

# Programação Orientada a Objetos: Operações Bancárias em C++

Projeto Final - Programação III

Rubens Heryson de Lima Lavor  
Centro Tecnológico de Joinville  
Universidade Federal de Santa Catarina-UFSC  
Joinville, Brasil  
rubens.lavor@grad.ufsc.br

Thaiz Antunes Izidoro  
Centro Tecnológico de Joinville  
Universidade Federal de Santa Catarina-UFSC  
Joinville, Brasil  
thaiz.izidoro@grad.ufsc.br

**Resumo**—Este projeto visa demonstrar os principais conceitos de orientação a objeto usando a linguagem C++; são eles: Classe, Encapsulamento, Agregação e Composição, Templates, Sobrecarga de operador, Herança e Polimorfismo. Por meio de operações bancárias tais como abertura de contas, transferências, demonstrativos bancários (extratos), entre outros.

**Index Terms**—C++, Programação, Orientação a Objetos, Operações Bancárias.

## I. INTRODUÇÃO

Este relatório tem o objetivo de detalhar a lógica de funcionamento do programa, apresentar as principais classes e demonstrar os conceitos usados para o desenvolvimento das operações do sistema bancário, feito para o trabalho final da disciplina de programação 3. Todo o código está documentado nos arquivos de cabeçalho.

## II. DESENVOLVIMENTO

### A. Classe Conta

A classe Conta define um tipo de dado abstrato para a criação da estrutura de classes de contas bancárias, é a estrutura central do projeto e classe base para as classes derivadas ContaComum, ContaEspecial e ContaPoupanca, ou seja, estas três, herdam as funcionalidades de Conta. Nela encontramos atributos e métodos comuns a todas as contas (e o que esperamos que elas tenham no mundo real). Exemplos de atributos presentes: senha, número da conta, saldo; exemplos de métodos: depositar, sacar, entre outras.

Conceitos fundamentais de OO a partir da Classe Conta:

- 1) **Encapsulamento**: Em linguagens orientadas a objeto, é a capacidade de ocultação de detalhes de implementação por parte de entidades de manipulação de dados, por meio dos especificadores de acesso, em C++ são três: **public**, **private** e **protected**. Cada atributo oferece um nível de ocultação para membros de classes.
- 2) **Agregação**: Conta possui uma referência para a classe Pessoa, trata-se de uma associação na forma de agregação, pois o objeto do tipo Pessoa não deixa de

```
1 #ifndef CONTA_H
2 #define CONTA_H
3 #include <iostream>
4 #include <string>
5 #include "pessoa.h"
6
7 class Conta {
8     private:
9         static int contasCriadas;
10        std::string senha = "";
11    protected:
12        int numero;
13        Pessoa *correntista;
14        float saldo{};
15    public:
16        Conta();
17        Conta(int numero, Pessoa &correntista, float saldo, std::string senha);
18        virtual ~Conta();
19
20        bool validacao(std::string chave) const;
21        int getNumero();
22        void setNumero(int num);
23        Pessoa getCorrentista();
24        void setCorrentista(Pessoa &correntista);
25        float getSaldo();
26        void setSaldo(float saldo);
27        int getNumeroTotalDeContas();
28        bool movimentar(float valor, int operacao);
29        void depositar(float valor);
30
31        virtual void info() const;
32        virtual bool sacar(float valor);
33 };
34 #endif
```

Figura 1. Arquivo de cabeçalho conta.h

existir quando o objeto do tipo Conta, associado a ele, é destruído. Com isso um objeto Conta, por meio dos métodos presentes na classe Pessoa, pode ter acesso ao nome e CPF do correntista.

- 3) **Herança**: Herança é um dos pontos chave de programação orientada a objetos. Ela fornece meios de promover a extensibilidade do código, a reutilização e uma maior coerência lógica no modelo de implementação.
- 4) **Polimorfismo**: O polimorfismo em C++ se apresenta sob diversas formas diferentes, desde as mais simples, como funções com mesmo nome e lista de parâmetros diferentes, até as mais complexas como funções virtuais, cujas formas de execução são dependentes da classe a qual o objeto pertence e são identificadas em tempo de execução. No método sacar, por exemplo, o polimorfismo é necessário, pois a classe ContaEspecial permite

```

1 #ifndef PESSOA_H
2 #define PESSOA_H
3 #include <iostream>
4 #include <string>
5
6 class Pessoa {
7     private:
8         std::string nome;
9         std::string CPF;
10    public:
11        Pessoa();
12        Pessoa(std::string _nome, std::string _CPF);
13        ~Pessoa();
14        void setNome(std::string _nome);
15        void setCPF(std::string _CPF);
16        std::string getNome();
17        std::string getCPF();
18 };
19 #endif

```

Figura 2. Arquivo de cabeçalho pessoa.h

```

1 #ifndef CONTAPOUPANCA_H
2 #define CONTAPOUPANCA_H
3 #include "conta.h"
4 #include "taxa.h"
5
6 class ContaPoupanca : public Conta, public Taxa {
7     private:
8         //Não há atributos em ContaPoupanca além dos definidos nas classes Conta e Taxa
9     public:
10        ContaPoupanca(int numero, Pessoa &correntista, float saldo);
11        ~ContaPoupanca();
12        virtual void Info() const override;
13        virtual void incremento_juros() override;
14        virtual void descontarTaxaManutencao() override{};
15 };
16
17 #endif

```

Figura 3. Arquivo de cabeçalho contaPoupanca.h

que o correntista saque um valor além do saldo, trata-se de um limite especial que esse tipo de conta oferece. Mesmo com o saldo zerado, é possível realizar um saque até determinado valor. Isso já não acontece, por exemplo, em ContaPoupanca, onde não é possível sacar valores além do saldo em conta, cada classe apresenta um comportamento distinto para a funcionalidade sacar.

## B. Classe Movimento

A classe Movimento registra todas as atividades de uma determinada conta. Sempre que uma nova transação é realizada, um objeto Movimento é criado e associado a conta correspondente.

```

1 #ifndef MOVIMENTO_H
2 #define MOVIMENTO_H
3 #include "conta.h"
4
5 class Movimento {
6     private:
7         Conta conta;
8         std::string historico;
9         float valor;
10        float saldoAnterior;
11        int operacao;
12

```

Figura 4. Arquivo de cabeçalho movimento.h

A relação entre as classes Movimento e Conta é na forma de uma agregação. Os atributos “historico”, “valor” e “operacao”

armazenam, a descrição da transação, valor da transação, se é saque ou depósito; respectivamente.

## C. Classe Transacao

Transacao controla operações de movimentações bancárias. Mesmo sendo totalmente possível um objeto ContaComum, por exemplo, realizar um saque ou depósito, no código do projeto isso é feito apenas por meio de um objeto do tipo Transacao. Pois assim é possível criar um objeto Movimento e registrar as atividades bancárias realizadas.

```

1 #ifndef TRANSACAO_H
2 #define TRANSACAO_H
3 #include <vector>
4 #include "movimento.h"
5
6 class Transacao {
7     private:
8         std::vector<Movimento> movimentos;
9
10    public:
11        ~Transacao();
12        bool realizarTransacao(Conta&, float, std::string, int, float);
13        void estornaTransacao();
14        void extrato(Conta &c1, std::string senha) const;
15 };
16 #endif

```

Figura 5. Arquivo de cabeçalho Transacao.h

A classe Transacao é composta por objetos do tipo Movimento, existe uma relação entre a classe Transacao e a classe Movimento na forma de uma composição, ou seja, no momento em que uma instância de Transacao deixar de existir, serão destruídas todas as instâncias de Movimento, associadas ao objeto. O método “realizarTransacao” cria e adiciona um objeto do tipo Movimento ao vector movimentos, caso a operação seja realizada.

## D. Classe Lista

Template de classe e sobrecarga de operador

```

1 #ifndef LISTA_H
2 #define LISTA_H
3 #include <iostream>
4 #include <memory>
5
6 template <class T>
7 class Lista {
8     private:
9         std::unique_ptr<T[]> contas;
10        size_t tam = 0;
11        size_t size = 0;
12        template <typename>
13        friend std::ostream &operator<<(std::ostream &, Lista<T> &);
14    public:
15        Lista(size_t size = 10) : size_(size) {
16            contas = std::make_unique<T[]>(size);
17        }
18        size_t getTam() const { return tam; }
19        void operator+=(T &c) {
20            contas[tam] = c;
21            tam++;
22        }
23        T &operator[](size_t indice) const { return contas[indice]; }
24 };
25
26 template <class T>
27 std::ostream &operator<<(std::ostream &output, const Lista<T> &lista) {
28     output << "Tamanho da lista: " << lista.getTam() << "\n";
29     std::string nome{};
30     int numero;
31     for (size_t i = 0; i < lista.getTam(); i++) {
32         nome = lista[i].getCorrentista().getNome();
33         numero = lista[i].getNumero();
34         output << "Correntista: " << nome << "    Numero da Conta: " << numero << "\n";
35     }
36     return output;
37 }
38 #endif

```

Figura 6. Arquivo de cabeçalho Lista.h

A classe Lista é composta por elementos do tipo T a ser definido pelo compilador em tempo de execução. Apesar do C++ ser uma linguagem fortemente tipada, ela permite que isso seja feito através dos templates. Além disso outro conceito muito importante presente nesta classe é o de sobrecarga de operador. Os operadores sobrecarregados são +=, [] e <<. Por meio das sobrecargas a classe provém métodos de adicionar um elemento ao array e imprime no terminal informações referentes a esses elementos.

### III. DISCUSSÃO

O arquivo main.cpp do projeto, exemplifica o uso das classe apresentadas anteriormente, da seguinte forma:

- Cria-se 3 objetos do tipo pessoa, passando nome e numero do CPF.
- Em seguida 3 contas, comum, especial e poupança, passando um objeto pessoa, saldo inicial e o número da conta. Conta também espera uma senha, que se não for passada, por padrão é "123".
- Após a criação das contas, é criada uma lista do tipo Conta e tamanho 5, nela são colocadas as 3 contas. Nesse ponto fica clara a função do operador sobrecarregado +=. O operador de saída << também é usado em sobrecarga.
- Por fim um objeto do tipo Transacao é instaciado e 3 movimentações são feitas: José paga o telefone, Maria recebe o salário e Lúcia faz um empréstimo. Depois é emitido um extrato de cada correntista.

```
1 #include <iostream>
2
3 #include "../includes/conta.h"
4 #include "../includes/Lista.hpp"
5 #include "../includes/contaComum.h"
6 #include "../includes/contaEspecial.h"
7 #include "../includes/contaPoupanca.h"
8 #include "../includes/pessoa.h"
9 #include "../includes/transacao.h"
10
11 int main() {
12
13     Pessoa p1("José", "123.456.789-09");
14     Pessoa p2("Maria", "001.456.789-99");
15     Pessoa p3("Lucia", "002.456.789-88");
16
17     ContaComum c1(190521, p1, 750.00f);
18     ContaPoupanca c2(1255534, p2, 1000.00f);
19     ContaEspecial c3(123456, p3, 0.00, 1000);
20
21     //template e sobrecarga de operador
22     Lista<Conta> lista(5);
23
24     lista += c1;
25     lista += c2;
26     lista += c3;
27
28     std::cout << lista;
29
30     /*Polimorfismo-----*/
31     Transacao t1;
32
33     t1.realizarTransacao(c1, 100.00, "Pagamento Telefone", 0, c1.getSaldo());
34     t1.realizarTransacao(c2, 1400.00, "Salário", 1, c2.getSaldo());
35     t1.realizarTransacao(c3, 350.00, "Empréstimo", 0, c3.getSaldo());
36
37     t1.extrato(c1, "123");
38     t1.extrato(c2, "123");
39     t1.extrato(c3, "123");
40
41     return 0;
42 }
```

Figura 7. main.cpp

O programa não recebe nenhuma entrada, apenas imprime demonstrativos das suas funcionalidades. Ele é simples no

seu entendimento e implementação, porém contempla todos os conceitos propostos de orientação a objeto.

Saída do programa após rodar o arquivo executável:

```
rubens@rubens-300ESM-300ESL: ~/Documentos/C++/c++ projeto/banco$ ./main
Tamanho da lista:3
Correntista: José   Numero da Conta: 190521
Correntista: Maria  Numero da Conta: 1255534
Correntista: Lucia  Numero da Conta: 123456

-----
Emissão de Extrato:
Conta Comum :
Numero da conta.....: 190521
Correntista.....: José
Saldo.....: 650
CPF.....: 123.456.789-09

Movimentações:
Descrição.....:Pagamento Telefone
Valor.....:100
Operação.....:Saque

Saldo Anterior: 750
Saldo Atual: 650

-----
Emissão de Extrato:
Conta Poupança
Numero.....: 1255534
Correntista.....: Maria
Saldo.....: 2400
CPF.....: 001.456.789-99

Movimentações:
Descrição.....:Salário
Valor.....:1400
Operação.....:Deposito

Saldo Anterior: 1000
Saldo Atual: 2400

-----
Emissão de Extrato:
Conta Especial - Numero.....: 123456
Correntista.....: Lucia
Saldo.....: -350
Limite Conta Especial.....: 1000
Limite Atual.....: 650
CPF.....: 002.456.789-88

Movimentações:
Descrição.....:Empréstimo
Valor.....:350
Operação.....:Saque

Saldo Anterior: 0
Saldo Atual: -350
rubens@rubens-300ESM-300ESL: ~/Documentos/C++/c++ projeto/banco$
```

Figura 8. Saída do Programa

#### A. Ambiente de Desenvolvimento

O código foi desenvolvido e testado na IDE VScode em máquina com o sistema operacional Ubuntu 20.04 LTS, compilador GCC (GNU Compiler Collection) na versão 9.3.0.

#### B. Uso de Memória, Flags para Compilação e Makefile

O uso de memória foi verificado usando valgrind. Valgrind é um software livre que auxilia o trabalho de depuração de programas. Ele possui ferramentas que detectam erros decorrentes do uso incorreto da memória dinâmica, como por exemplo os vazamentos de memória, alocação e desalocação incorretas e acessos a áreas inválidas

De maneira bem simplista um makefile é um arquivo de texto que contém instruções sobre como compilar e executar (ou construir) um conjunto de arquivos de código-fonte C++.

Flags usadas na compilação do projeto: -std=c++17 -Wall -Wextra -Werror -Wshadow -Wconversion -Wcast-align -Wcast-qual -Wctor-dtor-privacy -Wdisabled-optimization -Wformat=2 -Winit-self -Wlogical-op -Wmissing-declarations -Wmissing-include-dirs -Wnoexcept -Wold-style-cast -Woverloaded-virtual -Wredundant-decls -Wsign-conversion

atendendo às boas práticas de programação. Referente aos conceitos ensinados durante o semestre, o projeto foi essencial para o entendimento dos mesmos.

Uso das flags mencionadas na compilação:



Figura 9. Flags

## Rodando o arquivo executável com o Valgrind

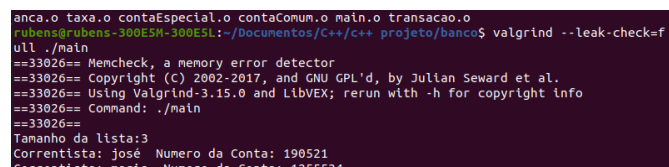


Figura 10. Executando com o valgrind

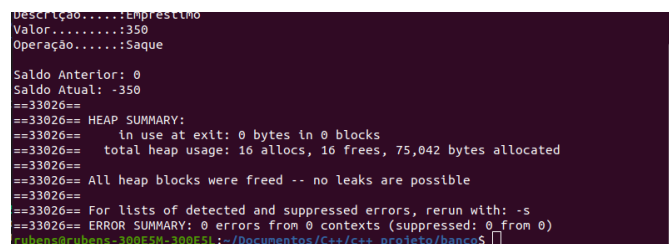


Figura 11. Resultado da Analise

#### IV. CONCLUSÃO

O programa desenvolvido mostrou que atende todos os requisitos propostos, além de apresentar boa performance e

## V. REFERÊNCIAS

- [1] - Documentação Oficial C++ <http://www.cplusplus.com/>
- [2] - Documentação Microsoft da linguagem C++ <https://docs.microsoft.com/pt-br/cpp/?view=msvc-160>
- [3] - Wikilivros - [https://pt.wikibooks.org/wiki/Programar\\_em\\_C%2B%2B](https://pt.wikibooks.org/wiki/Programar_em_C%2B%2B)