Entrega 1 do trabalho de Qualidade e Teste - 2025.2

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

1. Descrição do escopo do sistema

Após intensa análise de projetos no repositório disponibilizado pela professora, escolhemos o projeto *Banco BIC* como tema candidato para a realização deste trabalho, dos testes e objetivando pôr em prática tudo que nos foi ensinado em sala de aula.

O *Banco BIC* é uma aplicação orientada a objetos desenvolvida em Java puro, cujo propósito é simular o funcionamento de um banco digital denominado *Banco do Instituto de Computação* – chamaremos de BIC. Navegando no repositório do BIC, é possível observar que é uma aplicação bem trabalhada, unindo conceitos como herança, polimorfismo, encapsulamento e até mesmo tratamento de exceções.

A aplicação possui funcionalidades como criação de contas, realização de transações, geração e pagamento de boletos, operações via Pix, gerenciamento de cartões e empréstimos etc. É importante ressaltar que o projeto atende, também, à complexidade ciclomática mínima exigida, desta forma, unimos o útil ao agradável e demos vida a este trabalho.

2. Ferramentas utilizadas

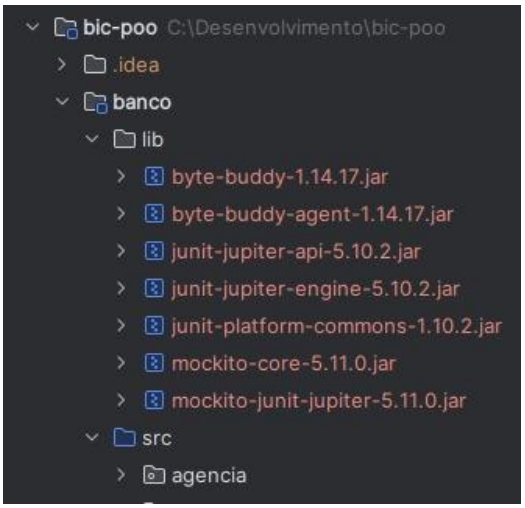
Ferramenta	Função
IntelliJ IDEA	Ambiente de desenvolvimento integrado utilizado para edição e execução do código.
JUnit 5	Framework para realizar os testes unitários e fazer validações de comportamentos.
Mockito	Ferramenta de *mocking* para simular dependências.
TestLink	Ferramenta de apoio para documentação dos casos de testes.
GitHub	Repositório de controle de versão e armazenamento do código-fonte.
Google Docs	Organização e documentação geral do trabalho.

3. Configuração manual do ambiente de testes

Pelo fato do *BIC* ser um projeto Java puro sem Maven/Gradle, foi preciso fazer uma pequena adaptação no ambiente, como a criação de uma nova pasta na raiz do projeto denominada **lib/**, onde baixamos e inserimos os JARs essenciais para o perfeito funcionamento do JUnit 5 e do *Mockito*, como: **junit-jupiter-api**, **junit-jupiter-engine**,

mockito-core, **byte-buddy**, **byte-buddy-agent** e **objenesis**. Após salvar os JARs, foi preciso realizar um caminho específico para a configuração manual do ambiente de testes. No IntelliJ IDEA, faça o seguinte caminho: Menu File → Project Structure → Modules → Dependencies → + → JARs or directories… → Selecionados todos os arquivos .jar da pasta **lib**, clicar em Apply e dar OK. Além disso, tivemos que utilizar o JDK 18, conforme recomendado pelos desenvolvedores.

Seguindo os passos acima obtivemos uma estrutura final mais ou menos assim:



4. Casos de Testes Unitários

Para este projeto a complexidade ciclomática exigida para cada classe foi de maior ou igual a 10, isto é, uma complexidade razoável, utilização de classe não-CRUD e que fosse possível realizar um estudo mais aprofundado nos conceitos dados em sala de aula. Desta forma, analisamos e escolhemos as classes para implementarmos os testes.

Classe	Responsável	Descrição
VerificadorTransacao.java	Daniel Fontoura	Implementa verificações de transações bancárias e validação de regras.
Conta.java	Daniele Pimenta	Realiza operações de conta, com foco em empréstimos.
VerificadorClientes.java	Irhael Chagas	Realiza validações de dados e registros de clientes.
VerificadorPix.java	João Gabriel Otogali	Verifica chaves Pix e regras de transferência instantânea.

5. Responsabilidade de cada integrante

Após o processo de escolha da classe a ser trabalhada, cada membro do grupo ficou responsável por:

- Analisar a lógica da classe de interesse, identificando métodos críticos, fluxos alternativos etc;
- Elaborar casos de teste unitário usando JUnit 5 e criar mocks com *Mockito*, quando necessário;
- Executar os testes de suas classes e documentar os resultados na ferramenta TestLink;
- Gerar evidências de execução e resultados dos métodos testados.

6. Plano de Teste

O relatório de execução do plano de teste foi gerado como PDF pela plataforma TestLink e encontra-se anexo nos links úteis no final deste documento. Nele contém:

1. Identificação e descrição de cada caso de teste;
2. Pré-condições de cada caso de teste, dados de entrada e passos de execução;
3. Resultados esperados e observados pelos testadores;
4. Evidências e logs de execução.

7. Casos de Teste Manuais - TestLink

Nesta etapa utilizamos o TestLink como uma ferramenta auxiliar para documentar, planejar e acompanhar a execução dos testes no Banco BIC. A ferramenta em si não executa código nem roda o teste, contudo, serve para organizar e registrar formalmente todo o processo de testes.

Abaixo especificamos os casos de testes, sua estrutura e autoria. Para uma visão mais detalhada do nosso Suite de Testes, o Relatório de Execução do Plano de Teste deverá ser consultado nos links úteis.

1. **Criação de Conta (Daniele Pimenta)**
   BIC-9: Acessar tela inicial
   BIC-11: Criar cadastro

2. **Movimentação de Conta (Daniele Pimenta)**
   BIC-13: Pagar empréstimo total
   BIC-15: Pedir empréstimo
   BIC-16: Pagar parcela do empréstimo

3. **UC001 (Irhael Chagas)**
   BIC-23: R20: Tentar criar cadastro com campo 'Nome' vazio

4. **UC003 (Daniel Fontoura)**
   BIC-17: R17: Valor não numérico
   BIC-18: R17: Valor negativo
   BIC-19: R17: Depósito Conta Standard
   BIC-20: R17: Depósito Conta Premium
   BIC-21: R17: Depósito Conta Diamond
   BIC-22: R17: Tipo de Conta Inválida

5. **UC004 (João Gabriel Otogali)**
   BIC-2: R9: Gerar um novo boleto com sucesso
   BIC-3: R10: Visualizar boletos gerados

6. **UC005 (João Gabriel Otogali)**
   BIC-4: R11: Validar acesso à área de cartões com senha
   BIC-5: R12: Adicionar um novo cartão de crédito
   BIC-6: R13: Visualizar todos os cartões cadastrados

7. **UC006 (João Gabriel Otogali)**
   BIC-7: Iniciar fluxo de agendamento de transferência
   BIC-8: Negar agendamento para chave PIX inexistente

8. Conclusão

A execução dos testes unitários garantiu que as principais regras de negócio do sistema *Banco BIC* fossem devidamente validadas, envolvendo métodos com diferentes fluxos e tratamento de exceções. Foi possível criar testes mais robustos e isolados graças ao JUnit 5 e *Mockito*, e o uso do TestLink, embora um pouco confuso para quem não conhece a ferramenta, garantiu uma rastreabilidade dos casos de testes e documentação mais detalhada dos resultados desses testes.

9. Links úteis

Apresentação do trabalho: Canva
Relatório de execução do plano de teste: via TestLink
Nosso repositório no Github: danhvf/bic-poo: Banco BIC
Repositório oficial do projeto: Asunnya/bic-poo: Banco BIC