*# Linguagem de Programação II*

*# AC03 ADS-EaD - Banco*

*#*

*# Email: alexandro.carvalho@aluno.faculdadeimpacta.com.br*

*from* typing *import* Union, List, Dict

Number = Union[int, float]

class Cliente():

*"""*

*Classe Cliente do Banco.*

*possui os atributos PRIVADOS:*

*- nome,*

*- telefone,*

*- email.*

*caso o email não seja válido (verificar se contém o @) gera um ValueError,*

*caso o telefone não seja um número inteiro gera um TypeError*

*"""*

def \_\_init\_\_(self, nome: str, telefone: int, email: str):

self.\_\_nome = nome

self.\_\_telefone = telefone

self.\_\_email = email

def get\_nome(self) -> str:

*"""Acessor do atributo Nome."""*

*return* self.\_\_nome

def get\_telefone(self) -> int:

*"""Acessor do atributo Telefone."""*

*return* self.\_\_telefone

def set\_telefone(self, novo\_telefone: int) -> None:

*"""*

*Mutador do atributo telefone, caso não receba um número,*

*gera um TypeError*

*"""*

*if* (type(novo\_telefone) == int):

self.\_\_telefone = novo\_telefone

*return* self.\_\_telefone

*else*:

*raise* TypeError('Número Inválido')

def get\_email(self) -> str:

*"""Acessor do atributo Email."""*

*return* self.\_\_email

def set\_email(self, novo\_email: str) -> None:

*"""*

*Mutador do atributo Email, caso não receba um email válido*

*(contendo o @), gera um ValueError.*

*"""*

*if* '@' in novo\_email:

self.\_\_email = novo\_email

*return* self.\_\_email

*else*:

*raise* ValueError('Email inválido')

class Banco():

*"""*

*A classe Banco deverá receber um nome em sua construção e deverá*

*implementar os métodos:*

*- abre\_conta(): abre uma nova conta, atribuindo o numero da conta em ordem*

*crescente a partir de 1 para a primeira conta aberta.*

*- lista\_contas(): apresenta um resumo de todas as contas do banco*

*DICA: crie uma variável interna que armazene todas as contas do banco*

*DICA2: utilze a variável acima para gerar automaticamente o número das*

*contas do banco*

*"""*

def \_\_init\_\_(self, nome: str):

self.\_\_nome = 'nome'

self.\_\_contas = []

def get\_nome(self) -> str:

*"""Acessor do Atributo Nome."""*

*return* self.\_\_nome

def abre\_conta(self, clientes: List[Cliente], saldo\_ini: Number) -> None:

*"""*

*Método para abertura de nova conta, recebe os clientes*

*e o saldo inicial.*

*Caso o saldo inicial seja menor que 0 devolve um ValueError*

*"""*

*#nova\_conta = [Conta([clientes],1,saldo\_ini)]*

*#while saldo\_ini < 0:*

*#raise ValueError('Valor negativo')*

n\_conta = len(self.\_\_contas) + 1

*while* saldo\_ini < 0:

*raise* ValueError('Valor negativo')

c1 = Conta('Banco1',n\_conta,0)

self.\_\_contas.append(c1)

def lista\_contas(self) -> 'conta':

*"""Retorna a lista com todas as contas do banco."""*

*#conta = Conta([Cliente],1,saldo\_inicial = 100)*

*#conta2 = Conta([Cliente],2,saldo\_inicial = 500)*

*#lista = [conta]*

*return* self.\_\_contas

class Conta():

*"""*

*Classe Conta:*

*- Todas as operações (saque e deposito, e saldo inicial) devem aparecer*

*no extrato como tuplas, exemplo ('saque', 100), ('deposito', 200) etc.*

*- Caso o saldo inicial seja menor que zero deve lançar um ValueError*

*- A criação da conta deve aparecer no extrato com o valor*

*do saldo\_inicial, exemplo: ('saldo\_inicial', 1000)*

*DICA: Crie uma variável interna privada para guardar as*

*operações feitas na conta*

*"""*

def \_\_init\_\_(self, clientes: List[Cliente],

numero\_conta: int,

saldo\_inicial: Number):

self.\_\_clientes = clientes

self.\_\_numero\_conta = numero\_conta

self.\_\_saldo\_inicial = saldo\_inicial

saldo = self.\_\_saldo\_inicial

self.\_\_extrato = [('saldo\_inicial',saldo\_inicial)]

def get\_clientes(self) -> List[Cliente]:

*'''*

*Acessor para o atributo clientes*

*'''*

*return* self.\_\_clientes

def get\_saldo(self) -> Number:

*'''*

*Acessor para o atributo saldo*

*'''*

*return* self.\_\_saldo\_inicial

def get\_numero(self) -> int:

*'''*

*Acessor para o atributo numero*

*'''*

*return* self.\_\_numero\_conta

def saque(self, valor: Number) -> None:

*'''*

*Método de saque da classe Conta, operação deve aparecer no extrato*

*Caso o valor do saque seja maior que o saldo da conta,*

*deve retornar um ValueError, e não efetuar o saque*

*'''*

*try*:

*if* valor < self.\_\_saldo\_inicial:

self.\_\_saldo\_inicial = self.\_\_saldo\_inicial - valor

extrato = (('saque',valor))

self.\_\_extrato.append(extrato)

*except* ValueError:

ValueError('Valor maior que o saldo')

def deposito(self, valor: Number):

*'''*

*Método depósito da classe Conta, operação deve aparecer no extrato*

*'''*

self.\_\_saldo\_inicial = self.\_\_saldo\_inicial + valor

extrato = (('deposito',valor))

self.\_\_extrato.append(extrato)

def extrato(self) -> List[Dict[str, Number]]:

*'''*

*Retorna uma lista com as operações (Tuplas) executadas na Conta*

*'''*

*return* self.\_\_extrato