```
# Programação Orientada a Objetos
# AC02 ADS-EaD - Implementação de classes
#
# Email Impacta: _____________@aluno.faculdadeimpacta.com.br

"""
Nesta atividade você deve implementar uma classe que modela uma conta
em um banco.
Um "esqueleto" da classe encontra-se montado mais abaixo, e você deve
utilizá-lo
para completar o que falta.
Siga as instruções abaixo detalhadamente.

Parte 1) definir os seguintes atributos privados no construtor da
classe:
        - titular: inicializado através do parâmetro de mesmo nome;
        - agencia: inicializado através do parâmetro de mesmo nome;
        - numero: inicializado através do parâmetro de mesmo nome;
        - saldo: inicializado através do parâmetro saldo_inicial;
        - ativa: inicializado com o valor booleano False;
        - operacoes: lista inicializada com o valor inicial: [('saldo
inicial', saldo_inicial)]

        * OBS: note que o(s) valor(es) contido(s) na lista de
operações (que é um
        atributo privado) sempre serão tuplas, cada uma com 2 valores.


Parte 2) definir as seguintes propriedades (properties):
        titular: retorna o valor do atributo privado titular;
        agencia: retorna o valor do atributo privado agencia;
        numero: retorna o valor do atributo privado numero;
        saldo: retorna o valor do atributo privado saldo;
        ativa: retorna o valor do atributo privado ativa;


Parte 3) definir o seguinte setter:
        ativa: recebe um valor booleano (situacao), e atribui esse
valor ao atributo
        privado ativa.

        *OBS: antes de atribuir o valor recebido no setter ao atributo
privado ativa,
        você deve verificar se o tipo de dado do parâmetro situacao é
booleano;

        *DICA: Para verificar o tipo de dado de uma variável, você
pode utilizar
        a função isinstance().


Parte 4) implemente os seguintes métodos públicos:
        Item 4.1: depositar(): recebe um valor para depósito na conta,
adiciona
        esse valor ao saldo atual (atributo privado saldo), e adiciona
a seguinte
```

operação ao final da lista de operações (atributo privado operacoes): ('deposito', valor)

*OBS: Um depósito só pode ser feito se as seguintes condições forem atendidas:
a conta está ativa e o valor do depósito deve ser maior que zero. Você deve
implementar essas verificações.

Item 4.2: sacar(): recebe um valor para saque na conta, subtrai esse valor
do saldo atual (atributo privado saldo), e adiciona a seguinte operação ao
final da lista de operações (atributo privado operacoes): ('saque', valor)

*OBS: Um saque só pode ser feito se as seguintes condições forem atendidas:
a conta está ativa, o valor do saque deve ser maior que zero, e o valor do
saque não pode ser maior que o saldo atual da conta. Você deve implementar
essas verificações.

Item 4.3: transferir(): recebe dois parâmetros: a conta de destino, que é
um outro objeto já instanciado da classe Conta, e o valor da transferência.
Esse valor será subtraído do saldo atual (atributo privado saldo) da conta
de origem (o objeto referenciado pelo parâmetro self) e, em seguida, deve
ser depositado na conta de destino. Ao final, adiciona a seguinte operação
no fim da lista de operações (atributo privado operacoes): ('transferencia', valor)

*OBS: Uma transferencia só pode ser realizada se as seguintes condições
forem atendidas: a conta de origem e a conta de destino estão ativas (ao mesmo
tempo), o valor transferido deve ser maior que zero, e o valor transferido
não pode ser maior que o saldo atual da conta de origem. Você deve implementar
essas verificações.

Item 4.4: tirar_extrato(): retorna a lista de operações (o atributo privado
operacoes). Essa lista contém todas as operações feitas na conta (abertura da conta,

```
        deposito, saque, transferencia, etc). Essas informações devem
ser armazenadas
        na lista de operações como tuplas, conforme os exemplos da
tabela a seguir:
        ------------------------------------------------
                operação          |    entrada no extrato
        ---------------------|--------------------------
        * abertura da conta  | ('saldo inicial', 1000)
        * depósito           | ('deposito', 250)
        * saque              | ('saque', 100)
        * transferencia      | ('transferencia', 50)
        ------------------------------------------------

        *OBS: As operações devem aparecer na ordem em que foram
feitas, ou seja,
         o primeiro item da lista deverá sempre ser o de abertura da
conta (contendo
         o saldo inicial e o seu valor).

        *OBS: note que a descrição das operações contidas nas tuplas
não possui acentos.
        Você deve seguir exatamente esse padrão, utilizando letras
minúsculas e sem
        acentos.

"""

class Conta:

        def __init__(self, titular, agencia, numero, saldo_inicial):
                """
                Crie os atributos da classe e inicie seus respectivos
valores de acordo
                com as especificações.
                """
                pass

        @property
        def titular(self):
                """
                Implemente a property titular
                """
                pass

        @property
        def agencia(self):
                """
                Implemente a property agencia
                """
                pass

        @property
        def numero(self):
                """
                Implemente a property numero
                """
                pass
```

```
@property
def saldo(self):
        """
        Implemente a property saldo
        """
        pass

@property
def ativa(self):
        """
        Implemente a property ativa
        """
        pass

@ativa.setter
def ativa(self, situacao):
        """
        Implemente o setter ativa
        """
        pass

def depositar(self, valor):
        """
        Implemente o método depositar()
        """
        pass

def sacar(self, valor):
        """
        Implemente o método sacar()
        """
        pass

def transferir(self, conta_destino, valor):
        """
        Implemente o método transferencia()
        """
        pass

def tirar_extrato(self):
        """
        Implemente o método tirar_extrato()
        """
        pass
```