Exercícios – Sprint 4

Nesta sprint, iremos realizar uma manutenção em um sistema orientado a objeto. O sistema simula um banco.
O sistema orientado a objetos possui as classes: Agência, Conta, ContaPessoaFisica, ContaPessoaJuridica e ContaPoupanca. Esta sprint possui atividades de análise de código, criação/alteração de código e questões dissertativas, que deverão ser respondidas via formulário.

Legenda:

- Atividade marcada desta forma envolve criação de código. Somente os testes unitários estão prontos.
- **Atividade marcada desta forma está implementada e testada.**
- **Atividade marcada desta forma devem ser respondidas no formulário.**

Classe Conta

1) Criar a classe Conta. Desenvolver o método **construtor**. Este construtor deve receber via kwargs os parâmetros: nome, limite e saldo. No construtor, uma lista chamada extrato deve ser criada.Também devem ser criados os atributos nome, limite e saldo.
1.1) Caso o nome não seja informado, um erro de valor (ValueError) deve ser gerado: “Nome não informado”
1.2) Caso o limite seja inferior a zero, um erro de valor (ValueError) deve ser gerado: “Saldo negativo”
1.3) Caso o limite não seja informado, adotar o valor de 500.

2) Utilize o método help() para descobrir mais informações sobre o construtor da classe Conta:

```
In [1]: from banco.contas.Conta import Conta

In [2]: help(Conta)
```

Após observar a saída do método help, pressione a letra **q** para sair.

Ao criar as classes e métodos desta sprint, lembre-se de criar docstrings para que outros desenvolvedores entendam o que a classe e o método deve fazer.

Classe ContaPessoaFisica

3) Criar a classe ContaPessoaFisica. **Esta classe deve ser uma subclasse da classe Conta**. Desenvolver o método **construtor**. Este construtor deve receber via kwargs os parâmetros: nome, limite e saldo e cpf. No construtor, uma lista chamada extrato deve ser criada. Também devem ser criados os atributos nome, limite, saldo e cpf.
3.1) Caso o nome não seja informado, um erro de valor (ValueError) deve ser gerado: “Nome não informado”
3.2) Caso o limite seja inferior a zero, um erro de valor (ValueError) deve ser gerado: “Saldo negativo”
3.3) Caso o limite não seja informado, adotar o valor de 500.
3.4) Caso o cpf não seja informado, um erro de valor (ValueError) deve ser gerado: “CPF inválido”

4) Desenvolver o método **deposito**. Este método recebe um parâmetro inteiro ou número float. Este método deve incrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “D” de depósito e o segundo elemento com o valor do depósito.
4.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O depósito precisa ser numérico” será disparada.

4.2) Caso o parâmetro seja um número negativo, uma exception do tipo ValueError com a mensagem “'Valor do depósito precisa ser maior que zero” será disparada.

5) Desenvolver o método **saque**. Este método recebe um parâmetro inteiro ou número float. Este método deve decrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “S” de saque e o segundo elemento com o valor do depósito.
4.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O valor do saque precisa ser numérico” será disparada.
4.2) Caso o parâmetro seja um número maior do que o valor do saldo somado ao limite, uma exception do tipo ValueError com a mensagem “'Valor do saque supera seu saldo e seu limite” será disparada.

6) Desenvolver o método **get_extrato**. Este método não recebe parâmetro. Este método retorna a lista com o histórico de extratos da conta.

7) Desenvolva os métodos **__str__** e **__repr__** para representar o objeto instanciado. A saída deve seguir o modelo:

Conta PF:<**nome_cliente**>,saldo=<**saldo**>

Classe ContaPessoaJuridica

8) Criar a classe **ContaPessoaJuridica**. Esta classe deve ser uma subclasse da classe Conta. Desenvolver o método construtor. Este construtor deve receber via kwargs os parâmetros: nome, limite e saldo e cnpj. No construtor, uma lista chamada extrato deve ser criada. Também devem ser criados os atributos nome, limite, saldo e cnpj.
8.1) Caso o nome não seja informado, um erro de valor (ValueError) deve ser gerado: “Nome não informado”
8.2) Caso o limite seja inferior a zero, um erro de valor (ValueError) deve ser gerado: “Saldo negativo”
8.3) Caso o limite não seja informado, adotar o valor de 1500.
8.4) Caso o cpf não seja informado, um erro de valor (ValueError) deve ser gerado: “CNPJ inválido”

9) Desenvolver o método **deposito**. Este método recebe um parâmetro inteiro ou número float. Este método deve incrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “D” de depósito e o segundo elemento com o valor do depósito.
9.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O depósito precisa ser numérico” será disparada.
9.2) Caso o parâmetro seja um número negativo, uma exception do tipo ValueError com a mensagem “'Valor do depósito precisa ser maior que zero” será disparada.

10) Desenvolver o método **saque**. Este método recebe um parâmetro inteiro ou número float. Este método deve decrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “S” de saque e o segundo elemento com o valor do depósito.
9.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O valor do saque precisa ser numérico” será disparada.
9.2) Caso o parâmetro seja um número maior do que o valor do saldo somado ao limite, uma exception do tipo ValueError com a mensagem “'Valor do saque supera seu saldo e seu limite” será disparada.

11) Desenvolver o método **get_extrato**. Este método não recebe parâmetro. Este método retorna a lista com o histórico de extratos da conta.

12) Desenvolva os métodos **__str__** e **__repr__** para representar o objeto instanciado. A saída deve seguir o modelo:

Conta PJ:<**nome_cliente**>,saldo=<**saldo**>

Classe ContaPoupanca

13) Criar a classe ContaPoupanca. **Esta classe deve ser uma subclasse da classe Conta**. Desenvolver o método **construtor**. Este construtor deve receber via kwargs os parâmetros: nome, saldo e cpf. No construtor, uma lista chamada extrato deve ser criada. Também devem ser criados os atributos nome, saldo e cpf.
13.1) Caso o nome não seja informado, um erro de valor (ValueError) deve ser gerado: “Nome não informado”
13.2) Caso o limite seja inferior a zero, um erro de valor (ValueError) deve ser gerado: “Saldo negativo”
11.3) O limite não deve ser informado. Caso seja informado, adotar o valor padrão de 0.
13.4) Caso o cpf não seja informado, um erro de valor (ValueError) deve ser gerado: “CPF inválido”

14) Desenvolver o método **deposito**. Este método recebe um parâmetro inteiro ou número float. Este método deve incrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “D” de depósito e o segundo elemento com o valor do depósito.
14.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O depósito precisa ser numérico” será disparada.
14.2) Caso o parâmetro seja um número negativo, uma exception do tipo ValueError com a mensagem “'Valor do depósito precisa ser maior que zero” será disparada.

15) Desenvolver o método **saque**. Este método recebe um parâmetro inteiro ou número float. Este método deve decrementar o saldo da conta e adicionar uma tupla na lista extrato. Esta tupla deve conter dois elementos: o primeiro a letra “S” de saque e o segundo elemento com o valor do depósito.
15.1) Caso o parâmetro não seja do tipo indicado, uma exception do tipo TypeError com a mensagem “O valor do saque precisa ser numérico” será disparada.
15.2) Caso o parâmetro seja um número maior do que o valor do saldo somado ao limite, uma exception do tipo ValueError com a mensagem “'Valor do saque supera seu saldo.” será disparada.

16) Desenvolver o método **get_extrato**. Este método não recebe parâmetro. Este método retorna a lista com o histórico de extratos da conta.

17) Desenvolva os métodos **__str__** e **__repr__** para representar o objeto instanciado. A saída deve seguir o modelo:

Conta Poupança:<**nome_cliente**>,saldo=<**saldo**>

Classe Agencia

18) Criar a classe Agencia. Desenvolver o método **construtor**. Este construtor deve receber uma string, com o nome da agência. No construtor, uma lista chamada clientes deve ser criada.

19) Desenvolver o método **add_conta_PF**. Este método recebe um parâmetro nome, cpf, saldo e limite. Caso o limite não seja passado, adotar None como padrão.
Este método deve instanciar um objeto ContaPessoaFisica e adicionar (com append) a lista de clientes.

20) Desenvolver o método **add_conta_PJ**. Este método recebe um parâmetro nome, cnpj, saldo e limite. Caso o limite não seja passado, adotar None como padrão.
Este método deve instanciar um objeto ContaPessoaJuridica e adicionar (com append) a lista de clientes.

21) Desenvolver o método **add_conta_poupanca**. Este método recebe um parâmetro nome, cpf, saldo. Caso o limite não seja passado, adotar None como padrão.
Este método deve instanciar um objeto ContaPoupanca e adicionar (com append) a lista de clientes.

Refatoração do código

Neste momento, espera-se que todos os testes estejam plenamente funcionando. Rode o pytest e certifique-se que nenhum erro ocorre.

22)
Mover o método **get_extrato** da classe **ContaPessoaFisica** para a classe **Conta**.
Remover o método **get_extrato** da classe **ContaPessoaJuridica.**
Remover o método **get_extrato** da classe **ContaPoupanca.**

Com as alterações propostas no exercício 22, houve algum teste que falhou ? Caso sim, quais. Caso não, explique porque. Se houve algum efeito colateral, voltar a alteração antes dos testes quebrarem.

23)
Mover o método **deposito** da classe **ContaPessoaJuridica** para a classe **Conta**.
Remover o método **deposito** da classe **ContaPessoaFisica.**
Remover o método **deposito** da classe **ContaPoupanca**.

Com as alterações propostas no exercício 23, houve algum teste que falhou ? Caso sim, quais. Caso não, explique porque. Se houve algum efeito colateral, voltar a alteração antes dos testes quebrarem.

24) Mover o método **saque** da classe **ContaPessoaJuridica** para a classe **Conta**.
Remover o método **saque** da classe **ContaPessoaFisica.**

Com as alterações propostas no exercício 24, houve algum teste que falhou ? Caso sim, quais. Caso não, explique porque. Se houve algum efeito colateral, voltar a alteração antes dos testes quebrarem.

25) Remover o método **saque** da classe **ContaPoupanca**.

Com as alterações propostas no exercício 25, houve algum teste que falhou ? Caso sim, quais. Caso não, explique porque. Se houve algum efeito colateral, voltar a alteração antes dos testes quebrarem.