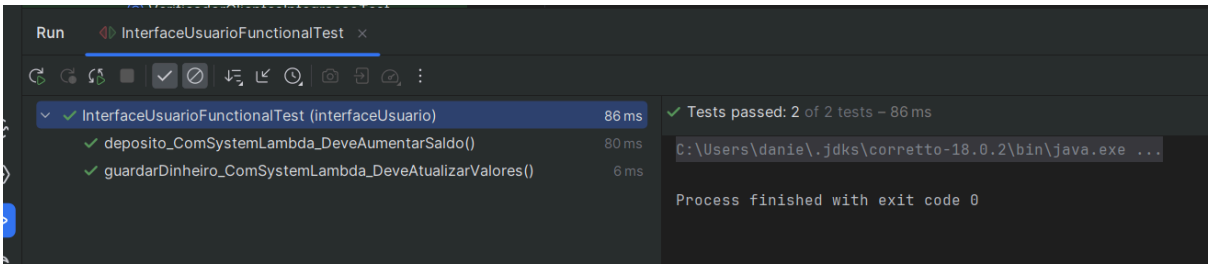
O objetivo aqui foi validar se o software atende aos requisitos funcionais e regras de negócio especificados, focando nas entradas e saídas esperadas sem considerar a estrutura interna do código.

Aplicação: Mapeamos as Regras de Negócio (RF) e Fluxos de Uso para Casos de Teste específicos.

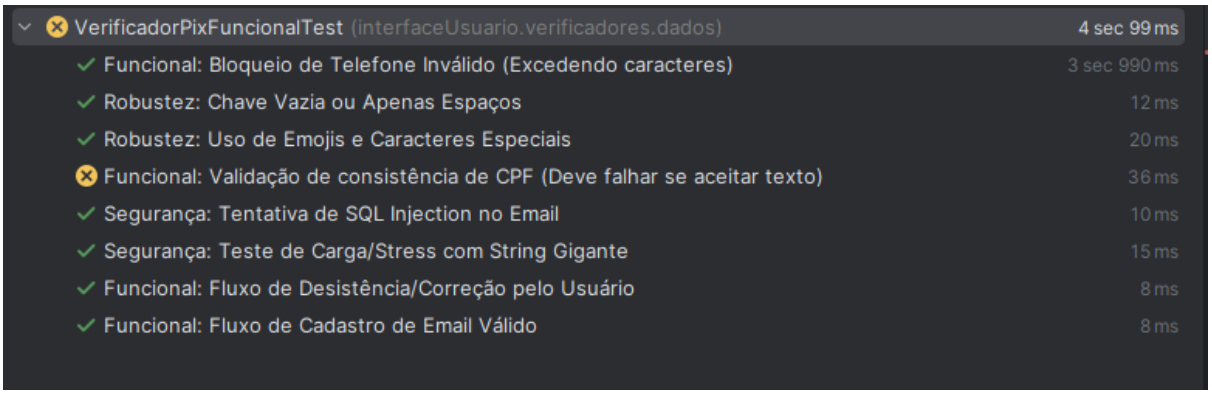
ID Requisito	Descrição da Regra / Fluxo	ID do Caso de Teste	Status
RF-01	Rejeitar valores não numéricos ou negativos em transações.	1.1.1 (Erro Formato)1.1.2 (Valor Negativo)	Aprovado

RF-02	Impedir transferências sem saldo suficiente.	1.2.1 (Saldo OK)1.2.1-F (Sem Saldo)	Aprovado
RF-03	Respeitar limites de depósito por tipo de conta (Standard/Premium).	1.3.1, 1.3.2, 1.3.3	Aprovado
RF-04	Validar limite diário de depósitos.	1.3.1-S 1.3.2-S, 1.3.3-S (Soma Excede)	Aprovado
RF-05	Impedir pagamento de fatura maior que a dívida.	2.1.3 (Valor Excedente)	Aprovado
RF-06	Validar criação de boletos (Data/Multa).	4.3 (Multa Neg.), 4.4 (Data Inv.)	Aprovado
RF-08	Fluxo completo de Cadastro e Login.	SYS-01	Aprovado
RF-09	Fluxo de Transferência via PIX entre clientes.	SYS-04	Aprovado
RF-10	Ciclo de contratação e pagamento de Empréstimo.	SYS-05	Aprovado

7.1.2 Daniele Pimenta

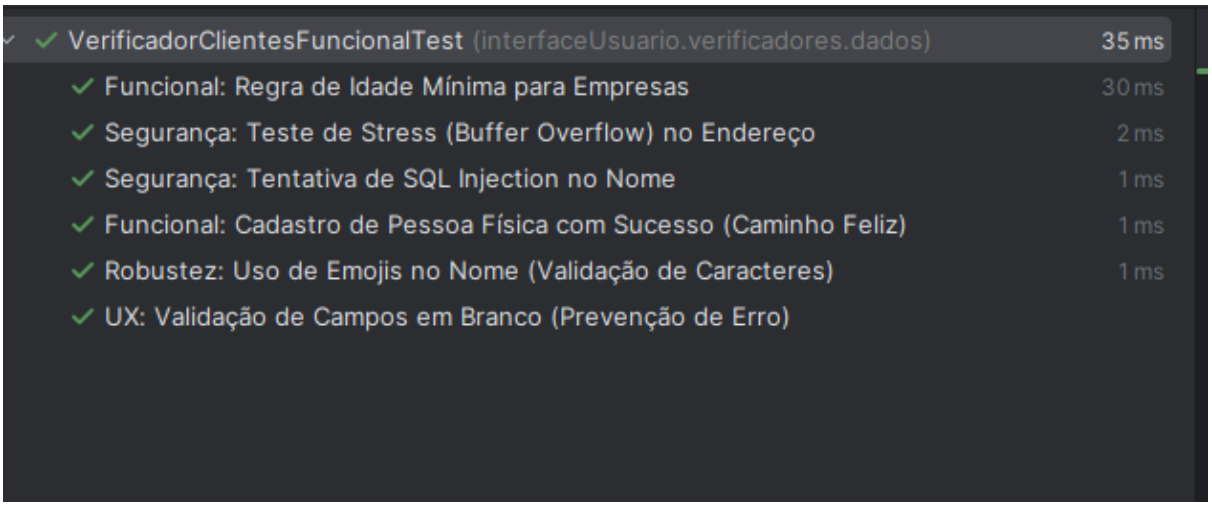


7.1.3 João Otogali



ID	Cenário de Teste	Ação Realizada	Resultado Esperado	Resultado Obtido
TF-01	Cadastro com Sucesso	Usuário insere um e-mail válido.	Sistema deve aceitar e finalizar.	Sistema aceitou.
TF-02	Limite de Caracteres	Usuário insere telefone com mais de 12 dígitos.	Sistema deve bloquear e pedir correção.	Sistema bloqueou.
TF-03	Fluxo de Desistência	Usuário digita "0" para cancelar a operação.	Sistema deve cancelar/reinici ar.	Sistema cancelou.
TF-04	Validação de Documento	Usuário tenta cadastrar texto ("batata") como CPF.	Sistema deve bloquear a entrada inválida.	Sistema ACEITOU o texto como CPF válido.

7.1.4. Irhael Sousa das Chagas



ID	Cenário de Teste	Ação Realizada	Resultado Esperado	Resultado Obtido
TF-01	Caminho Feliz	Cadastro de Pessoa Física com dados padrões e válidos.	Sucesso (Aprovado).	Sucesso (Aprovado).
TF-02	Segurança (SQL Injection)	Tentativa de injeção de comandos SQL no campo de nome.	Sucesso (Bloqueado)..	Sucesso (Bloqueado).
TF-03	Segurança (Buffer Overflow)	Teste de estresse enviando input massivo (50k caracteres) no endereço.	Sucesso (Rejeitado sem travar o sistema).	Sucesso (Rejeitado sem travar o sistema).
TF-04	Robustez (Sanitização)	Tentativa de cadastro utilizando Emojis e caracteres especiais no nome..	Sucesso (Bloqueado)..	Sucesso (Bloqueado).
TF-05	Compliance (Regra de Negócio)	Validação da idade mínima para empresas (3 anos de mercado).	Sucesso (Regra aplicada corretamente).	Sucesso (Regra aplicada corretamente).

TF-06	UX/Usabilidade	Tentativa de envio com campos obrigatórios em branco.	Sucesso (Bloqueado com mensagem de orientação).	Sucesso (Bloqueado com mensagem de orientação).

7.2 Técnica Estrutural

7.2.1 Daniel Fontoura

Aqui o objetivo foi garantir a cobertura do código fonte, assegurando que todas as estruturas de decisão foram exercitadas. Critério Utilizado: Todas-Arestas - Cada desvio verdadeiro/falso de if, switch ou catch foi percorrido.

Aplicação: Análise do fluxo de controle da classe VerificadorTransacao.

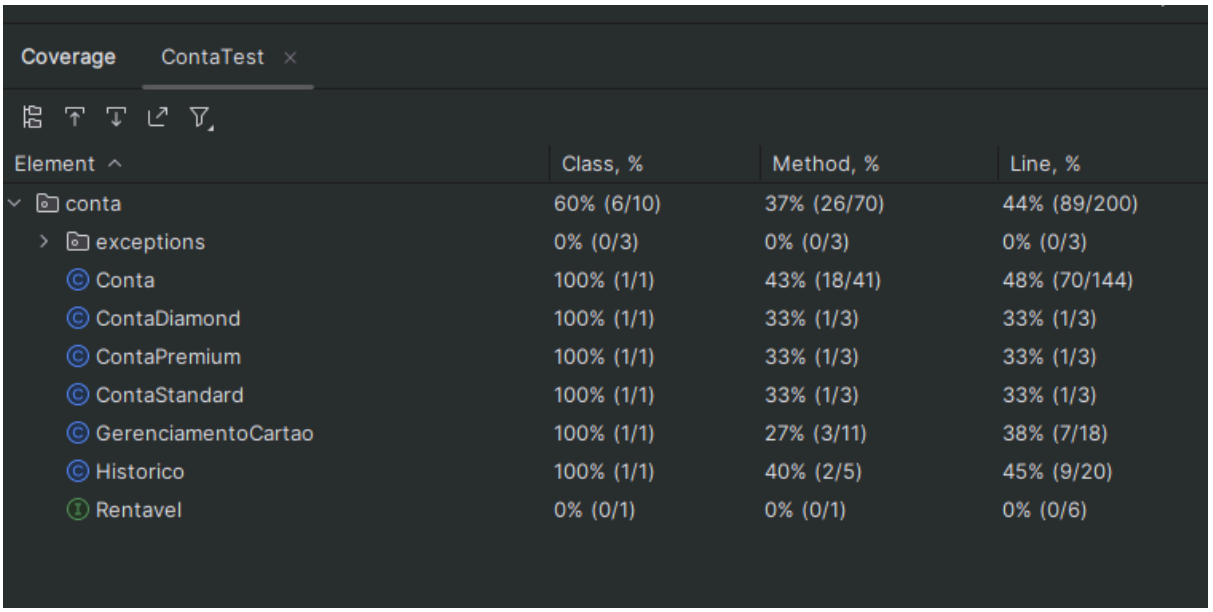
Métrica Alcançada: 80% de Cobertura de Arestas (Meta: >80%)

Estrutura de Decisão	**Aresta (Caminho)**	**Teste que Cobre a Aresta**
Try/Catch (Parse)	Falha no parse (Catch)	1.1.1 (Entrada "abc")
IF (TipoOperacao)	Verdadeiro (Transferência)	1.2.1
ELSE IF (TipoOperacao)	Verdadeiro (Depósito)	1.3.1
IF (Valor > 0)	Verdadeiro (> 0)	1.3.2
IF (Valor > 0)	Falso (<= 0)	1.4.1 (Valor 0), 1.4.2 (Valor -50)
SWITCH (TipoConta)	Case STANDARD	1.3.1
SWITCH (TipoConta)	Case PREMIUM	1.3.2
SWITCH (TipoConta)	Case DIAMOND	1.3.3
SWITCH (TipoConta)	Default (Inválido)	1.3.4
IF (Pagar Fatura)	Verdadeiro	2.1.1
IF (Valor > Fatura)	Verdadeiro (Erro)	2.1.3
ELSE IF (Aumentar)	Verdadeiro	2.2.1

Element	Class, %	Method, %	Line, %	Branch, %
interfaceUsuario.verificadores.	100% (1/1)	100% (6/6)	90% (37/41)	80% (33/41)
VerificadorTransacao	100% (1/1)	100% (6/6)	90% (37/41)	80% (33/41)

7.2.2 Daniele Pimenta

Cobertura inicial:

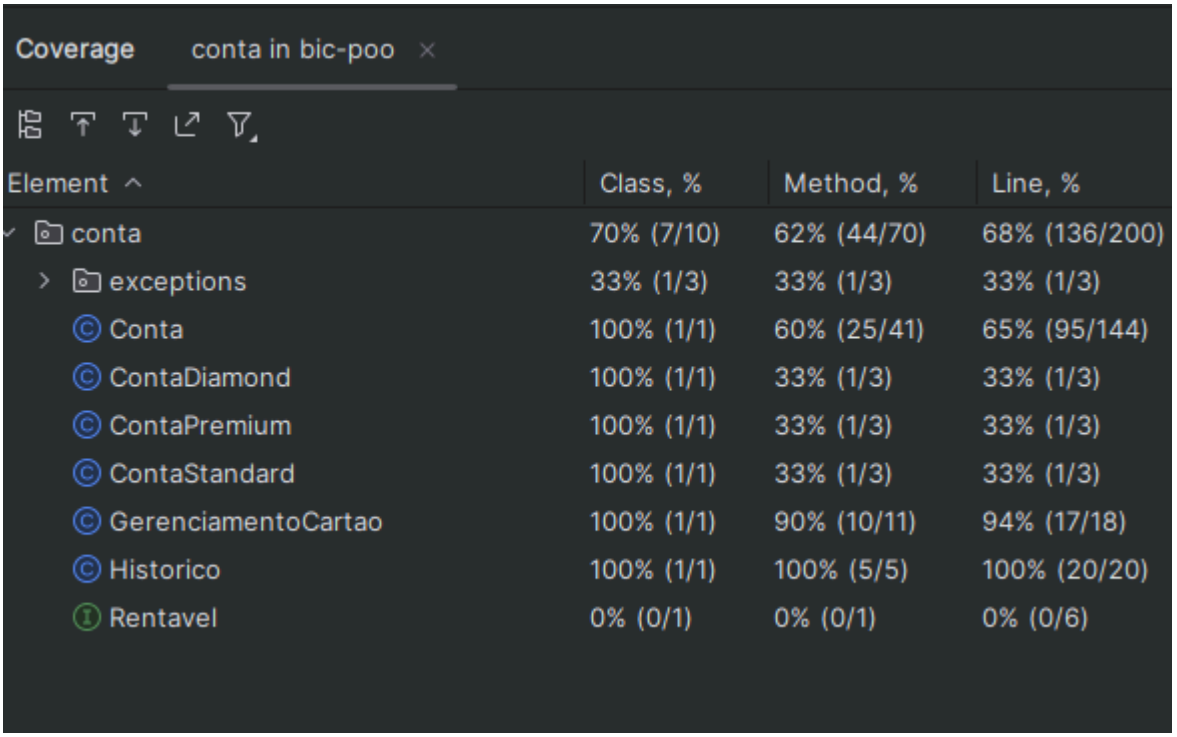


Element	Class, %	Method, %	Line, %
conta	60% (6/10)	37% (26/70)	44% (89/200)
exceptions	0% (0/3)	0% (0/3)	0% (0/3)
Conta	100% (1/1)	43% (18/41)	48% (70/144)
ContaDiamond	100% (1/1)	33% (1/3)	33% (1/3)
ContaPremium	100% (1/1)	33% (1/3)	33% (1/3)
ContaStandard	100% (1/1)	33% (1/3)	33% (1/3)
GerenciamentoCartao	100% (1/1)	27% (3/11)	38% (7/18)
Historico	100% (1/1)	40% (2/5)	45% (9/20)
Rentavel	0% (0/1)	0% (0/1)	0% (0/6)

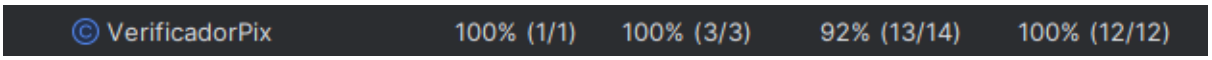
Criação dos novos testes:

Teste	Objetivo	Comportamento esperado	Resultado
criarCartao_TipoInvalido_DeveLancarErro()	Garantir que cartões inválidos geram exceção.	Deve lançar TipoInvalido.	Exceção lançada corretamente.
setDinheiroGuardado_Guardar()	Verificar guardar dinheiro na conta.	Saldo deve ficar 400 e dinheiroGuardado 100.	Valores atualizados corretamente
setDinheiroGuardado_Resgatar()	Verificar resgate do dinheiro guardado.	Saldo volta para 500 e dinheiroGuardado fica 0.	Lógica funciona corretamente
pagarFatura_DeveDiminuirSaldoEAumentarLimite()	Verificar pagamento de fatura.	Saldo final deve ser 400 e limite restaurado.	Saldo atualizado conforme esperado
diminuirLimiteAtual_DeveAumentarLimiteUsado()	Validar registro do uso de limite no cartão.	Fatura deve ser 100.	Atualizado corretamente
getLimiteMaximo_SemCartao_DeveLancarExcecao()	Validar retorno de limite máximo.	Deve retornar 1000	Retornado corretamente

Resultado final da cobertura após a implementação dos novos testes:

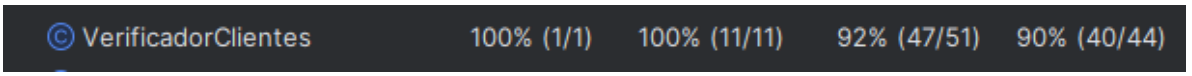


7.2.3 João Otogali




```java
public class VerificadorPix {  50 usages  Isadora Ribeiro
    public static boolean chavePix(String entrada, String tipoChavePix) {  41 usages  Isadora Ribeiro
        System.out.println("A CHAVE INSERIDA " + entrada + " ESTA CORRETA? [1] SIM! [0] NAO, PRECISO TROCAR");
        if (TECLADO.nextLine().equals("0")) {
            return true;
        } else {
            switch (tipoChavePix) {
                case DadosChavesPix.TELEFONE:
                    return !VerificadorClientes.verificarTelefone(entrada);
                case DadosChavesPix.EMAIL:
                    return !VerificadorClientes.verificarEmail(entrada);
                case DadosChavesPix.CHAVE_ALEATORIA:
                    return !chaveAleatoria(entrada);
                case DadosChavesPix.IDENTIFICACAO:
                    return !VerificadorClientes.verificadorIdentificacao(entrada);
            }
        }
        return true;
    }
```

7.2.4. Irhael Sousa das Chagas




```java
public class VerificadorClientes {  Isadora Ribeiro

    private static boolean isAlphabetic(String e) { return !NAO_APENAS_ALFABETO.matcher(e).find(); }

    static boolean isAlphanumeric(String e) {  4 usages  Isadora Ribeiro
        return !NAO_APENAS_ALFABETO_DIGITO.matcher(e).find();
    }

    public static boolean verificarEndereco(String[] entradaEndereco) {  9 usages  Isadora Ribeiro
        try {
            Integer.parseInt(entradaEndereco[0]);
            Integer.parseInt(entradaEndereco[1]);
            if (!(entradaEndereco[0].length() == TAMANHO_CEP)) {
                throw new ValorInvalido( msg: "CEP INVALIDO");
            }
            return true;
        } catch (Exception ignored) {

        }
        return false;
    }

    protected static boolean verificarTelefone(String e) { return e.length() <= DIGITOS_MAXIMO_TELEFONE; }

    protected static boolean verificarEmail(String e) {  10 usages  Isadora Ribeiro
        Matcher matcher = EMAIL_VERIFICADOR.matcher(e);
        return matcher.matches();
    }

    public static boolean verificarIdade(String idade, TiposClientes tiposClientes) {  5 usages  Isadora Ribeiro
        if (tiposClientes == TiposClientes.CLIENTE_EMPRESA) {
            return Integer.parseInt(idade) >= 3;
        }
        return Integer.parseInt(idade) >= 18;
    }

    protected static boolean verificarSenha(String s) {  4 usages  Isadora Ribeiro
        if (HAS_WHITESPACE.matcher(s).find()) {
            return false;
        }
        return s.length() >= TAMANHO_MINIMO_SENHA;
```

7.3 Técnica Baseada em Defeitos

Para os testes baseados em defeitos, optamos por utilizar teste de mutação - Mutation Testing. O Mutation Testing nada mais é do que uma técnica que simula defeitos pequenos no código (que chamamos de mutantes) e que verifica se os testes conseguem detectá-los, avaliando a qualidade dos testes.

7.3.1 Daniel Fontoura

Como não foi possível utilizar a ferramenta PitTest, realizei a mutação de forma manual em alguns cenários, visando simular o comportamento mutante e validando o antes e depois dos testes com as mutações aplicadas.

Mutante **MT-01**: Alteração de Condicional (Depósito)

- Onde: Método dadosTransacao.
- O que faz: Permite depósitos de valor zero (que deveriam ser rejeitados).

Mutante **MT-02**: Alteração de Limite (Conta Standard)

- Onde: Método validarRegrasDeposito, dentro do case STANDARD.
- O que faz: Rejeita um depósito que seja exatamente igual ao limite máximo (o que deveria ser permitido).

Mutante **MT-03**: Enfraquecimento Lógico

- Onde: Método validarRegrasDeposito, dentro do case STANDARD (ou qualquer outro case).
- O que faz: Permite o depósito se APENAS UMA das condições for verdade

Mutante **MT-04**: Remoção de Instrução (Pagamento)

- Onde: Método valorFatura, dentro do bloco if (tipoOperacao == ... PAGAR_FATURA).
- O que faz: Remove a trava de segurança que impede pagar mais do que se deve.

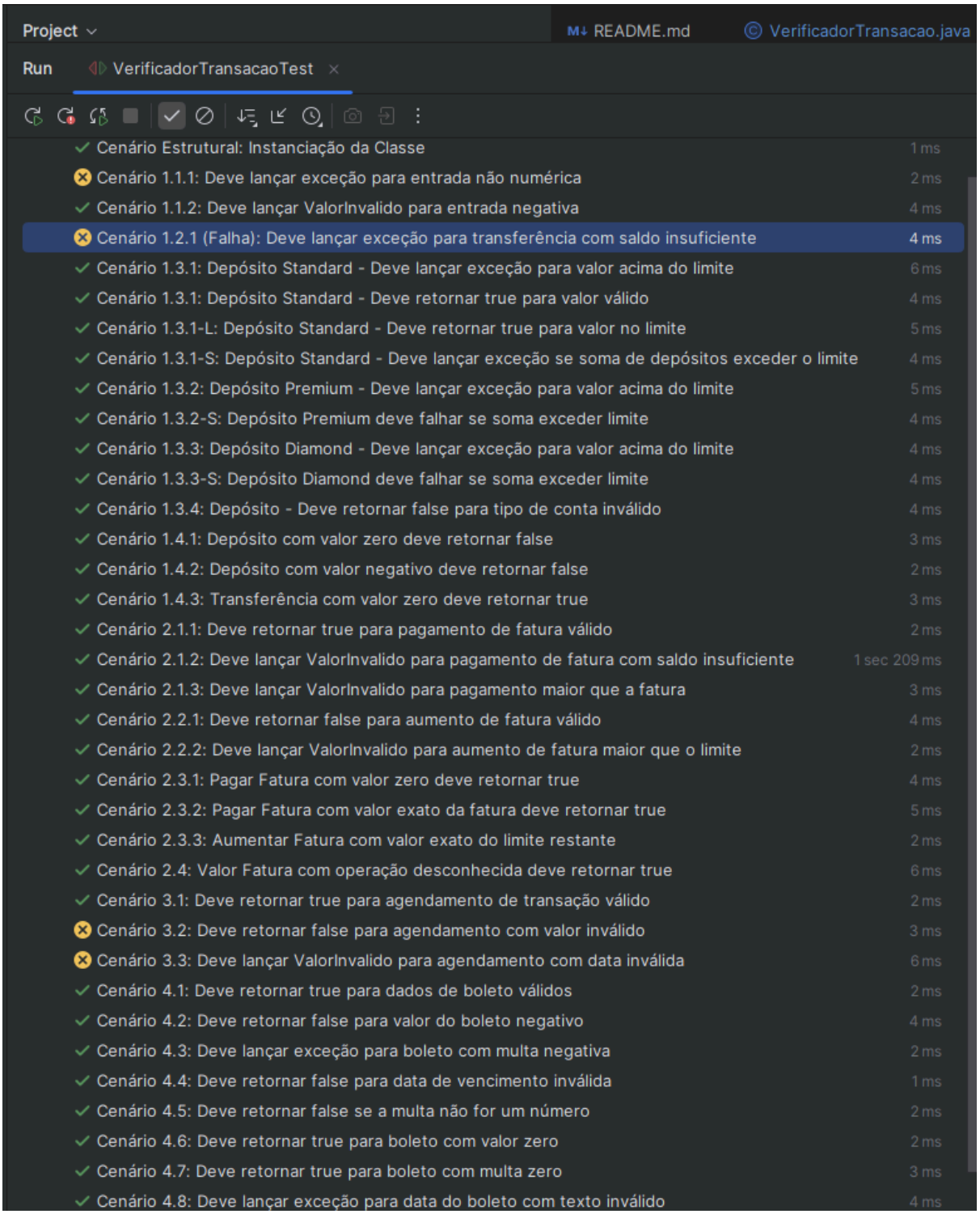
Mutante **MT-05**: Inversão Lógica (Aumento de Limite)

- Onde: Método valorFatura, dentro do bloco else if (... AUMENTAR_FATURA).
- O que faz: Inverte a lógica, impedindo aumentos válidos e permitindo inválidos.

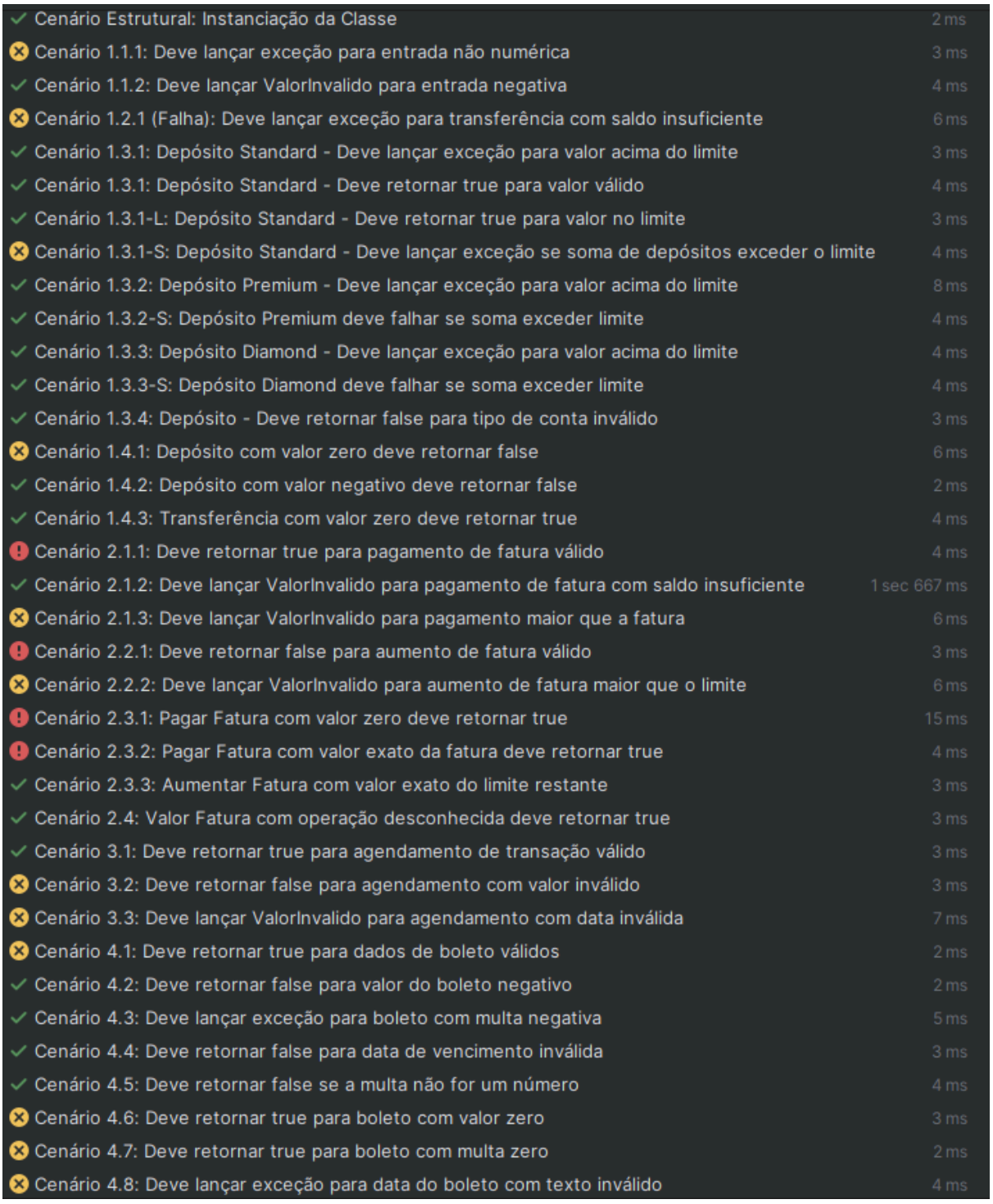
Mutante **MT-06**: Negação de Booleano (Boleto)

- Onde: Método verificarBoleto, na primeira linha do if.
- O que faz: Aceita datas inválidas e rejeita datas válidas.

Antes


Project
README.md
VerificadorTransacao.java
Run
VerificadorTransacaoTest
Cenário Estrutural: Instanciação da Classe 1 ms
Cenário 1.1.1: Deve lançar exceção para entrada não numérica 2 ms
Cenário 1.1.2: Deve lançar ValorInvalido para entrada negativa 4 ms
Cenário 1.2.1 (Falha): Deve lançar exceção para transferência com saldo insuficiente 4 ms
Cenário 1.3.1: Depósito Standard - Deve lançar exceção para valor acima do limite 6 ms
Cenário 1.3.1: Depósito Standard - Deve retornar true para valor válido 4 ms
Cenário 1.3.1-L: Depósito Standard - Deve retornar true para valor no limite 5 ms
Cenário 1.3.1-S: Depósito Standard - Deve lançar exceção se soma de depósitos exceder o limite 4 ms
Cenário 1.3.2: Depósito Premium - Deve lançar exceção para valor acima do limite 5 ms
Cenário 1.3.2-S: Depósito Premium deve falhar se soma exceder limite 4 ms
Cenário 1.3.3: Depósito Diamond - Deve lançar exceção para valor acima do limite 4 ms
Cenário 1.3.3-S: Depósito Diamond deve falhar se soma exceder limite 4 ms
Cenário 1.3.4: Depósito - Deve retornar false para tipo de conta inválido 4 ms
Cenário 1.4.1: Depósito com valor zero deve retornar false 3 ms
Cenário 1.4.2: Depósito com valor negativo deve retornar false 2 ms
Cenário 1.4.3: Transferência com valor zero deve retornar true 3 ms
Cenário 2.1.1: Deve retornar true para pagamento de fatura válido 2 ms
Cenário 2.1.2: Deve lançar ValorInvalido para pagamento de fatura com saldo insuficiente 1 sec 209 ms
Cenário 2.1.3: Deve lançar ValorInvalido para pagamento maior que a fatura 3 ms
Cenário 2.2.1: Deve retornar false para aumento de fatura válido 4 ms
Cenário 2.2.2: Deve lançar ValorInvalido para aumento de fatura maior que o limite 2 ms
Cenário 2.3.1: Pagar Fatura com valor zero deve retornar true 4 ms
Cenário 2.3.2: Pagar Fatura com valor exato da fatura deve retornar true 5 ms
Cenário 2.3.3: Aumentar Fatura com valor exato do limite restante 2 ms
Cenário 2.4: Valor Fatura com operação desconhecida deve retornar true 6 ms
Cenário 3.1: Deve retornar true para agendamento de transação válido 2 ms
Cenário 3.2: Deve retornar false para agendamento com valor inválido 3 ms
Cenário 3.3: Deve lançar ValorInvalido para agendamento com data inválida 6 ms
Cenário 4.1: Deve retornar true para dados de boleto válidos 2 ms
Cenário 4.2: Deve retornar false para valor do boleto negativo 4 ms
Cenário 4.3: Deve lançar exceção para boleto com multa negativa 2 ms
Cenário 4.4: Deve retornar false para data de vencimento inválida 1 ms
Cenário 4.5: Deve retornar false se a multa não for um número 2 ms
Cenário 4.6: Deve retornar true para boleto com valor zero 2 ms
Cenário 4.7: Deve retornar true para boleto com multa zero 3 ms
Cenário 4.8: Deve lançar exceção para data do boleto com texto inválido 4 ms

Depois:


Cenário Estrutural: Instanciação da Classe 2 ms
Cenário 1.1.1: Deve lançar exceção para entrada não numérica 3 ms
Cenário 1.1.2: Deve lançar ValorInvalido para entrada negativa 4 ms
Cenário 1.2.1 (Falha): Deve lançar exceção para transferência com saldo insuficiente 6 ms
Cenário 1.3.1: Depósito Standard - Deve lançar exceção para valor acima do limite 3 ms
Cenário 1.3.1: Depósito Standard - Deve retornar true para valor válido 4 ms
Cenário 1.3.1-L: Depósito Standard - Deve retornar true para valor no limite 3 ms
Cenário 1.3.1-S: Depósito Standard - Deve lançar exceção se soma de depósitos exceder o limite 4 ms
Cenário 1.3.2: Depósito Premium - Deve lançar exceção para valor acima do limite 8 ms
Cenário 1.3.2-S: Depósito Premium deve falhar se soma exceder limite 4 ms
Cenário 1.3.3: Depósito Diamond - Deve lançar exceção para valor acima do limite 4 ms
Cenário 1.3.3-S: Depósito Diamond deve falhar se soma exceder limite 4 ms
Cenário 1.3.4: Depósito - Deve retornar false para tipo de conta inválido 3 ms
Cenário 1.4.1: Depósito com valor zero deve retornar false 6 ms
Cenário 1.4.2: Depósito com valor negativo deve retornar false 2 ms
Cenário 1.4.3: Transferência com valor zero deve retornar true 4 ms
Cenário 2.1.1: Deve retornar true para pagamento de fatura válido 4 ms
Cenário 2.1.2: Deve lançar ValorInvalido para pagamento de fatura com saldo insuficiente 1 sec 667 ms
Cenário 2.1.3: Deve lançar ValorInvalido para pagamento maior que a fatura 6 ms
Cenário 2.2.1: Deve retornar false para aumento de fatura válido 3 ms
Cenário 2.2.2: Deve lançar ValorInvalido para aumento de fatura maior que o limite 6 ms
Cenário 2.3.1: Pagar Fatura com valor zero deve retornar true 15 ms
Cenário 2.3.2: Pagar Fatura com valor exato da fatura deve retornar true 4 ms
Cenário 2.3.3: Aumentar Fatura com valor exato do limite restante 3 ms
Cenário 2.4: Valor Fatura com operação desconhecida deve retornar true 3 ms
Cenário 3.1: Deve retornar true para agendamento de transação válido 3 ms
Cenário 3.2: Deve retornar false para agendamento com valor inválido 3 ms
Cenário 3.3: Deve lançar ValorInvalido para agendamento com data inválida 7 ms
Cenário 4.1: Deve retornar true para dados de boleto válidos 2 ms
Cenário 4.2: Deve retornar false para valor do boleto negativo 2 ms
Cenário 4.3: Deve lançar exceção para boleto com multa negativa 5 ms
Cenário 4.4: Deve retornar false para data de vencimento inválida 3 ms
Cenário 4.5: Deve retornar false se a multa não for um número 4 ms
Cenário 4.6: Deve retornar true para boleto com valor zero 3 ms
Cenário 4.7: Deve retornar true para boleto com multa zero 2 ms
Cenário 4.8: Deve lançar exceção para data do boleto com texto inválido 4 ms

Escore de Mutação:

- **Mutantes Gerados:** 6
- **Mutantes Mortos:** 6
- **Mutantes Vivos:** 0
- **Escore Final: 100%** (6/6)

Conclusão: A suíte de testes demonstrou alta eficácia (100%), sendo capaz de detectar alterações sutis na lógica de negócio, como mudanças em operadores relacionais e remoção de blocos de validação. O fato de múltiplos testes falharem para um único mutante indica uma redundância positiva na cobertura de testes.

7.3.2 Daniele Pimenta

Não consegui rodar o PIT (Mutation Testing) corretamente, provavelmente devido a estrutura desorganizada do projeto. Os testes funcionam, mas o Pit não consegue rodá-los.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
6	0%	0/227	0%	0/141	0%	0/0

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
conta	6	0%	0/227	0%	0/141	0%	0/0

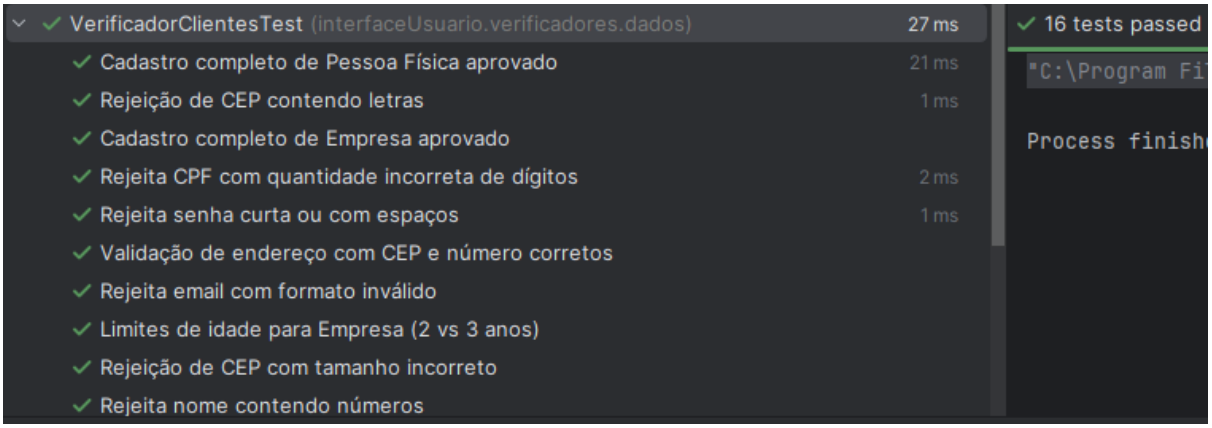
Report generated by PIT 1.9.0

7.3.1 Irhael Sousa das Chagas

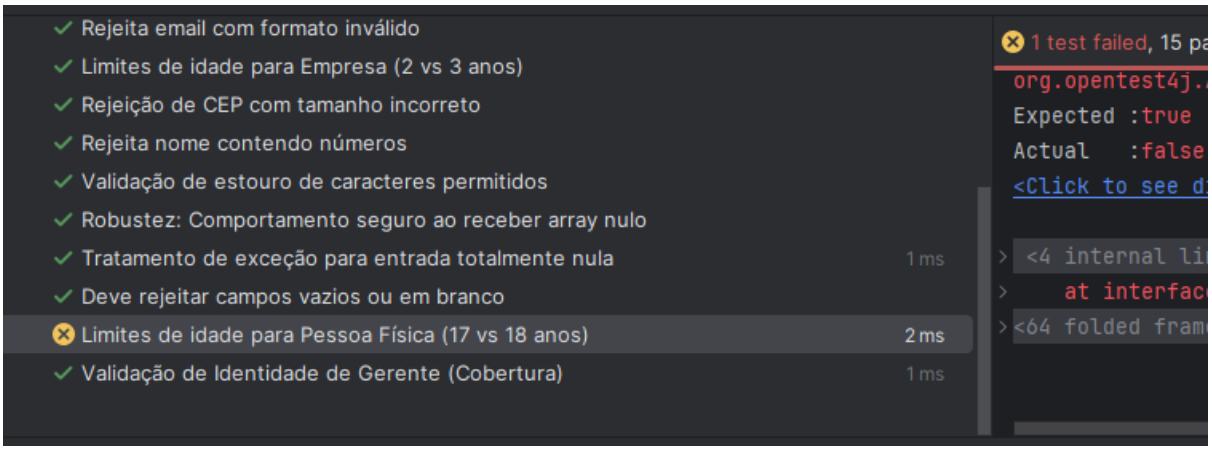
MUT - 01: O operador relacional da verificação de idade foi alterado propositalmente de >= (maior ou igual) para > (maior estrito).

```
public static boolean verificarIdade(String idade, TiposClientes tiposClientes) {  5 usages  Isadora Ribeiro *
    if (tiposClientes == TiposClientes.CLIENTE_EMPRESA) {
        return Integer.parseInt(idade) >= 3;
    }
    return Integer.parseInt(idade) > 18;
}
```

Antes:



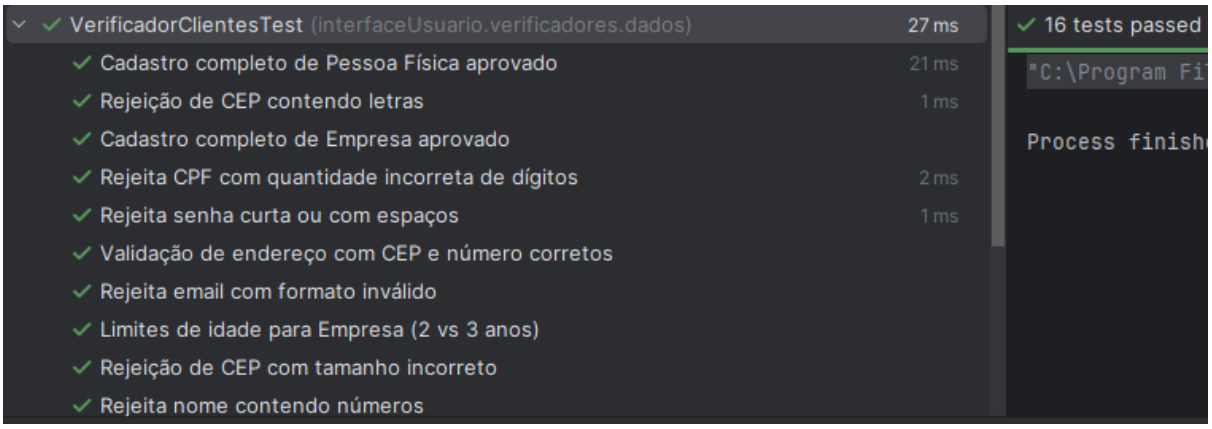
Depois:



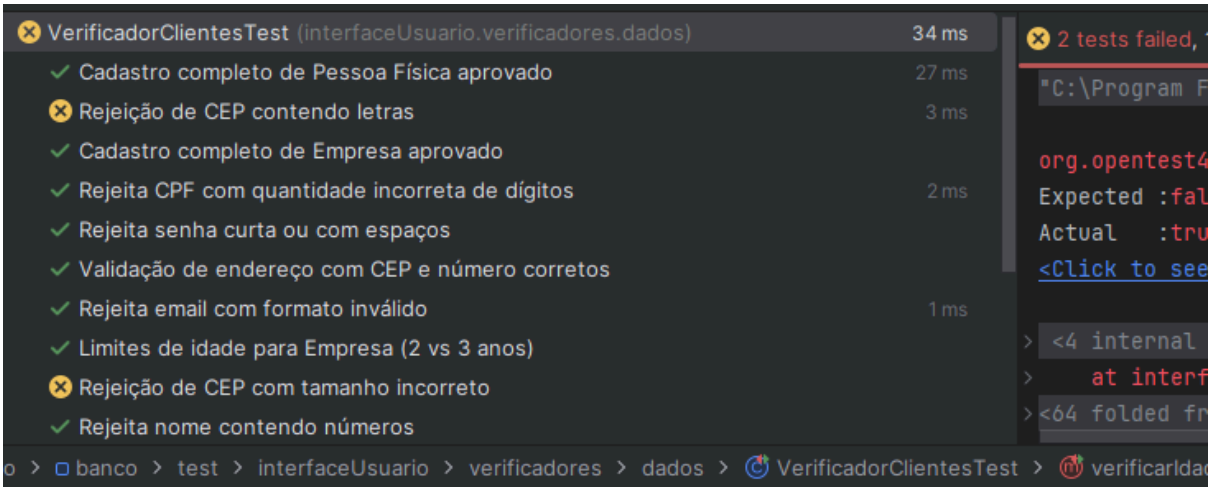
MUT - 02: O tratamento de erro no método verificarEndereco foi sabotado para retornar true (Válido) quando ocorresse uma falha de conversão numérica (Exceção), simulando uma falha silenciosa.

```
public static boolean verificarEndereco(String[] entradaEndereco) {  9 usages  Isadora Ribeiro *
    try {
        // ... código de validação ...
        return true;
    } catch (Exception e) {
        return true; // <--- ESSE É O BUG (MUTANTE)
        // Estamos dizendo: "Se der erro, finge que está tudo bem e aceita."
    }
    // return false;
}
```

Antes:



Depois:

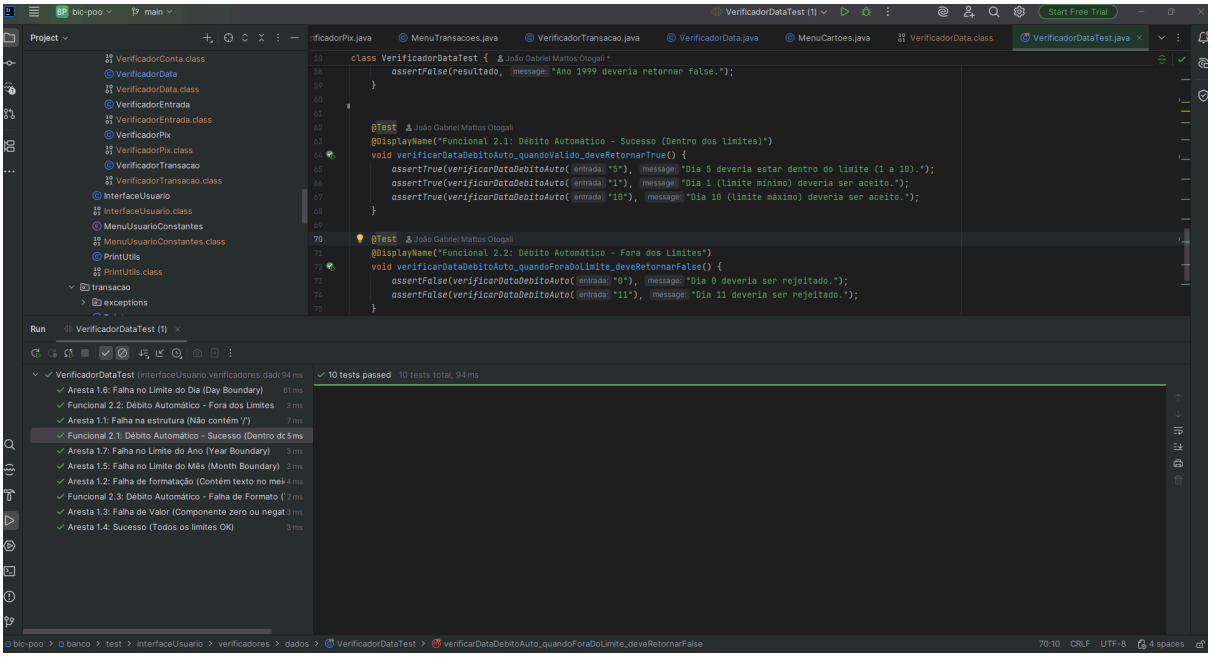


A análise via Mutação Manual confirmou a alta confiabilidade da suíte de testes desenvolvida para a classe VerificadorClientes. Ao introduzirmos falhas deliberadas na lógica de validação, especificamente nas regras de maioridade e tratamento de exceções, o sistema de testes reagiu instantaneamente, bloqueando o código alterado.

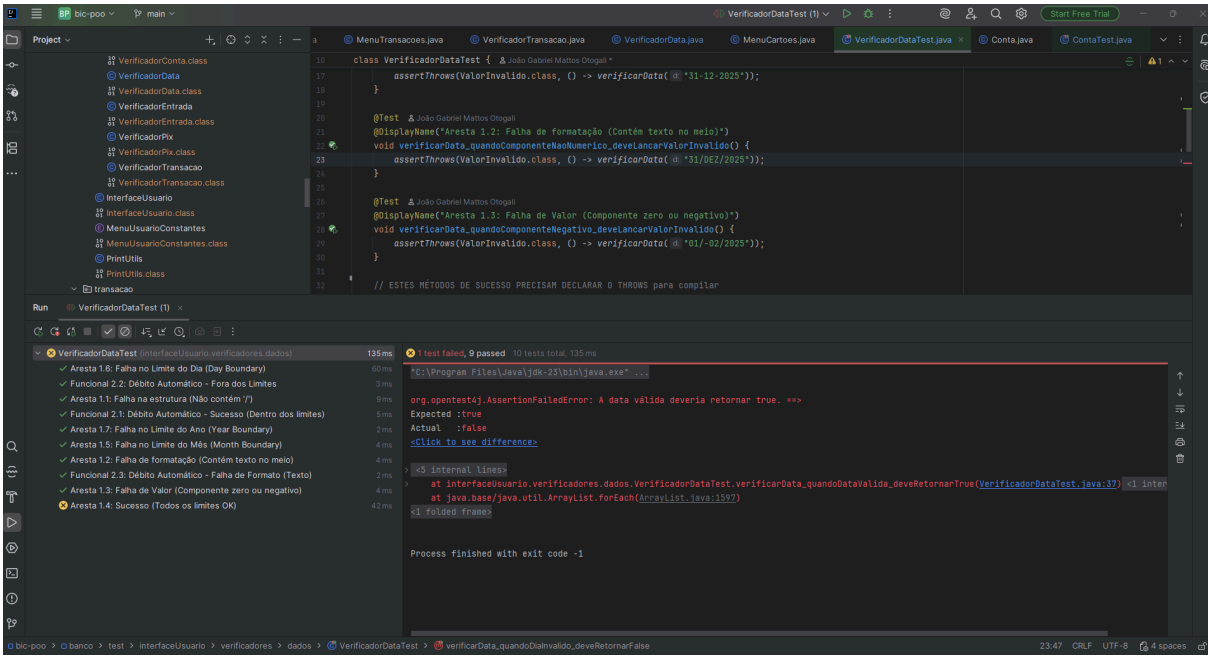
Essa resposta imediata (falha nos testes) evidencia que a cobertura atual não é apenas numérica, mas funcionalmente eficaz. O fato de o teste de fronteira ter barrado a mudança do operador relacional (de >= para >) comprova que a malha de segurança está ajustada para detectar até mesmo regressões sutis, garantindo que as regras de negócio cruciais do sistema permaneçam íntegras durante futuras manutenções.

7.3.2 João Otogali

Antes:



Depois:



ID	Operador de Mutação	Alteração Injetada na Classe	Teste que Falhou	Veredito
MUT-01	Condição Relacional	Alteração da condição de ano: >= 2000 foi mudado para > 2000.	verificarData_quandoDataValida_deveRetornarTrue	Morto
MUT-02	Constante (Simulação)	Alteração do valor de um retorno (return false para return true em um bloco de erro).	verificarDataDebitoAuto_quandoTexto	Morto

Foi aplicada a Mutação Manual Direcionada em pontos críticos da classe VerificadorData (CC=12).

O objetivo é provar que os testes falham quando o código está incorreto

A suíte de testes unitários demonstrou 100% de sensibilidade aos defeitos injetados.

O Mutante MUT-01 (alteração do limite de ano) foi imediatamente capturado, pois o teste verificarData_quandoDataValida_deveRetornarTrue que esperava um resultado True (data válida) recebeu False, gerando a falha (AssertionFailedError).

Isso valida que os testes não são frágeis e que o código está coberto por testes que verificam a lógica de negócio e os limites de fronteira, impedindo que alterações acidentais de operadores (>= para >) passem despercebidas para a próxima fase do desenvolvimento.

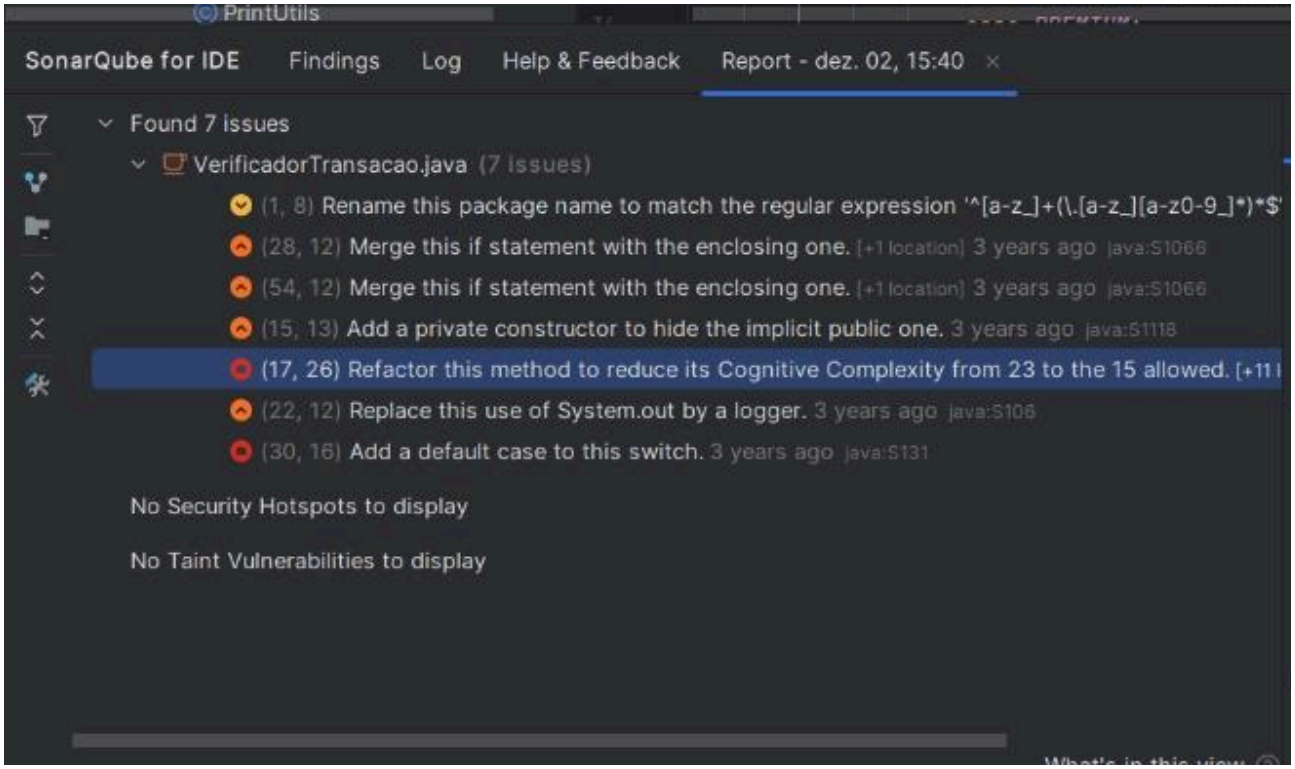
8. Inspeção do código-fonte com Sonar

De maneira resumida, o SonarQube tem o objetivo de analisar a qualidade do código-fonte, onde ele estuda e encontra de forma automática problemas que podem comprometer a manutenibilidade, segurança e confiabilidade do código. Fornece métricas importantes como Maintainability, Reliability, Security e Coverage.

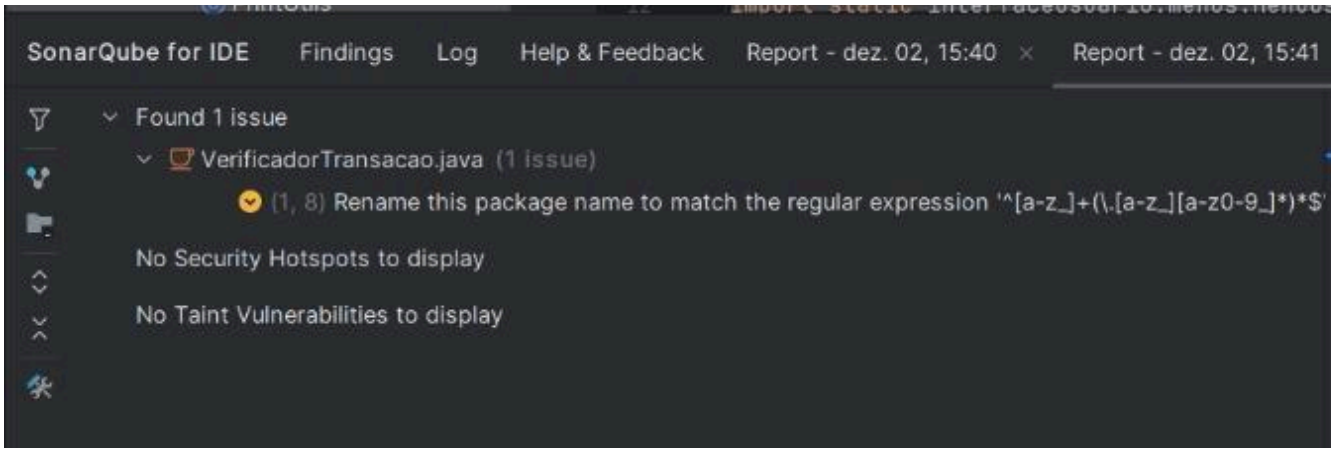
8.1 Daniel Fontoura

O Sonar apontou diversas sugestões de alteração na classe, as quais realizei a correção de todas, exceto uma que envolvia renomear o pacote de interfaceUsuario, para interfaceusuario, o que implicaria em ajustar o nome em todos os demais arquivos da pasta.

Antes:

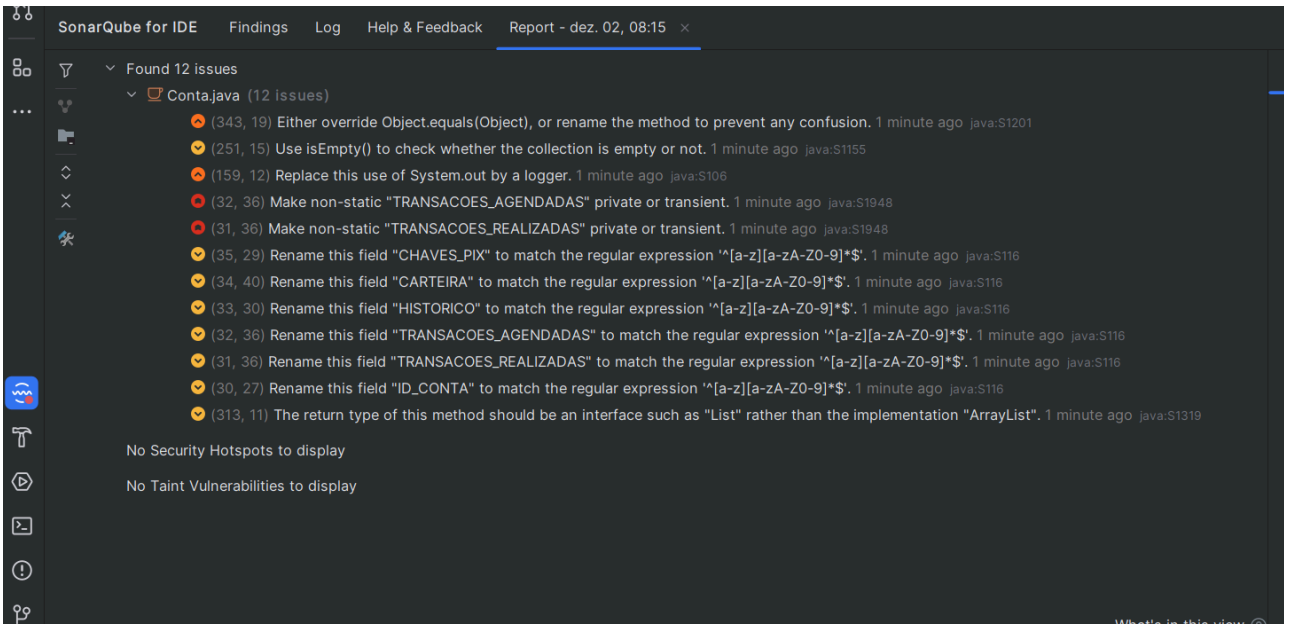


Depois:

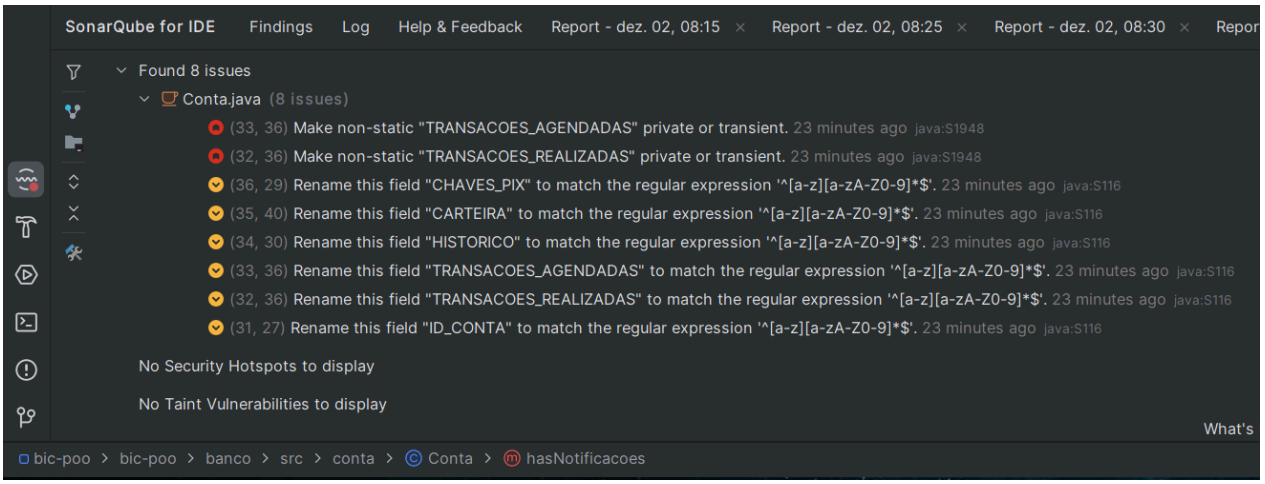


8.2 Daniele Pimenta

Início:



Final:



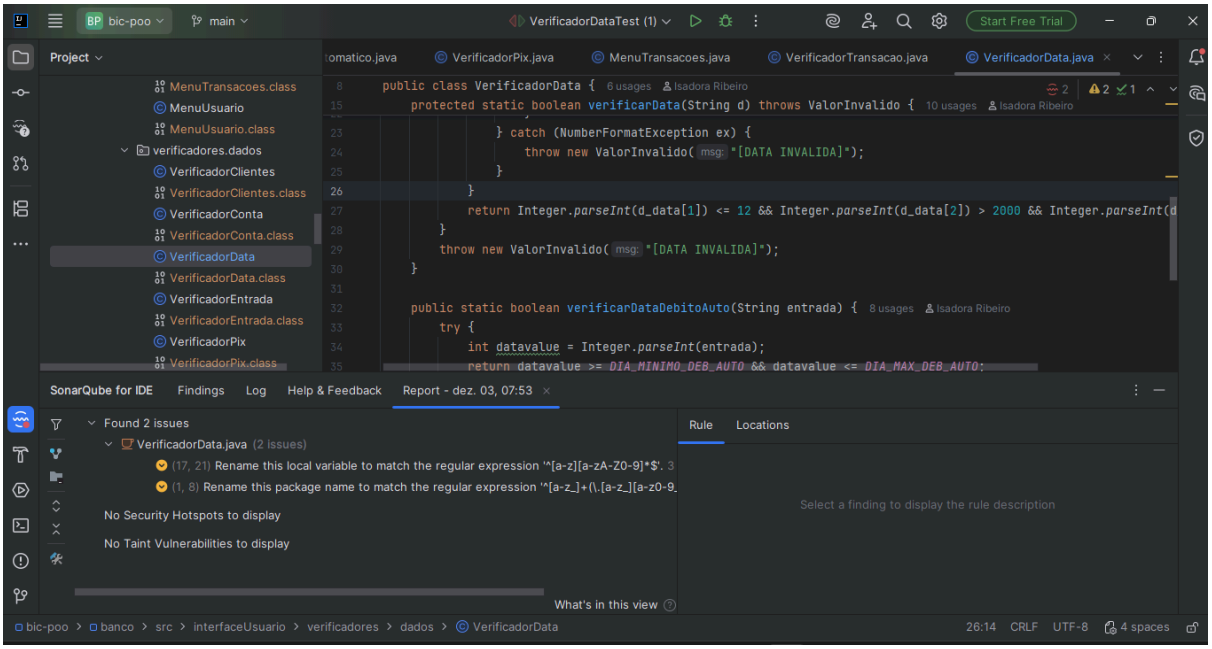
É importante ressaltar que para evitar a quebra de código, eu optei por não realizar os renames que o SonarQube sugeriu.

8.3 João Gabriel Otogali

Classe VerificadorData

Antes:

Depois dos ajustes:

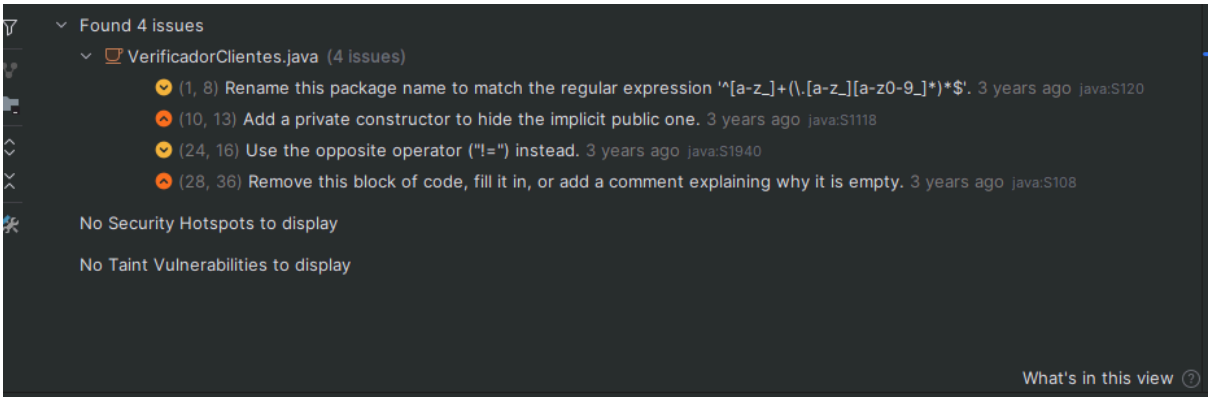


Optei não realizar o rename sugerido pelo Sonar.

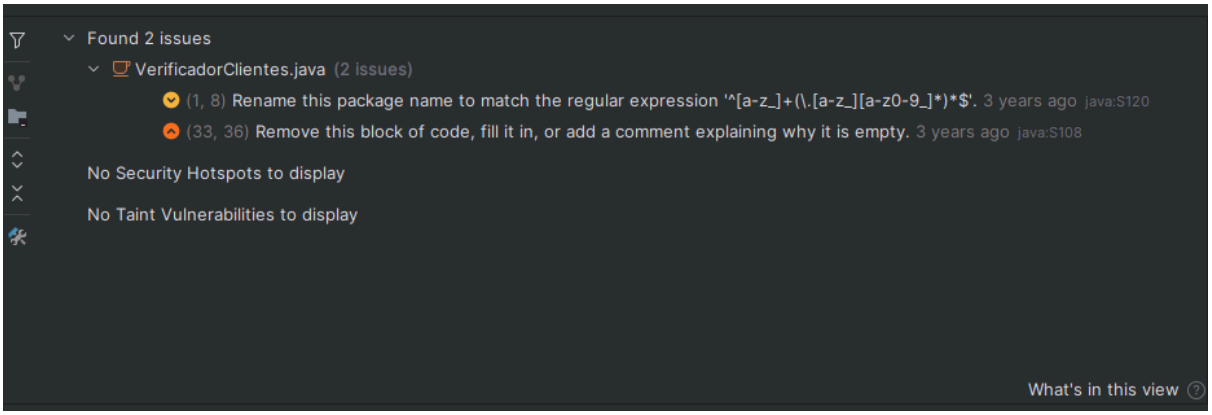
8.4 Irhael Sousa das Chagas

Classe VerificadorClientes

Antes:



Depois:



Para evitar quebra de código e problemas de performance, os package names não foram alterados. Em relação ao bloco vazio, é necessário para o funcionamento do código e não pode ser retirado.

9. Conclusão

A realização desta entrega permitiu consolidarmos diversos conceitos fundamentais de teste de software, de forma prática. Durante os estudos foram encontrados obstáculos pelo caminho, como a impossibilidade de usar selenium, necessitando adaptar o teste de sistema com a lib System Lambda, e também a necessidade de realizar testes de mutação de maneira manual, sem usar PitTest. Mas com determinação conseguimos superar a sua maioria.

10. Links úteis

Repositório antigo do projeto: https://github.com/danhvf/bic-poo/
Repositório Vaniacourses: https://github.com/vaniacourses/trabalho-qt-grupo-bic
Apresentação: Apresentação 2 - Grupo BIC - Google Slides