Gabriel Nunes Crispino – Erros:

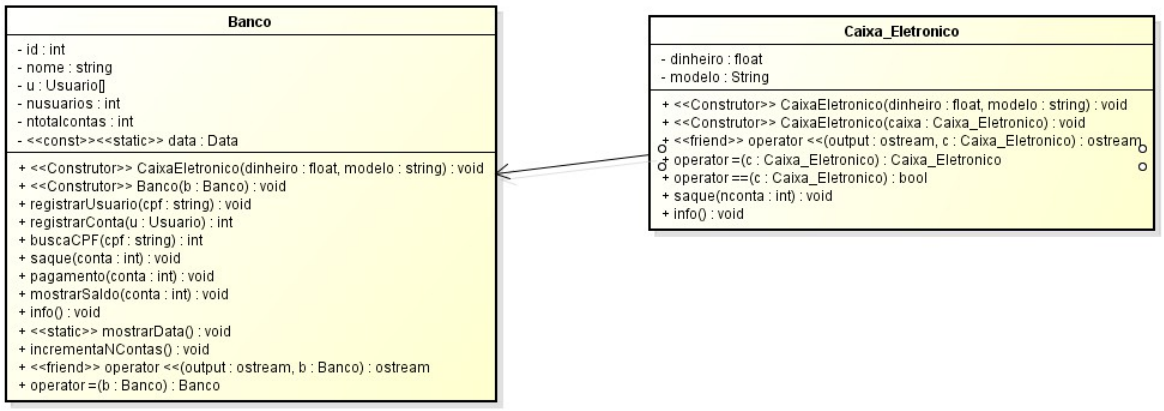
“Evitar retornar vetores. Melhor seria retornar uma conta em específico.

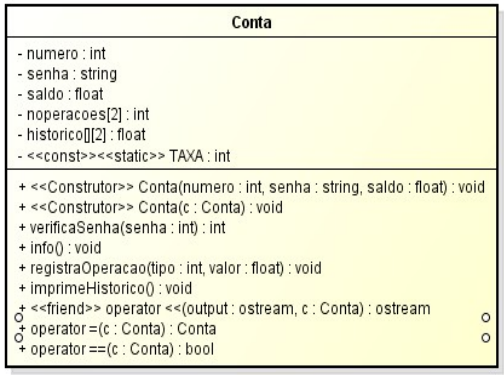
```
const vector<int> & getConta()”
```

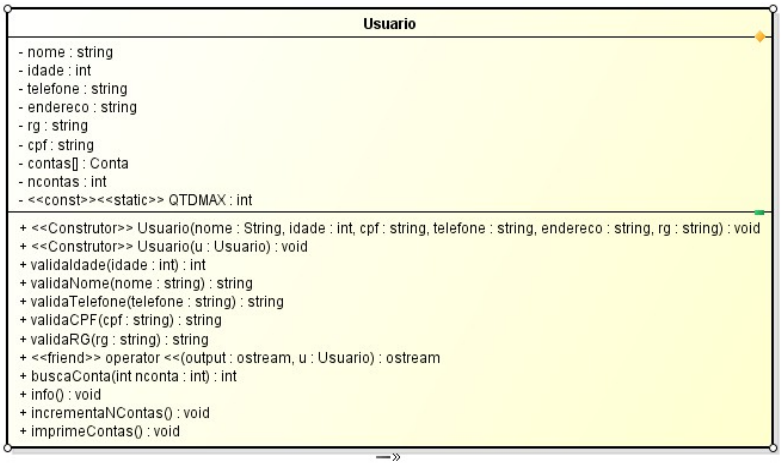
Função “vector <int> & getConta()” não existe mais, pois o vector<int> de contas foi substituído por um ponteiro que aponta para um array de objetos do tipo Conta, classe criada posteriormente. No entanto, existe uma função *“Conta * getContas()”*, onde eu acho que o mesmo conceito se aplica, de não ser adequado retornar um ponteiro para contas, ao invés de uma conta em específico. Ou nesse caso é adequado?

Requisitos herança:

- Diagrama de classes (obrigatório salvar também o png do diagrama no gitHub)
  Requisito cumprido, os arquivos .asta e .png foram salvos no repositório, em uma pasta “UML”. Seguem abaixo as imagens dos diagramas:







- Herança pública
  Requisito cumprido, a classe “Caixa_Eletronico” sofre herança pública da classe “Banco”.

```
20  class Caixa_Eletronico: public Banco
21  {
22  friend ostream &operator << (ostream &,const Caixa_Eletronico &); //sobrecarga do operador "<<".
23
24  private:
25      float dinheiro; //Dinheiro disponível no caixa eletrônico.
26      string modelo; //Modelo do caixa eletrônico.
27
28
29  public:
30      Caixa_Eletronico(float dinheiro = 0, string modelo = ""); //Construtor default que inicializa as variaveis dinheiro e modelo
31      Caixa_Eletronico(Caixa_Eletronico &); //Construtor de cópia
32      ~Caixa_Eletronico();
33
34      bool operator == (const Caixa_Eletronico &c1); //Sobrecarga do operador "==".
35      Caixa_Eletronico operator = (const Caixa_Eletronico &c); //Sobrecarga do operador "=".
36
37
38      void saque(int); //realiza uma operação de saque no caixa eletrônico.
39      void info() const; //imprime as informações do caixa na tela.
40  };
```

- Construtor de cópia, e sobrecargas dos operadores de atribuição (=) e << (cout << base) para a classe base e derivada :
  Como pode ser visto na imagem acima, há um construtor de cópia na classe derivada. Na imagem abaixo podemos ver um construtor de cópia na classe base (“Banco”).

```
9   class Banco
10  {
11      friend ostream &operator << (ostream &,const Banco &); //sobrecarga do operador "<<".
12
13  protected:
14      int id;
15      string nome;
16
17      Usuario *u;
18      int nusuarios;
19      int ntotalcontas;
20
21      const static Data d;
22  public:
23      Banco(string nome = "",int id = 123456);
24      Banco(Banco &b);
25      virtual ~Banco();
26
27      Banco operator = (const Banco &); //sobrecarga do operador "=".
28
29      //Funções get e set
30      int getNContas();
```

- Usar Protected acessando diretamente os atributos na classe derivada:
  Como pode ser visto na imagem acima, foi usado *protected* na classe banco, para que a sua classe derivada acessasse seus atributos diretamente.

- Conceitualmente errado, em sala me pergunta por que: `if (Caixa_Eletronico::nusuarios <= 1)`
Isso faltou o sr me explicar.

– Alocação dinâmica de memória na classe base e derivada:
  Requisito cumprido parcialmente, a alocação dinâmica foi feita apenas na classe base, como pode ser visto na última imagem acima (“*Usuario* **u*”).

– Sobrescrita de método: chamar dentro do método da classe derivada o método correspondente da classe base usando ::
  Requisito também parcialmente completo. Como pode ser visto nas imagens abaixo, temos dois métodos com a mesma assinatura na classe base e na derivada(“void *info() const”).* Só o que falta é a utilização desses métodos como requerido.
  PS: também pode ser feito com o método “*void saque(int)*”, o qual está presente em ambas as classes.

```
void saque(int); //realiza uma operação de saque no caixa eletrônico.
void info() const; //imprime as informações do caixa na tela.
```

```
38      void info() const;
39
40      void registrarUsuario(string &); //Registra um usuário novo no Caixa.
41      int registrarConta(Usuario &); //Registra uma conta nova no usuário passado como parâmetro.
42      int buscaCPF(string); //faz uma busca de um usuario através do seu CPF.
43      void saque(int); //faz um saque pelo caixa do banco.
44      void pagamento(int); //realiza uma operação de pagamento de uma conta para outra.
45      void mostrarSaldo(int) const; //imprime o saldo disponível na conta na tela.
46      void incrementaNContas();
47      static void mostrarData();//imprime a data na tela.
48  };
```

– No main: criar um ponteiro da classe base para alocar memória para a classe derivada e chamar os vários métodos implementados
  Esse requisito realmente não foi cumprido.