

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SÃO PAULO**

VINÍCIUS SILVA DE SAMPAIO

SISTEMA DE CAIXA ELETRÔNICO

**CAMPOS DO JORDÃO
2024**

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

SISTEMA DE CAIXA ELETRÔNICO

Entrega do projeto de desenvolvimento de um sistema de caixa eletrônico em C++, conforme descrito na 5ª edição do livro do Deitel para disciplina de Programação Orientada a Objetos, do curso de Análise e Desenvolvimento de Sistemas no Instituto Federal de São Paulo de Campos do Jordão como requisito parcial para aprovação na disciplina, sob a orientação do professor Paulo Giovani de Faria Zeferino

CAMPOS DO JORDÃO

2024

RESUMO

Este relatório apresenta o desenvolvimento de um sistema de caixa eletrônico (ATM) em C++, baseado no estudo de caso do livro C++ Como Programar de Deitel. O projeto teve como objetivo simular operações bancárias essenciais, como autenticação, consulta de saldo, saque e depósito, empregando conceitos de Programação Orientada a Objetos (POO) e boas práticas de modularidade. A metodologia seguiu etapas estruturadas, desde o levantamento de requisitos e o design do sistema até a implementação e testes contínuos. O sistema foi desenvolvido de forma incremental, com validação progressiva das funcionalidades para garantir a integração correta entre os componentes. Para o design, foi utilizado o Draw.io para criar o diagrama de classes, e a implementação foi realizada no Visual Studio Code, aproveitando seus recursos avançados para desenvolvimento em C++. Ao final, o sistema atendeu aos requisitos propostos, demonstrando estabilidade e funcionalidade adequada, consolidando o aprendizado prático de conceitos fundamentais de POO e técnicas de desenvolvimento de sistemas robustos.

Palavras-Chave: ATM; C++; Deitel; autenticação; saldo; saques; depósitos; transações bancárias; C++ Como Programar.

ABSTRACT

This report presents the development of an Automated Teller Machine (ATM) system in C++, based on the case study from the book C++ How to Program by Deitel. The project's objective was to simulate essential banking operations, such as authentication, balance inquiry, withdrawal, and deposit, employing Object-Oriented Programming (OOP) concepts and best practices in modularity. The methodology followed structured steps, starting from requirements gathering and system design to implementation and continuous testing. The system was developed incrementally, with progressive validation of functionalities to ensure proper integration between components. For design, Draw.io was used to create the class diagram, and the implementation was carried out in Visual Studio Code, leveraging its advanced features for C++ development. In the end, the system met the proposed requirements, demonstrating stability and proper functionality, consolidating practical learning of fundamental OOP concepts and techniques for developing robust systems.

Keywords: ATM; C++; Deitel; authentication; balance; withdrawals; deposits; banking transactions; C++ How to Program.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – <i>Livro C++: Como Programar</i> (2006)	12
FIGURA 2 – <i>Visual Studio Code</i> (2015)	13
FIGURA 3 – Diagrama de Classes do Sistema ATM (2024)	18
FIGURA 4 – Tela de autenticação do sistema (2024)	23
FIGURA 5 – tela de consulta de saldo (2024)	24
FIGURA 6 – tela de saque realizado (2024)	24

LISTA DE SIGLAS

IDE	<i>Integrated Development Environment</i> (Ambiente de Desenvolvimento Integrado)
POO	<i>Object-Oriented Programming</i> (Programação Orientada a Objetos)
ATM	<i>Automated Teller Machine</i> (Caixa Eletrônico Automatizado)
GCC	<i>GNU Compiler Collection</i> (Coleção de Compiladores GNU)
VS	<i>Visual Studio</i>
PIN	<i>Personal Identification Number</i> (Número de Identificação Pessoal)

SUMÁRIO

1	INTRODUÇÃO _____	8
1.1	Objetivos _____	8
1.2	Justificativa _____	9
1.3	Aspectos Metodológicos _____	10
1.4	Aporte Teórico _____	10
2	FUNDAMENTAÇÃO TEÓRICA _____	11
3	PROJETO PROPOSTO (METODOLOGIA) _____	14
3.1	Considerações Iniciais _____	15
3.2	Descrição do Processo de Desenvolvimento _____	15
4	AVALIAÇÃO _____	22
4.1	Condução _____	22
4.2	Resultados _____	23
4.3	Discussão _____	25
5	CONCLUSÃO _____	26
	REFERÊNCIAS _____	27

1 INTRODUÇÃO

A programação orientada a objetos em C++ proporciona uma base robusta para o desenvolvimento de sistemas complexos e didáticos. Este trabalho tem como objetivo apresentar a construção de um sistema de caixa eletrônico (ATM), seguindo o estudo de caso descrito na 5ª edição do livro C++ Como Programar (DEITEL; DEITEL, 2005). A implementação foi realizada no ambiente *Visual Studio Code*, destacando conceitos fundamentais de orientação a objetos.

O sistema simula operações bancárias essenciais demonstrando a aplicação prática de técnicas de programação estruturada e modular. Além disso, este relatório busca detalhar os passos envolvidos no desenvolvimento, fundamentados nas práticas descritas por Deitel, projetado de forma a proporcionar uma compreensão clara dos conceitos de programação envolvidos. O foco foi em criar uma estrutura funcional e simples, adequando-se ao escopo do estudo e às necessidades do projeto proposto.

1.1 Objetivos

O objetivo principal deste projeto foi o desenvolvimento de um sistema de caixa eletrônico simples, baseado nos conceitos apresentados por Deitel. O sistema foi projetado para simular operações típicas de um caixa eletrônico, como autenticação de usuários, consulta de saldo, saques e depósitos, utilizando a linguagem C++. O foco do projeto foi aplicar conceitos fundamentais de programação, especialmente a programação orientada a objetos, e integrar eficientemente as ferramentas disponíveis no ambiente de desenvolvimento.

Para alcançar esse objetivo, foram estabelecidos os seguintes objetivos específicos:

- Analisar as operações essenciais de um caixa eletrônico, compreendendo as interações entre o usuário e o sistema, bem como as lógicas de transações financeiras;
- Utilizar C++ para implementar a estrutura básica do sistema, explorando conceitos de programação, como classes, objetos e manipulação de dados;

- Aplicar conceitos de programação em C++ para implementar mecânicas de jogo, como movimentação e colisão;
- Desenvolver funcionalidades como autenticação, consulta de saldo e transações financeiras, mantendo o sistema simples e eficiente;
- Utilizar o *Visual Studio Code* como IDE, garantindo um ambiente de desenvolvimento ágil e organizado, e integrar as ferramentas necessárias para o gerenciamento eficiente das dependências;
- Realizar testes do sistema, validando a correta execução das operações e assegurando uma experiência de usuário satisfatória.

1.2 Justificativa

A implementação do sistema de caixa eletrônico descrito no livro C++: Como Programar (Deitel & Deitel, 2005) foi escolhida devido à sua relevância acadêmica e à sua aplicabilidade prática no ensino de conceitos fundamentais de Programação Orientada a Objetos. O estudo de caso proposto na obra permite a exploração de pilares da POO, além de fomentar o uso de boas práticas de modularidade e organização do código, elementos essenciais para o desenvolvimento de sistemas robustos e escaláveis. A abordagem clara e prática do livro, amplamente utilizado em contextos educacionais, oferece uma base sólida para a criação de sistemas que integram teoria e prática, contribuindo significativamente para a formação de desenvolvedores de software.

A escolha do Visual Studio Code como ambiente de desenvolvimento (IDE) foi motivada por sua leveza, flexibilidade e suporte a extensões voltadas para a programação em C++. Ferramentas como autocompletar, depuração integrada e navegação facilitada entre arquivos proporcionaram uma experiência de desenvolvimento ágil e eficiente, ideal para projetos que exigem precisão e organização, como o sistema de caixa eletrônico. Sua interface simples e a integração com sistemas de controle de versão, facilitaram o gerenciamento do código-fonte e o acompanhamento das versões do projeto.

Para a documentação do sistema, foi utilizada a ferramenta Draw.io, que possibilitou a criação de diagramas de classe que representam a estrutura do sistema de

forma visual e detalhada. O uso de diagramas foi essencial para compreender as relações entre as classes e os papéis desempenhados por cada componente, além de auxiliar na validação do design antes da implementação. Essa prática reflete a importância de adotar ferramentas de modelagem durante o desenvolvimento de software, promovendo uma visão mais ampla do ciclo de desenvolvimento e a identificação de possíveis melhorias estruturais.

1.3 Aspectos Metodológicos

A metodologia utilizada para o desenvolvimento do sistema ATM foi baseada em uma abordagem sequencial e modular, com foco na clareza e eficiência. O processo iniciou com o levantamento de requisitos, onde foram definidos os requisitos funcionais e não funcionais, como autenticação de usuários, consulta de saldo, saques e depósitos. Em seguida, foi realizado o projeto do sistema, estruturando-o em componentes lógicos e físicos, como as classes *ATM*, *BankDatabase*, *Transaction*, entre outras.

A implementação foi realizada de forma incremental, iniciando com a criação das classes básicas de dados, como *Account* e *BankDatabase*, seguidas pela construção da interface com o usuário, com as classes *Screen* e *Keypad*, e a implementação das transações (*BalanceInquiry*, *Withdrawal*, *Deposit*). O sistema foi, então, submetido a testes unitários e de integração para validar o funcionamento das funcionalidades e garantir a estabilidade do sistema. O ambiente de desenvolvimento utilizado foi o *Visual Studio Code*, com a extensão C++ instalada, permitindo a execução e depuração do código.

Essa metodologia garantiu o desenvolvimento de um sistema ATM funcional e eficiente, com foco na organização e modularização do código, além de assegurar a validação de todas as funcionalidades principais através de testes contínuos.

1.4 Aporte Teórico

Este relatório está estruturado em seções que abordam os principais aspectos do desenvolvimento do sistema ATM. A Seção 2 apresenta a fundamentação teórica,

discutindo conceitos essenciais como modelagem de dados e boas práticas de programação. A Seção 3 descreve a metodologia, detalhando as etapas de desenvolvimento, desde o levantamento de requisitos até os testes. Na Seção 4, são apresentados os resultados, destacando a funcionalidade e o desempenho do sistema. Por fim, a Seção 5 traz a conclusão, resumindo os objetivos alcançados e sugerindo melhorias para futuras versões.

2 FUNDAMENTAÇÃO TEÓRICA

O livro C++: Como Programar, de Deitel, 5ª edição, é uma referência reconhecida no ensino da linguagem C++, destacando-se por sua abordagem prática e didática. Voltado para iniciantes e programadores experientes, combina explicações teóricas detalhadas com exemplos aplicáveis, promovendo uma compreensão sólida dos fundamentos e conceitos avançados da programação orientada a objetos.

A obra inclui estudos de caso completos, como o sistema de caixa eletrônico, utilizado neste relatório, que integra conceitos de POO em um contexto prático. Além disso, apresenta exercícios desafiadores e boas práticas de programação, tornando-se essencial para estudantes e profissionais que desejam consolidar habilidades técnicas e desenvolver sistemas robustos e organizados.



Figura 1 – Livro C++: Como Programar (2006)

A Programação Orientada a Objetos é uma abordagem amplamente utilizada no desenvolvimento de software, oferecendo conceitos como encapsulamento, herança e polimorfismo, que auxiliam na criação de sistemas estruturados e organizados. Esses pilares são essenciais para a construção de soluções reutilizáveis e modulares, sendo aplicados no desenvolvimento do sistema ATM descrito no livro C++: Como Programar (Deitel & Deitel, 2005). O encapsulamento permite organizar os dados em classes e ocultar detalhes internos, como ocorre em *Account* e *BankDatabase*, que manipulam informações bancárias de forma segura. A herança viabiliza o reaproveitamento e a especialização de código, visível na classe base *Transaction* e suas subclasses (*BalanceInquiry*, *Withdrawal* e *Deposit*), promovendo organização e extensibilidade no sistema. Já o polimorfismo permite tratar objetos de diferentes classes de forma uniforme, como no gerenciamento de transações via ponteiros para *Transaction*.

Além dos fundamentos de POO, a modularidade também desempenha um papel crucial no desenvolvimento de sistemas. A divisão do sistema em componentes me-

nores, como classes independentes e especializadas, facilita a manutenção, a legibilidade e a escalabilidade do código. Essa prática é reforçada pelo uso de ferramentas de documentação e modelagem, como o *Draw.io*, que permite a criação de diagramas de classe para visualizar as relações entre os componentes do sistema e validar o design antes da implementação.

A escolha do *Visual Studio Code* como ambiente de desenvolvimento reflete a importância de ferramentas que promovam eficiência e organização durante o processo de codificação. Recursos como depuração integrada, extensões para C++ e integração com sistemas de controle de versão facilitam o desenvolvimento e o gerenciamento de projetos de programação, permitindo ao desenvolvedor focar na implementação dos conceitos estudados.

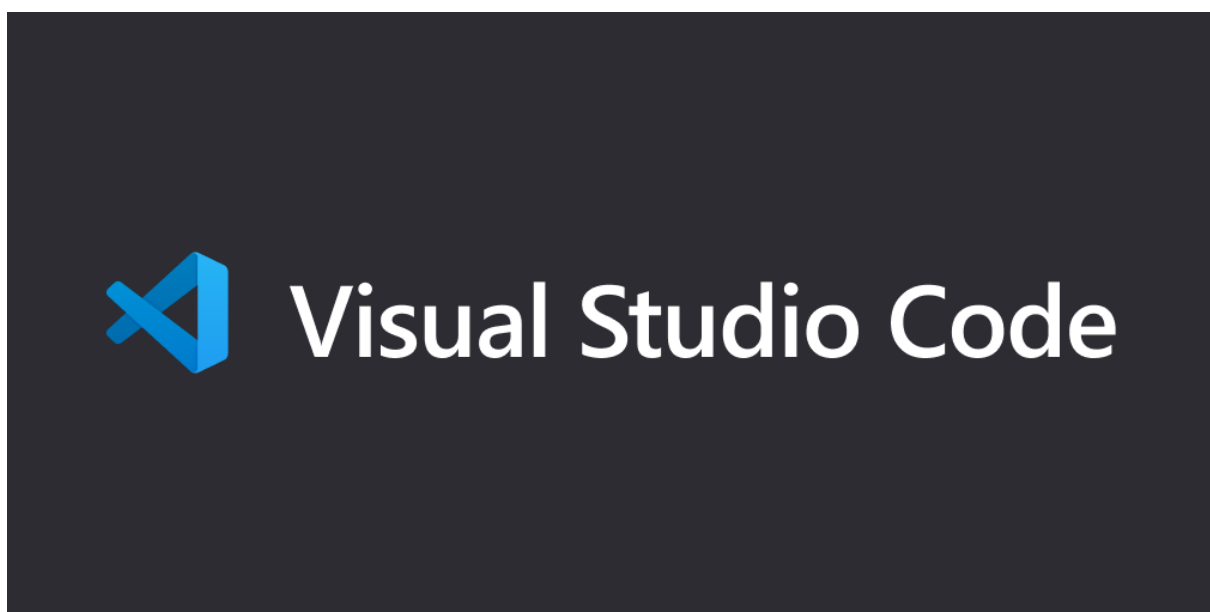


Figura 2 – *Visual Studio Code* (2015)

3 PROJETO PROPOSTO

A metodologia adotada para o desenvolvimento do sistema ATM foi planejada de forma a garantir uma abordagem clara, eficiente e estruturada, permitindo a criação de um sistema funcional e de fácil manutenção. O processo de desenvolvimento foi dividido em etapas sequenciais, cada uma com um conjunto de atividades e objetivos bem definidos. Inicialmente, foi realizado o levantamento de requisitos, identificando as funcionalidades essenciais para o sistema, como autenticação, consulta de saldo, saques e depósitos.

Após a definição dos requisitos, o projeto do sistema foi desenvolvido com base em uma arquitetura modular, visando facilitar a implementação e a futura escalabilidade do código. A implementação foi dividida em fases incrementais, com cada etapa focada em um conjunto específico de funcionalidades, como o gerenciamento das contas bancárias, a interface do usuário e a lógica das transações bancárias.

Além disso, a metodologia incluiu a realização de testes unitários e de integração ao longo de todo o processo de desenvolvimento, garantindo que cada componente do sistema funcionasse corretamente e que as interações entre os componentes fossem eficazes. O ambiente de desenvolvimento utilizado foi o *Visual Studio Code*, com a extensão C++ instalada, o que permitiu o desenvolvimento e a depuração do código de forma eficiente.

Essa abordagem metodológica permitiu a construção de um sistema ATM robusto e confiável, atendendo aos requisitos propostos e garantindo a validação de todas as funcionalidades essenciais do sistema. A seguir, são detalhadas as etapas que compõem essa metodologia, abordando desde a análise de requisitos até os testes e validações realizados.

3.1 Considerações Iniciais

Para o desenvolvimento do sistema ATM, foi necessário configurar o ambiente de programação com suporte à linguagem C++. O *Visual Studio Code* (VS Code) foi escolhido como editor de código, e a extensão C/C++ da Microsoft foi instalada para fornecer recursos como *IntelliSense*, depuração e realce de sintaxe.

Além disso, foi necessário configurar o compilador C++. Para isso, foi utilizado o *MinGW*, um compilador GCC adaptado para Windows. Após a instalação do *MinGW*, foi necessário adicionar o caminho do compilador ao *PATH* do sistema para garantir que o VS Code pudesse acessá-lo facilmente. Essa configuração permitiu compilar e executar o código diretamente do editor. Para garantir o funcionamento adequado do ambiente, um arquivo de configuração do VS Code foi criado, definindo a execução do compilador e a depuração do código, permitindo compilar e testar o sistema ATM sem problemas.

3.2 Descrição do Processo de Desenvolvimento

Na fase inicial do desenvolvimento, foi realizado o levantamento dos requisitos do sistema ATM, com o objetivo de entender de maneira clara as funcionalidades que o sistema deveria oferecer, bem como as condições de operação. Durante essa etapa, foram identificados os requisitos que serviriam de base para o desenvolvimento do sistema.

1. Requisitos do Sistema:

- **Autenticação de Usuário:** O sistema deveria autenticar o usuário por meio de um número de conta e PIN, garantindo a segurança e a privacidade das transações realizadas.
- **Consulta de Saldo:** O sistema deveria permitir que o usuário consultasse o saldo disponível de sua conta bancária, bem como o total de saldo em sua conta, com a devida exibição na interface de usuário.

- **Realização de Saques e Depósitos:** O sistema deveria permitir que o usuário realizasse transações bancárias simples, como saques, com verificação de saldo e depósitos, com validação do envelope de depósito.

A partir desses requisitos, foram identificadas as principais entidades do sistema (como contas bancárias, transações, e componentes do ATM), bem como as operações necessárias para atender a cada um desses requisitos. Essa análise orientou a criação do projeto do sistema, garantindo que todos os aspectos funcionais e não funcionais fossem atendidos.

2. Projeto do Sistema:

O projeto do sistema seguiu uma abordagem orientada a objetos e foi baseado no diagrama de classes apresentado no livro "C++ Como Programar" para o caso específico do ATM.

Componentes Físicos:

- **ATM:** A classe principal do sistema, responsável por controlar a execução das operações e coordenar a interação entre os demais componentes.
- **Screen (Tela):** Responsável por exibir informações ao usuário, como o saldo da conta ou mensagens de erro.
- **Keypad (Teclado):** Permite que o usuário insira dados, como número da conta e PIN, além de selecionar as opções de transações.
- **DepositSlot (Slot de Depósito):** Simula o espaço onde o usuário deposita envelopes de depósito.
- **CashDispenser (Dispenser de Dinheiro):** Simula a entrega de dinheiro para saques.

Componentes Lógicos:

- **BankDatabase (Banco de Dados):** Responsável pelo armazenamento e gerenciamento das contas bancárias, com métodos para verificar saldo, realizar depósitos e saques. Para utilizar o tipo `size_t`, que é necessário

para representar tamanhos de objetos na memória, foi incluído o cabeçalho `#include <cstdint>`, garantindo o uso correto dessa estrutura no controle das transações.

- *Account* (Conta Bancária): Cada conta bancária é representada por uma instância dessa classe, que armazena dados como número da conta, PIN e saldo.
- *Transaction* (Transação): Classe base para todas as operações bancárias, que são modeladas como transações independentes. As subclasses de *Transaction* incluem:
 - *BalanceInquiry* (Consulta de Saldo): Realiza a consulta do saldo do cliente.
 - *Withdrawal* (Saque): Realiza o saque, verificando se o saldo é suficiente e se o caixa possui notas disponíveis.
 - *Deposit* (Depósito): Realiza o depósito, validando o envelope de depósito.

O diagrama de classes foi estruturado de maneira a garantir uma boa separação de responsabilidades e facilitar a manutenção e expansão do sistema no futuro.

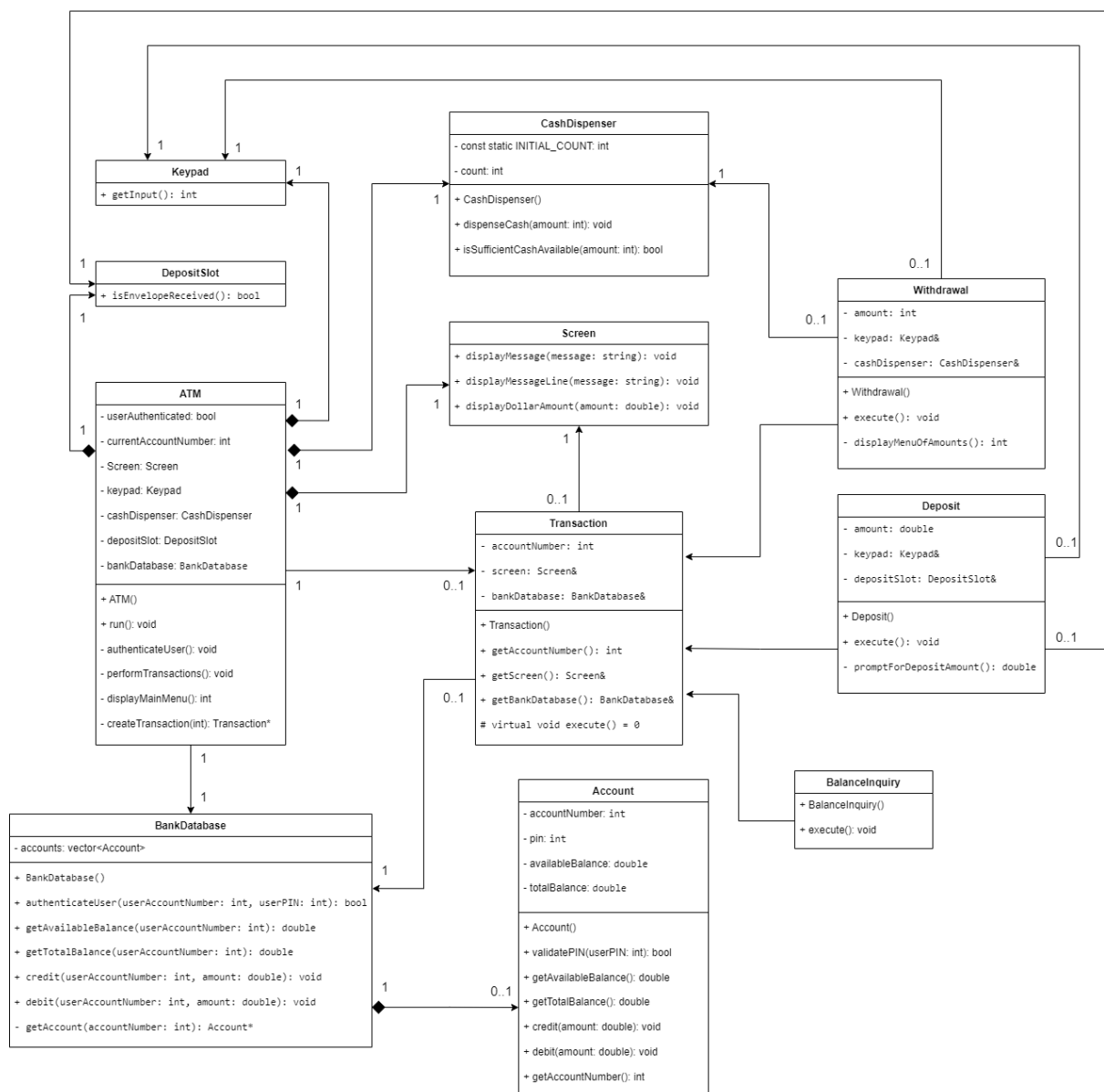


Figura 3 – Diagrama de Classes do Sistema ATM (2024)

3. Implementação Modular

A implementação do sistema seguiu um fluxo incremental e iterativo, em que as funcionalidades foram desenvolvidas e validadas progressivamente, a cada nova etapa de desenvolvimento. O objetivo era garantir que cada parte do sistema funcionasse corretamente antes de passar para a próxima fase.

Primeira Etapa: Implementação das Classes Básicas

Na primeira etapa, foram desenvolvidas as classes *Account* e *BankDatabase*. Essas classes são responsáveis por armazenar e gerenciar os dados bancários, como o número da conta, o PIN do cliente e o saldo. Além disso, implementaram-se os métodos que validam o PIN do usuário e verificam o saldo disponível para as transações de saque e depósito. Essa fase foi essencial para validar os mecanismos de autenticação e manipulação de dados.

Segunda Etapa: Implementação da Interface com o Usuário

A segunda etapa consistiu em desenvolver o sistema de interface com o usuário, representado pelas classes *Screen* e *Keypad*. A classe *Screen* foi responsável por exibir as informações ao usuário, enquanto o *Keypad* capturou as entradas do usuário (como o número da conta, PIN e a escolha de transação). A interação entre essas classes foi cuidadosamente projetada para garantir uma experiência de uso intuitiva e eficiente.

Terceira Etapa: Implementação das Transações

A terceira etapa envolveu a criação da classe *Transaction* e suas subclasses (*BalanceInquiry*, *Withdrawal* e *Deposit*). Cada tipo de transação foi modelado de forma independente, seguindo o princípio da responsabilidade única, ou seja, cada classe foi responsável por uma operação específica. A classe *Withdrawal*, por exemplo, foi responsável por verificar se o saldo era suficiente para o saque e se o *CashDispenser* tinha dinheiro disponível, enquanto a classe *Deposit* validava a recepção do envelope de depósito.

Quarta Etapa: Integração dos Componentes Físicos

Após a implementação das funcionalidades lógicas, foi realizada a integração dos componentes físicos simulados, como o *CashDispenser* e o *DepositSlot*. Nessa etapa, as operações de saque e depósito passaram a interagir diretamente com essas classes, simulando o comportamento do hardware do caixa eletrônico, como a entrega de cédulas ou o recebimento de envelopes de depósito.

Quinta Etapa: Desenvolvimento do Controlador Principal

A última etapa do processo de implementação foi a criação da classe ATM, que atua como o controlador principal do sistema. A classe ATM coordena todas as interações entre os componentes, como o *BankDatabase*, *Screen*, *Keypad* e as diferentes Transações, coordenando o fluxo de execução de forma integrada.

4. Testes e Validação

Os testes foram realizados de forma estruturada, com o objetivo de garantir a qualidade e robustez do sistema. Foram realizadas duas fases de testes: testes unitários e testes de integração.

Testes Unitários:

Na fase de testes unitários, cada classe foi testada individualmente para verificar se suas funcionalidades estavam sendo executadas corretamente. Exemplos de testes unitários incluem a verificação de:

- A autenticação correta do usuário.
- A verificação de saldo na classe *Account*.
- O cálculo de saque e depósito nas classes *Withdrawal* e *Deposit*.

Testes de Integração:

Após a implementação das classes, o sistema foi testado como um todo. Os testes de integração verificaram o fluxo completo de operações, desde a autenticação até a realização de transações de saque e depósito. Durante essa fase, foram simulados cenários como "Numero de conta ou PIN invalido. Tente novamente.", onde o usuário fornecia um número de conta ou PIN incorretos.

Com todos os componentes implementados e testados, o sistema foi integrado e executado, garantindo que todas as transações bancárias fossem realizadas corretamente e que o fluxo de operações estivesse em conformidade com os requisitos do sistema.

Após implementar todas as funcionalidades do sistema ATM, o próximo passo foi compilar o código para gerar o executável. Utilizando o compilador g++ no terminal do *Visual Studio Code*, o seguinte comando foi executado:

```
g++ ATMCaseStudy.cpp ATM.cpp Account.cpp BalanceInquiry.cpp BankData-  
base.cpp CashDispenser.cpp Deposit.cpp DepositSlot.cpp Keypad.cpp Screen.cpp  
Transaction.cpp Withdrawal.cpp -o PROGRAMA
```

Esse comando compilou todos os arquivos de código-fonte necessários e gerou o executável PROGRAMA.exe. Para rodar o sistema, foi utilizado o comando:

```
.\PROGRAMA.exe
```

Esse processo permitiu realizar o teste final do sistema, verificando o funcionamento de todas as funcionalidades, como autenticação, consulta de saldo, saques e depósitos. O teste confirmou que o sistema estava funcionando corretamente e atendia aos requisitos estabelecidos.

4 AVALIAÇÃO

A avaliação do sistema ATM foi realizada para verificar se todas as funcionalidades essenciais, como autenticação de usuários, consulta de saldo, saques e depósitos, funcionaram corretamente. Durante os testes, foi possível validar que as transações eram realizadas de acordo com os requisitos, sem falhas nas funcionalidades principais. O sistema apresentou boa responsividade, com uma interface clara e fácil de interagir.

Além disso, a implementação de funcionalidades adicionais, como bloqueio de conta após várias tentativas de autenticação falhas, pode ser útil para tornar o sistema mais robusto. De modo geral, o sistema atendeu aos objetivos propostos, oferecendo uma experiência funcional e confiável para as operações bancárias.

4.1 Condução

Os testes foram planejados para avaliar a funcionalidade e a qualidade do sistema de forma abrangente. Inicialmente, foram definidos critérios chave, como a precisão das operações bancárias, a resposta do sistema a erros de entrada e a sua estabilidade durante a execução de transações.

Os testes foram realizados utilizando o *Visual Studio Code*, com a extensão C++ devidamente configurada. Durante o processo, foram simulados diversos cenários de uso para observar como o sistema interage com o usuário, além de verificar o desempenho e a capacidade de resposta do sistema tanto a entradas válidas quanto inválidas. Também foram feitos testes de validação para garantir que o sistema tratasse corretamente transações como saques acima do saldo disponível, assegurando que as regras de negócio fossem seguidas corretamente.

4.2 Resultados

Nesta seção, são apresentados os resultados alcançados ao longo do desenvolvimento e dos testes do sistema ATM, evidenciando o funcionamento das funcionalidades implementadas. Inicialmente, foi conduzida uma série de testes para validar as operações principais do sistema. As imagens a seguir demonstram a execução dessas funcionalidades, confirmando que a lógica de transações foi corretamente desenvolvida e integrada ao sistema.

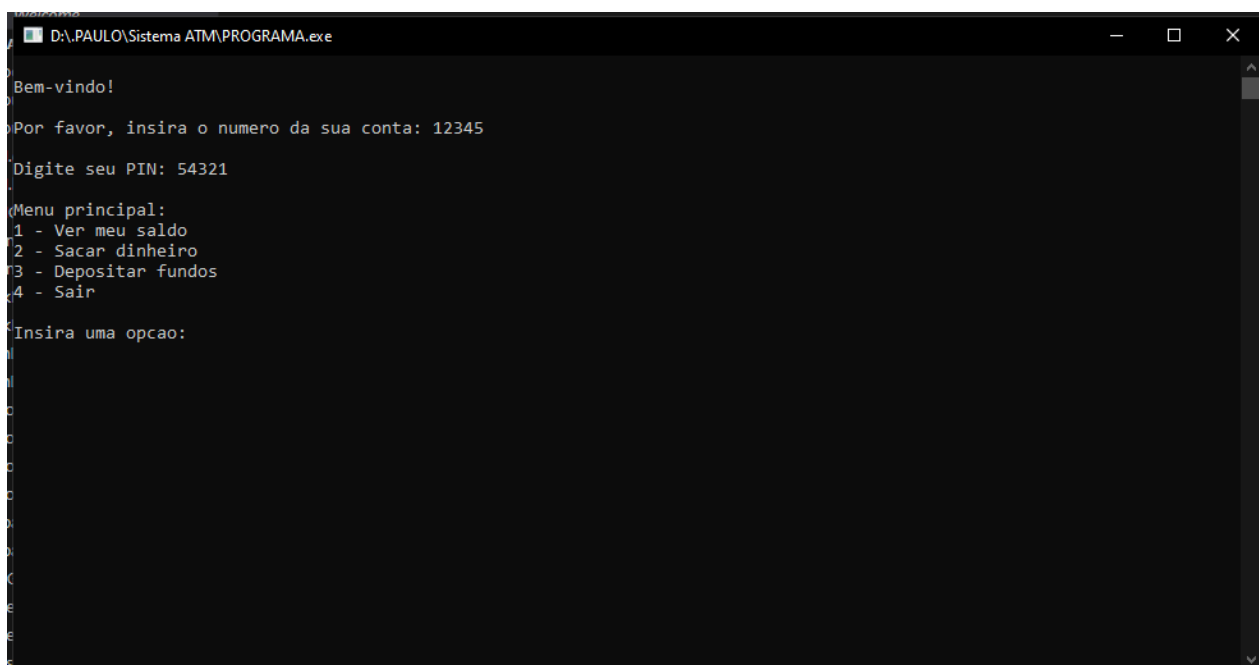
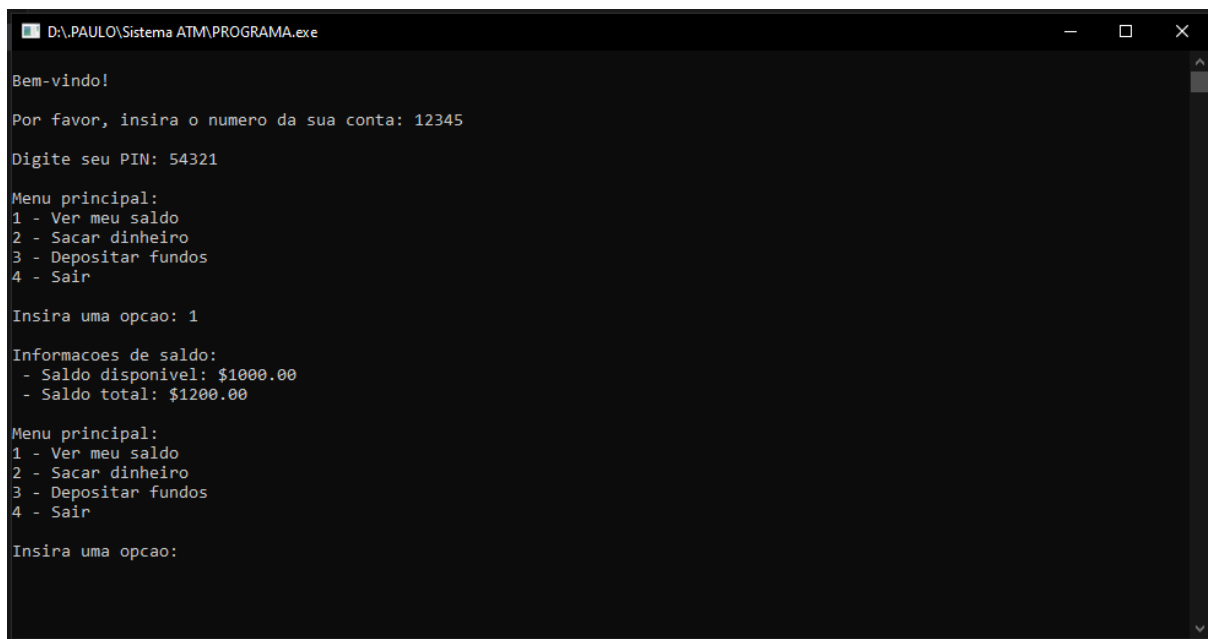


Figura 4 – Tela de autenticação do sistema (2024)

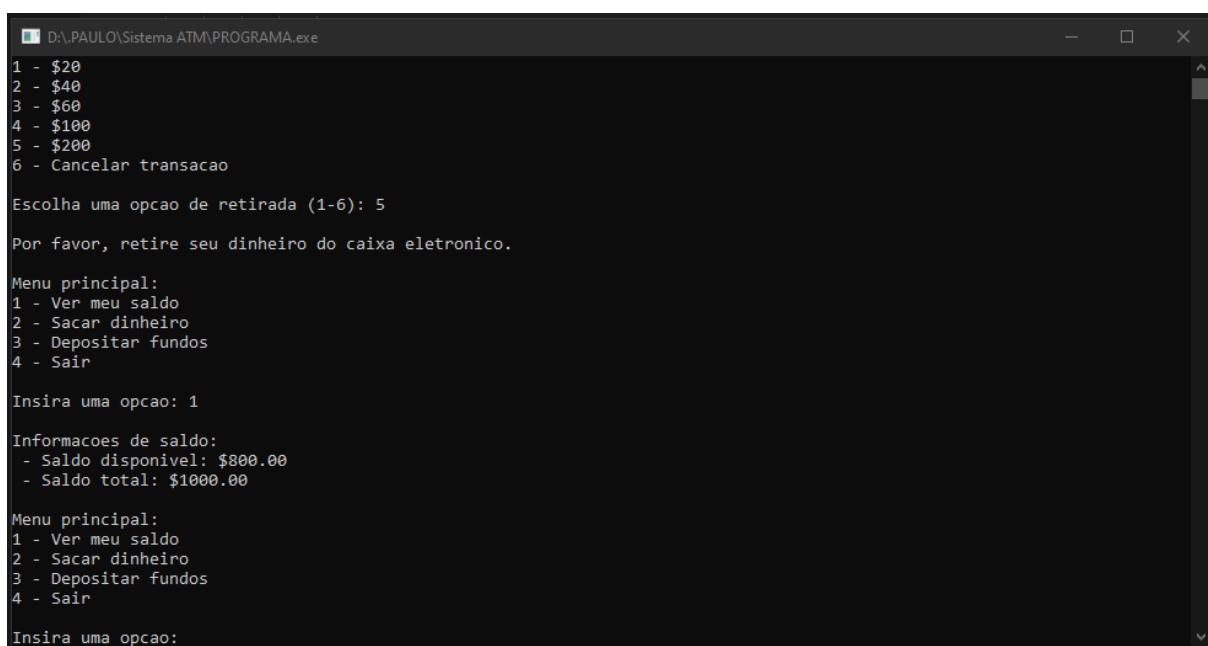
A imagem mostra a tela inicial de autenticação, onde o usuário insere seu número de conta e PIN. A autenticação foi validada com sucesso, permitindo que o sistema verificasse as credenciais corretamente e redirecionasse o usuário para a próxima etapa.



```
D:\PAULO\Sistema ATM\PROGRAMA.exe
Bem-vindo!
Por favor, insira o numero da sua conta: 12345
Digite seu PIN: 54321
Menu principal:
1 - Ver meu saldo
2 - Sacar dinheiro
3 - Depositar fundos
4 - Sair
Insira uma opcao: 1
Informacoes de saldo:
- Saldo disponivel: $1000.00
- Saldo total: $1200.00
Menu principal:
1 - Ver meu saldo
2 - Sacar dinheiro
3 - Depositar fundos
4 - Sair
Insira uma opcao:
```

Figura 5 – Tela de consulta de saldo (2024)

Aqui, é apresentada a tela de consulta de saldo, demonstrando que o sistema retornou corretamente o saldo disponível na conta do usuário. Esse teste validou a precisão da consulta de saldo, funcionando conforme o esperado.



```
D:\PAULO\Sistema ATM\PROGRAMA.exe
1 - $20
2 - $40
3 - $60
4 - $100
5 - $200
6 - Cancelar transacao
Escolha uma opcao de retirada (1-6): 5
Por favor, retire seu dinheiro do caixa eletronico.
Menu principal:
1 - Ver meu saldo
2 - Sacar dinheiro
3 - Depositar fundos
4 - Sair
Insira uma opcao: 1
Informacoes de saldo:
- Saldo disponivel: $800.00
- Saldo total: $1000.00
Menu principal:
1 - Ver meu saldo
2 - Sacar dinheiro
3 - Depositar fundos
4 - Sair
Insira uma opcao:
```

Figura 6 – Tela de saque realizado (2024)

A imagem ilustra a tela de saque, onde o usuário realiza a transação. O sistema corretamente verificou o saldo disponível e permitiu a retirada de valores dentro do limite da conta, conforme especificado nos requisitos.

4.3 Discussão

A análise dos resultados demonstrou que o sistema ATM atendeu aos requisitos definidos durante o planejamento, com um desempenho consistente e sem falhas críticas nas operações principais. A implementação de transações como saque, depósito e consulta de saldo foi bem-sucedida, com a correta validação dos dados de entrada, o que garantiu a execução precisa de cada operação.

Embora o sistema tenha funcionado adequadamente nas condições testadas, alguns pontos podem ser aprimorados para aumentar sua robustez. Por exemplo, a parte de segurança poderia ser reforçada com a inclusão de um sistema de bloqueio temporário após várias tentativas incorretas de autenticação, prevenindo acessos indevidos.

A estabilidade do sistema durante os testes foi satisfatória, com a aplicação respondendo de maneira rápida e sem problemas, mesmo com múltiplas operações realizadas em sequência.

termos de ambiente de desenvolvimento, o *Visual Studio Code* mostrou-se uma escolha sólida, com a extensão C++ configurada corretamente. A utilização dessa ferramenta proporcionou um ambiente de desenvolvimento ágil e adequado para a construção e testes do sistema. A integração entre os componentes foi bem-sucedida, e a abordagem modular adotada, dividindo o sistema em diferentes classes, facilitou a manutenção e compreensão do código.

5 CONCLUSÃO

O desenvolvimento do sistema ATM foi concluído com êxito, alcançando todos os objetivos estabelecidos no início do projeto. O sistema foi projetado para implementar as suas devidas operações essenciais, e esses objetivos foram atendidos de forma satisfatória. A arquitetura modular e a utilização de classes específicas para representar cada componente, como a interface com o usuário e o banco de dados, permitiram a criação de uma solução organizada, funcional e de fácil manutenção.

Durante os testes, o sistema demonstrou ser preciso e confiável, com um desempenho estável e adequado em todos os cenários testados. As transações foram executadas corretamente, e as respostas a erros de entrada, como tentativas de saques acima do saldo disponível, foram tratadas adequadamente.

Embora o sistema tenha atendido plenamente aos requisitos, algumas áreas podem ser aprimoradas, como a segurança e a experiência do usuário. A implementação de novos recursos, como bloqueios após tentativas de autenticação incorretas e a adição de mensagens de erro mais detalhadas, pode tornar o sistema mais robusto e amigável.

Em resumo, os objetivos iniciais do projeto foram plenamente alcançados, com a entrega de um sistema funcional e eficiente. As melhorias identificadas servem como oportunidades para futuras versões, que poderão agregar novas funcionalidades e aprimorar a experiência do usuário.

REFERÊNCIAS

A. ONLINE:

DRAW.IO. Disponível em: <<https://app.diagrams.net/>>. Acesso em: 6 dez. 2024.

MICROSOFT. *Visual Studio Code*. Disponível em: <<https://code.visualstudio.com/>>. Acesso em: 6 dez. 2024.

SOURCEFORGE. *MinGW: Minimalist GNU for Windows*. Disponível em: <<https://sourceforge.net/projects/mingw/>>. Acesso em: 6 dez. 2024.

MICROSOFT. *cstddef (C++ Standard Library)*. Disponível em: <<https://learn.microsoft.com/pt-br/cpp/standard-library/cstddef?view=msvc-170>>. Acesso em: 7 dez. 2024.

FIGUEIREDO, D. *UML: Diagrama de classes e relacionamentos*. Disponível em: <https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagrama-classes-relacionamentos_v01.pdf>. Acesso em: 6 dez. 2024.

B. Livro:

DEITEL, H. M.; DEITEL, P. J. *C++ como programar*. 5. ed. São Paulo: Pearson Prentice Hall, 2006.