PROJETO INTERDISCIPLINAR 2º SEMESTRE
APS – Atividade Prática Supervisionada
DISCIPLINAS: Programação Orientada a Objetos e Técnicas de Programação

1 – Descrição do Projeto

Desenvolver um sistema para gerenciamento de uma instituição financeira (Banco).

O sistema bancário deve permitir o cadastro de clientes (Nome, cpf e telefone) e de contas bancárias comuns (Conta) e contas bancárias especiais (ContaEspecial). Para as contas comuns, é necessário armazenar o número da conta, o cliente, a data de abertura e o saldo. Para as contas especiais, além dos dados da conta comum, armazenar também o valor de limite. O limite é até quanto o cliente pode ficar devendo ao banco quando não tiver mais saldo.

As funcionalidades básicas que o sistema deve ter são:

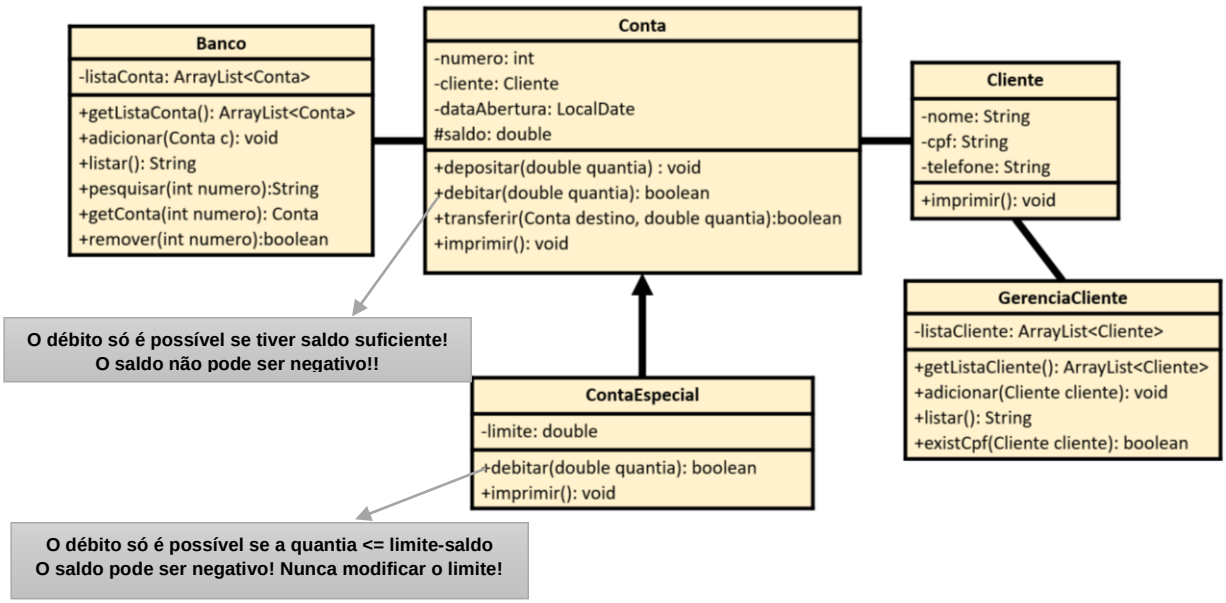
- Cadastro de clientes;
- Listar clientes cadastrados;
- Cadastro de contas / contas especiais – neste caso lembre se estabelecer um valor para o limite;
- Realizar o depósito de uma quantia em uma conta específica;
- Realizar o débito (saque) de uma quantia em uma conta específica. Verificar se há saldo suficiente para realizar a operação de débito. Caso seja uma conta especial, **nunca modificar o valor do limite**. Modificar apenas o saldo podendo ficar negativo;
- Listar contas cadastradas. Listar primeiro as contas comuns seguidas das especiais;
- Pesquisar pelo número de uma conta. Retornar todos os dados da conta encontrada;
- Transferir uma quantidade de uma conta origem para uma conta destino. Realizar a transferência apenas se houver saldo suficiente na conta origem.
- Remover uma conta pelo seu número. Para excluir uma conta ela deve estar com o saldo zerado. Caso contrário, não é possível realizar a exclusão. O método deve retornar um boolean/inteiro para indicar se foi possível realizar a exclusão ou não;

2 – Detalhamento para a disciplina Programação Orientada a Objetos

Implementar a hierarquia e associação de classes do diagrama a seguir. O modificador **protected** é representado por # no diagrama.

O saldo da classe **Conta** nunca pode ficar negativo. Ao usar o limite da **ContaEspecial**, o saldo dessa conta especial deve ficar negativo. Após estabelecer um valor para o limite, ele **não deve** ser modificado. Usar o limite de R$ 1.000,00 para todas as contas especiais.

O número da **Conta** deve ser incremental. Pesquisar o uso do modificador **static** para definir um número incremental.



As operações de **depositar()**, **debitar()** e **transferir()** devem ser implementadas nas classes Conta/ContaEspecial.

- **depositar()** – recebe uma quantia e realiza o depósito atualizando o saldo;
- **debitar()** – recebe uma quantia, realiza o débito (caso seja possível) e retorna um **boolean** para indicar se o débito foi realizado com sucesso ou não;
- **transferir()** – recebe uma conta destino (**Conta**) e a quantia, realiza a transferência (caso seja possível) e retorna um **boolean** para indicar se a transferência foi realizada com sucesso ou não;

Implementar a classe **GerenciaCliente** para gerenciar um ArrayList tipado em **Cliente**. Implementar os seguintes métodos para esta classe:

- **adicionar()** – cadastrar objetos **Cliente** na lista;
- **listar()** – retorna uma String com os dados de todos os clientes da lista;
- **existCpf** – recebe um objeto Cliente e verifica se já existe um cliente com o cpf do objeto parâmetro cadastrado na lista. Retorna **true** se já existir e **false** se não existir.

Implementar a classe **Banco** com um atributo que é um ArrayList tipado em **Conta** podendo armazenar contas comuns e especiais.

Implementar os seguintes métodos para a classe **Banco**:

- **adicionar()** – cadastrar objetos **Conta** na lista;
- **listar()** – retorna uma String com os dados de todas as contas comuns primeiro, seguidos dos dados de todas as contas especiais;
- **pesquisar()** – recebe o número de uma conta e retorna uma String com todos os dados da conta;
- **getConta()** – recebe o número de uma conta e retorna um objeto Conta da lista de contas. Caso não encontre, retorna null;
- **remover()** – recebe o número da conta que deseja remover. Para excluir uma conta ela deve estar com o saldo zerado. Caso contrário, não é possível realizar a exclusão. O método deve retornar um boolean para indicar se foi possível realizar a exclusão ou não;

Todas as interações de entrada e saída do sistemas devem utilizar a classe **JOptionPane** e os métodos **showInputDialog()** e **showMessageDialog()**. Todas as mensagens de entrada e saída de dados devem estão na classe da aplicação.

Desenvolver uma classe de Aplicação que tenha um menu como segue:

=== Menu do Banco ===
1 – Cadastrar Cliente
2 – Cadastrar Conta
3 – Listar Conta
4 – Pesquisar Conta
5 – Depositar
6 – Debitar (Sacar)
7 – Transferir
8 – Remover Conta
9 – Sair
Escolha uma opção:

Ao escolher o item 2 do menu, deve-se perguntar ao usuário qual o tipo da conta ele deseja cadastrar.

3 – Detalhamento para a disciplina Técnicas de Programação

Implementar os cabeçalhos apresentados abaixo.

banco.h:

```
#include "conta.h"
TConta contas[10];
int numeroConta=0;

void adicionarConta(TConta conta);
char* listarContas();
char* pesquisarConta(int numero);
TConta getConta(int numero);
int removerConta(int numero);
```

conta.h:

```
#include "cliente.h"
typedef struct Data{
int dia;
int mes;
int ano;
}TData;

typedef struct Conta{
int numero;
TCliente cliente;
TData dataAbertura;
float saldo;
int ativa;
}TConta;

void depositar(TConta conta,float valor);
int debitar(TConta conta, float quantia);
int transferir(TConta origem, TConta destino,
                                    float quantia);
void imprimirConta(TConta conta);
```

gerenciaClientes.h:

```
#include "cliente.h"

TCliente clientes[10];
int totalClientes =0;

void adicionarCliente(TCliente cliente);
int existeCPF(TCliente cliente);
char* listarClientes();
```

cliente.h:

```
#ifndef CLIENTE_H
#define CLIENTE_H
typedef struct Cliente{
char nome[10];
char cpf[11];
char telefone[10];
}TCliente;

void imprimirCliente(TCliente cliente);
#endif
```

O saldo do cabeçalho conta.h, representado pelo registro TConta pode ser negativo. Ao usar o limite do registro TConta, o saldo dessa conta deve ficar negativo. Usar o limite de R$ 1.000,00 para todas as contas.

O número da conta deve ser incremental. Pesquisar o uso de variáveis globais para definir um número incremental. A conta possui a variável **int** ativa**,** para indicar se a conta está ativa (valor 1) ou inativa (valor 0).

As operações de **depositar()**, **debitar()** e **transferir()** devem ser implementadas no cabeçalho Conta.

- **depositar()** – recebe uma quantia e realiza o depósito atualizando o saldo;
- **debitar()** – recebe uma quantia, realiza o débito (caso seja possível) e retorna um **int** para indicar se o débito foi realizado com sucesso (valor 1) ou não (valor 0);
- **transferir()** – recebe uma conta origem, uma conta destino (**TConta**) e a quantia, realiza a transferência (caso seja possível) e retorna um **int** para

indicar se a transferência foi realizada com sucesso (valor 1), ou não (valor 0);

Implementar o cabeçalho **gerenciaClientes.h** para gerenciar um vetor do tipo **TCliente**. Implementar as seguintes funções para este cabeçalho:

- **adicionarCliente()** – inserir registros **TCliente** no vetor;
- **listarClientes()** – retorna um vetor de char com os dados de todos os clientes da lista;
- **existeCpf** – recebe um registro TCliente e verifica se já existe um cliente com o cpf do registro parâmetro cadastrado no vetor. Retorna **1** se já existir e **0** se não existir.

Implementar o cabeçalho **banco.h** com um vetor do tipo **TConta**.

Implementar os seguintes métodos para o cabeçalho **banco.h**:

- **adicionarConta()** – cadastrar registros do tipo **TConta** no vetor;
- **listarContas()** – retorna um vetor de char com os dados de todas as contas;
- **pesquisarConta()** – recebe o número de uma conta e retorna um vetor de char com todos os dados da conta;
- **getConta()** – recebe o número de uma conta e retorna um registro TConta do vetor de contas. Caso não encontre, não retorna nada;
- **remover()** – recebe o número da conta que deseja remover. Para excluir uma conta ela deve estar com o saldo zerado. Caso contrário, não é possível realizar a exclusão. Uma conta é removida caso o valor da variável int ativa seja 0. O método deve retornar um int para indicar se foi possível realizar a exclusão ou não;

Desenvolver uma classe de Aplicação que tenha um menu como segue:

=== Menu do Banco ===
1 – Cadastrar Cliente
2 – Cadastrar Conta
3 – Listar Conta
4 – Pesquisar Conta
5 – Depositar
6 – Debitar (Sacar)
7 – Transferir
8 – Remover Conta
9 – Sair
Escolha uma opção:

4 – Entrega

A entrega deve ser realizada pelo Blackboard. Cada grupo deve entregar o projeto (Netbeans / CodeBlocks) compactado em um arquivo **ZIP** (não usar extensão rar ou 7zip).