INF101 – Introdução à Programação II

Roteiro de Prática: 16 de novembro de 2023

Introdução

Nesta aula, vamos modelar um banco de varejo. Em aulas teóricas, modelamos as classes componentes de um banco: `Cliente` e `Conta`. Hoje vamos modelar o banco em si e vamos também acrescentar uma classe especializada para conta corrente que permite saque a descoberto.

Dizemos, em metodologia de orientação a objetos, que um banco **tem** clientes e contas. Logo a classe `Banco` fará uso dessas duas classes e o banco armazena todas as contas de seus clientes. Os atributos de um banco serão o nome (do banco) e a lista de suas contas. Como métodos, teremos a operação de abertura de conta corrente e a listagem (impressão) de todas as contas do banco.

Para a classe da conta especializada, dizemos, em orientação a objetos, que ela **herda** as características e o comportamento de sua superclasse `Conta`. A classe especializada é uma subclasse de `Conta` e a chamaremos de `ContaEspecial`.

Siga as instruções abaixo para elaborar o programa.

Instruções

1. Abra o IDLE e crie um novo arquivo fonte denominado `p10.py`. Não se esqueça de salvá-lo de tempos em tempos, porque pode ocorrer falha de energia elétrica durante a aula prática.
2. Digite os comentários obrigatórios (nome, matrícula, data e uma breve descrição sobre o que o programa faz).
3. Organize seu programa com as classes `Cliente`, `Conta`, `Banco`, `ContaEspecial` e a função `main()`.
4. As classes `Cliente`, `Conta` e `Banco` já estão prontas. Foram feitas em aulas teóricas. Baixe, para seu diretório de trabalho, a implementação delas a partir do *site* de entrega. Elas estão implementadas no arquivo `banco.py`.
5. A classe `Banco`, que já foi elaborada em aula teórica, define o atributo `nome` que será inicializado a partir de um parâmetro passado pelo construtor `__init__`. Além disso, deverá ter também o atributo `contas` que deverá ser inicializado com a lista vazia. Os métodos de `Banco` deverão ser: `abreConta` que passará um objeto `conta` como parâmetro; e `listaContas` que não terá nenhum parâmetro a não ser o objeto `self` para indicar que é um método da classe `Banco`. A operação de abertura de conta simplesmente acrescenta a conta transmitida como parâmetro à lista de contas do `Banco`. A operação de listagem das contas imprime um cabeçalho com o nome do banco e, a seguir, o resumo de todas as contas do `Banco`. Lembre-se de que o método `resumo` já está definido e pronto na classe `Conta`.
6. A subclasse `ContaEspecial` herda da classe `Conta`. Ela deve redefinir o construtor `__init__` com os parâmetros `clientes`, `numero`, `saldo` (inicializado com `0.00` por *default*) e `limite` (também inicializado com `0.00` por *default*). Então ela deve chamar o `__init__` da classe base e inicializar o atributo `self.limite` com o valor transmitido pelo parâmetro `limite`. Além disso, a subclasse `ContaEspecial` deve redefinir o método `saque` de modo a permitir a realização de saque a descoberto até o limite dado de saldo negativo. Além disso, vamos acrescentar uma mensagem de advertência caso haja tentativa de saque além do saldo mais o limite a descoberto da conta. Veja como deve ser essa mensagem no Exemplo de Execução do Programa abaixo.

7. A função `main` deve testar as classes da biblioteca `banco.py` fornecida e a subclasse `ContaEspecial`. Use os dados que são mostrados a seguir:

```
from banco import Cliente, Conta, Banco

...

cliente1 = Cliente("João Silva", "3456-7890")
cliente2 = Cliente("Maria Silva", "3456-7890")
cliente3 = Cliente("José Vargas", "2351-1809")
cliente4 = Cliente("Marina Lima", "(21) 3509-4390")
contaJM = ContaEspecial([cliente1, cliente2],
                        "76534", 100.00, 500.00)
contaJ  = Conta([cliente3], "80297", 10.00)
contaM  = Conta([cliente4], "81020")

banco = Banco("Tatu")
banco.abreConta(contaJM)
banco.abreConta(contaJ)
banco.abreConta(contaM)
banco.listaContas()

contaJM.saque(50.00)
contaJ.deposito(300.00)
contaJM.saque(190.00)
contaJ.deposito(95.26)
contaJ.saque(245.00)

contaJM.saque(654.38)  # não será realizado; ultrapassa limite
contaM.saque(102.85)   # idem; conta sem saldo suficiente

contaJM.extrato()
contaJ.extrato()

banco.listaContas()
```

8. Não se esqueça de chamar a função `main` no final de seu código fonte para iniciar todo o processo.
9. Veja como deve ser a saída do programa no exemplo dado na página seguinte.

Após certificar-se de que seu programa esteja correto, envie o arquivo do programa fonte (`p10.py`) através do sistema de entrega do LBI.

Exercício Extra (Não é necessário entregar)

Na classe `Banco`, modique a estrutura de dados de armazenamento das contas para dicionário em vez de lista como proposto acima no passo 5. As chaves do dicionário devem ser os números das contas. Observe que os respectivos comandos nos métodos `__init__`, `abreConta` e `listaContas` têm de ser adequadamente modificados para lidar com dicionário. Para a listagem das contas em `listaContas`, imprima-as em ordem crescente do número da conta.

Exemplo de Execução do Programa

```
Contas do Banco Tatu

CC nº 76534 Saldo:     100.00
CC nº 80297 Saldo:      10.00
CC nº 81020 Saldo:       0.00

***CC nº 76534: saque de 654.38 não realizado; saldo disponível de 360.00

***CC nº 81020: saque de 102.85 não realizado; saldo disponível de 0.00



Extrato CC nº 76534

  DEPÓSITO     100.00
     SAQUE      50.00
     SAQUE     190.00

     SALDO    -140.00


Extrato CC nº 80297

  DEPÓSITO      10.00
  DEPÓSITO     300.00
  DEPÓSITO      95.26
     SAQUE     245.00

     SALDO     160.26


Contas do Banco Tatu

CC nº 76534 Saldo:    -140.00
CC nº 80297 Saldo:     160.26
CC nº 81020 Saldo:       0.00
```