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

***Abstract*—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 o funcionamento do algoritmos desenvolvido para simular operações do sistema bancário feito como trabalho final para a disciplina de programação 3.

II. Desenvolvimento

A. Classe Conta

```
#ifndef CONTA_H
#define CONTA_H
#include <iostream>
#include <string>
#include "pessoa.h"

class Conta {
   private:
    static int contasCriadas;
    std::string senha = "";
   protected:
    int numero;
    Pessoa *correntista;
    float saldo{};
   public:
    Conta();
    Conta(int numero, Pessoa &correntista, float saldo, std::string senha);
    virtual ~Conta();

    bool validacao(std::string chave) const;
    int getNumero();
    void setNumero(int num);
    Pessoa getCorrentista();
    void setCorrentista(Pessoa &correntista);
    float getSaldo();
    void setSaldo(float saldo);
    int getNumeroTotalDeContas();
    bool movimentar(float valor, int operacao);
    void depositar(float valor);

    virtual void info() const;
    virtual bool sacar(float valor);
};
#endif
```

Fig. 1. Arquivo de cabeçalho conta.h

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 existir quando 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.

```
#ifndef PESSOA_H
#define PESSOA_H
#include <iostream>
#include <string>

class Pessoa {
   private:
    std::string nome;
    std::string CPF;
   public:
    Pessoa();
    Pessoa(std::string _nome, std::string _CPF);
    ~Pessoa();
    void setNome(std::string _nome);
    void setCPF(std::string _CPF);
    std::string getNome();
    std::string getCPF();
};
#endif
```

Fig. 2. Arquivo de cabeçalho pessoa.h

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.

```
#ifndef CONTAPOUPANCA_H
#define CONTAPOUPANCA_H
#include "conta.h"
#include "taxa.h"

class ContaPoupanca : public Conta, public Taxa {
   private:
   //Não há atributos em ContaPoupanca além dos definidos nas classes Conta e Taxa
   public:
    ContaPoupanca(int numero, Pessoa &correntista, float saldo);
    ~ContaPoupanca();
    virtual void info() const override;
    virtual void incremento_juros() override;
    virtual void descontarTaxaManutencao() override{};
};

#endif
```

Fig. 3. Arquivo de cabeçalho contaPoupanca.h

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 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.

```
#ifndef MOVIMENTO_H
#define MOVIMENTO_H
#include "conta.h"

class Movimento {
   private:
    Conta conta;
    std::string historico;
    float valor;
    float saldoAnterior;
    int operacao;

```

Fig. 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.

```
#ifndef TRANSACAO_H
#define TRANSACAO_H
#include <vector>
#include "movimento.h"

class Transacao {
   private:
    std::vector<Movimento> movimentos;

   public:
    ~Transacao();
    bool realizarTransacao(Conta&, float, std::string, int, float);
    void estornaTransacao();
    void extrato(Conta &c1, std::string senha) const;
};
#endif
```

Fig. 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

```
#ifndef LISTA_H
#define LISTA_H
#include <iostream>
#include <memory>

template <class T>
class Lista {
 private:
  std::unique_ptr<T[]> contas;
  size_t tam = 0;
  size_t size = 0;
  template <typename>
  friend std::ostream &operator<<(std::ostream &, Lista<T> &);
 public:
  Lista(size_t _size = 10) : size(_size) {
    contas = std::make_unique<T[]>(size);
  }
  size_t getTam() const { return tam;}
  void operator+=(T &c) {
    contas[tam] = c;
    tam++;
  }
  T &operator[](size_t indice) const { return contas[indice];}
};

template <class T>
std::ostream &operator<<(std::ostream &output, const Lista<T> &lista) {
  output << "Tamanho da lista:" << lista.getTam() << "\n";
  std::string nome{};
  int numero;
  for (size_t i = 0; i < lista.getTam(); i++) {
    nome = lista[i].getCorrentista().getNome();
    numero = lista[i].getNumero();
    output << "Correntista: " << nome << "  Numero da Conta: " << numero << "\n";
  }
  return output;
}
#endif
```

Fig. 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 fortimente 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 anteriormeente, da segunite 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 saida << 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.

```cpp
#include <iostream>

#include "../includes/conta.h"
#include "../includes/Lista.hpp"
#include "../includes/contaComum.h"
#include "../includes/contaEspecial.h"
#include "../includes/contaPoupanca.h"
#include "../includes/pessoa.h"
#include "../includes/transacao.h"

int main() {

  Pessoa p1("josé", "123.456.789-09");
  Pessoa p2("maria", "001.456.789-99");
  Pessoa p3("lucia", "002.456.789-88");

  ContaComum c1(190521, p1, 750.00f);
  ContaPoupanca c2(1255534, p2, 1000.00f);
  ContaEspecial c3(123456, p3, 0.00, 1000);

  //template e sobrecarga de operador
  Lista<Conta> lista(5);

  lista += c1;
  lista += c2;
  lista += c3;

  std::cout << lista;

  /*Polimorfismo--------------------*/
  Transacao t1;

  t1.realizarTransacao(c1, 100.00, "Pagamento Telefone", 0, c1.getSaldo());
  t1.realizarTransacao(c2, 1400.00, "Salário", 1, c2.getSaldo());
  t1.realizarTransacao(c3, 350.00, "Empréstimo", 0, c3.getSaldo());

  t1.extrato(c1, "123");
  t1.extrato(c2, "123");
  t1.extrato(c3, "123");

  return 0;
}
```

Fig. 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-300E5M-300E5L: ~/Documentos/C++/c++ projeto/banco
rubens@rubens-300E5M-300E5L:~/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 :
Número 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
Número....................: 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 - Número...........: 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-300E5M-300E5L:~/Documentos/C++/c++ projeto/banco$
```

Fig. 8. Saída do Programa

A. Ambiente de Desenvolmento

O código foi desenvolvido e testado na IDE VScode em máquina com o sistema operacinal 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 maniera 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 -Wsign-promo -Wstrict-null-sentinel -Wstrict-overflow=5 -Wswitch-default -Wundef -Wno-unused

C. Demonstração de Funcionamento

Uso das flags mencionadas na compilação:

Fig. 9. Flags

Rodando o arquivo executável com o Valgrind

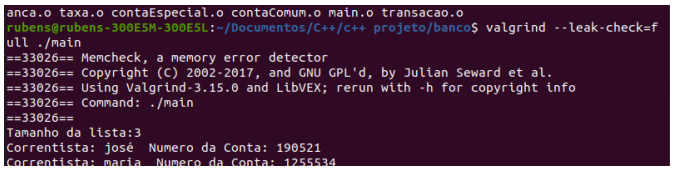

Fig. 10. Executando com o valgrind

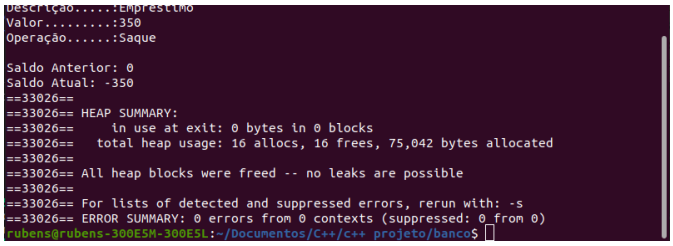

Fig. 11. Resultado da Analise

IV. Conclusão

O programa desenvolvido mostrou que atende todos os requisitos propostos, além de apresentar boa performace e atendendo às boas práticas de programação. Referente aos conceitos ensinados durante o semestre, o projeto foi essencial para o entendimento dos mesmos.

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