Código: 98H04-06 POD - Programação Orientada a Dados
Prof. Dr. Dalvan Griebler, dalvan.griebler@pucrs.br

Atualizado: 29/09/2021

Trabalho Individual sobre Python

Sistema e Aplicação do Mercado Financeiro

Contexto

Durante a disciplina de programação orientada a dados, vocês estão aprendendo a programar na linguagem de programação em **Python**, que é multi-paradigma. Neste trabalho, o objetivo é avaliá-lo relativo a parte de Programação Orientada a Objetos (POO), Módulos e Manipulação de Arquivos. O trabalho também será uma forma de exercitar e aplicar os conceitos de forma prática.

No **TecnoPuc** existe um grupo de estudantes de administração que decidiu criar uma startup financeira. Eles precisam desenvolver uma solução para gerenciar os dados dos clientes e permitir que sejam feitas operações financeiras. Porém, nenhum deles sabe programar e estão procurando por pessoas aptas para ajudar no desenvolvimento deste projeto. Portanto, o professor foi designado para criar uma lista de requisitos para este protótipo, com o objetivo de avaliar o seu conhecimento e o quão apto você está para atuar no desenvolvimento deste projeto.

Restrições

- Está permitido o uso de todos os métodos da classe **str** (string).
- Apenas os módulos **abc** e **sys** podem ser importados para implementar o sistema ou a aplicação bancária.
- Outros métodos como **len()** que são nativos e não dependem de importação de módulos estão autorizados (lista de métodos). **Métodos que não foram apresentados em aula, devem ser justificados e explicados através de comentário no código**.

Especificações

Você precisa desenvolver um sistema bancário que possa ser usado como um pacote do Python e também uma aplicação em Python que utilize este pacote para realizar uma sequência de operações que estarão pré-definidas em um arquivo.

Prof. Dr. Dalvan Griebler, dalvan.griebler@pucrs.br

Especificações e requisitos do sistema bancário:

1) Todo este sistema bancário deve ser implementado como um pacote do Python chamado **Banco**, composto de um módulo chamado **Contas**.
2) Espera-se que para a criação das contas bancárias possa ser feito o uso da palavra reservada **with**.
3) Implemente uma classe chamada **Moeda** para representar o tipo de moeda que a startup pretende trabalhar. Isso significa que todo valor depositado e sacado precisa ser deste tipo de moeda. As operações matemáticas com e sem atribuição, bem como lógicos de comparação precisam ser suportadas também com objetos do mesmo tipo e números de ponto flutuante.
4) Precisa ter uma Classe chamada **ContaBancaria**, a qual deve ser do tipo abstrata e será estendida pelas Subclasses.
5) Precisa ter uma Subclasses para cada tipo de conta bancária:
    a) Conta corrente deve ser representada pela classe **ContaCorrente**
    b) Conta para investimento deve ser representada pela classe **ContaInvestimento**
    c) Conta poupança deve ser representada pela classe **ContaPoupanca**
6) Todas as contas bancárias precisam permitir realizar o depósito, saque e consultar o saldo.
7) Todas as contas bancárias precisam armazenar os seguintes dados:
    a) Número da conta (um número de 4 dígitos)
    b) Nome completo do cliente (uma string com no máximo 50 caracteres)
    c) CPF (um número de 11 dígitos)
    d) Saldo (número positivo)
8) A conta corrente precisa ter um valor limite (é o limite do cheque especial) para saque que será especificado na criação da conta. Esse limite do cheque especial precisa ser menor ou igual a 0. Sua taxa de rendimento é fixada em 0.01 ao mês.
9) A conta poupança precisa ter uma taxa de rendimento ao mês associada (de 0 a 1) que será especificado na criação da conta.
10) Todas as contas precisam implementar um método para consultar qual é o rendimento (ex. Se a taxa for 0.1 e o saldo for 100, o rendimento mensal é de 10) dado um número de dias informado. Este deve ser um contrato da classe abstrata.
11) A conta de investimento só pode ser de três tipos de risco (baixo, médio e alto) que será especificado somente na criação da conta, tendo fixado as respectivas taxas de rendimento:
    a) Baixo possui uma taxa de rendimento de 0.1 ao mês

   b) Médio possui uma taxa de rendimento de 0.25 ao mês
   c) Alto possui uma taxa de rendimento de 0.5 ao mês
12) É necessário ter um controle de quantas contas bancárias foram criadas no total e também para cada tipo de conta. Forneça um função (**status_contas**) no módulo **Contas** para mostrar ao final da aplicação o status de criação de contas.
13) É necessário implementar controle de acesso aos dados das contas bancárias. Você pode optar por usar decoradores ou descritores.
14) É necessário que sejam implementadas classes para tratamento de exceções quando ocorrerem transações bancárias inadequadas. Por exemplo, valores negativos e saque sem saldo ou acima do limite.
15) É necessário também tratar erros com tipos de dados inválidos e incoerentes. Não poderão ser criadas contas com dados inválidos (ex: nome, conta, CPF, ...).
16) É necessário também tratar o acesso aos atributos e métodos desconhecidos.
17) Forneça uma função genérica chamada **saque_verboso(objeto,valor)** no módulo **Contas** que realize em sequencia/ordem as seguintes operações para qualquer um dos tipos de contas:
   a) Mostrar todos os dados da conta
   b) Sacar um valor
   c) Mostra o saldo
18) A função **saque_verboso(objeto,valor)** deve ser usada para realizar o saque de um valor na conta corrente.

Especificações e requisitos da aplicação bancária de teste:

1) A aplicação deve receber como argumento na linha de comando uma lista de arquivos texto (**1.txt, 2.txt, 3.txt, ...**), onde cada arquivo representa um indivíduo com seus dados e uma sequência de ações/operações/transações que devem ser feitas pelo aplicativo da startup financeira, caso obter sucesso na criação da conta bancária.
2) Após concluídas todas as ações/operações/transações contidas no arquivo de texto recebido como argumento da linha de comando, os dados da conta precisam ser armazenados em um arquivo de saída (**1.saida, 2.saida, 3.saida, ...**).
3) A aplicação não pode parar quando se deparar com ações/operações/transações não permitidas ou inválidas. Os erros devem ser armazenados em um arquivo de log, sendo um para cada indivíduo (**1.log, 2.log, 3.log, ...**). Por exemplo, se uma conta de um indivíduo não foi possível

criar, o aplicativo deve progredir para o próximo transação de outra conta ou passar para o próximo indivíduo. Este arquivo de log serve para monitorar o que foi possível realizar ou não.

Relato

Faça um resumo de no máximo 400 palavras relatando as dificuldades, desafios ou outras observações que achar relevante comentar sobre a realização deste trabalho. Você também pode comentar sobre sua evolução no conhecimento. Esse resumo deve ser enviado junto dos demais arquivos.

Entrega

Vocês devem enviar um arquivo compactado .zip contendo:

- **Banco** (diretório do pacote)
- **Individuos** (diretório aonde devem estar os arquivos lidos e gerados)
- **app.py** (aplicativo bancário)
- **README.md** (arquivo descrevendo como usar o aplicativo e o pacote)
- **RESUMO.md** (arquivo contendo o resumo de 400 palavras)