Universidade Federal do Ceará
Departamento de Computação

Disciplina	Técnicas de Programação para Ciência de Dados (CK0442) - T02	**Semestre**	2024.2
Professor	Lincoln Souza Rocha		
Trabalho Prático Final			

Descrição do Programa: MyTwitter
O MyTwitter é uma versão simplificada do popular serviço de *micro-blogging* Twitter (atual X). Os usuários, pessoas ou empresas, podem ser cadastrados no serviço por meio da criação de perfis específicos. Os usuários podem *tweetar*, visualizar seus *tweets*, visualizar sua *timeline* e seguir outros usuários. Além disso, é possível saber o número e listar os seguidores de um perfil particular.

1) Da Formação dos Grupos

Os grupos deverão ser formados por até 5 alunos. Cada grupo deve eleger um líder, que tratará com o professor das questões descritas nos próximos tópicos. Como todo trabalho em grupo, a cada membro deverá ser atribuído um conjunto de responsabilidades e informado no relatório final do trabalho.

2) Da Entrega do Projeto

O projeto deverá ser entre em duas etapas. **Etapa 1 (até dia 24/02/2025)**: envio via SIGAA do código fonte e do relatório final contendo: (i) um mapeamento dos responsáveis pela implementação de cada funcionalidade; e (ii) um vídeo gravado disponível no YouTube[1], demonstrando o programa em funcionamento com cobertura de todas as funcionalidades implementadas. **Etapa 2 (até dia 26/02/2025):** uma arguição sobre o código fonte do programa, participação obrigatória de todo o grupo.

3) Dos Critérios de Avaliação

O seu trabalho será avaliado em duas dimensões, a saber:

- **Dimensão 1**: FUNCIONALIDADE, COBERTURA E CORRETUDE
  - Essa dimensão tem o propósito de avaliar (i) o quanto dos requisitos obrigatórios (Seção 4) foram atendidos; (ii) o quanto das funcionalidades propostas na especificação do projeto foram implementadas; e (iii) se a implementação de cada funcionalidade foi feita de forma correta.

- **Dimensão 2**: DOMÍNIO, LEGIBILIDADE E BOAS PRÁTICAS
  - Essa dimensão tem como propósito avaliar o domínio de cada membro do grupo sobre a implementação do projeto entregue. Lembre-se que todo o grupo deve ser capaz de explicar qualquer parte do código fonte do programa.

[1]Fique atento para a qualidade da imagem e do áudio do vídeo. Veja algumas dicas nos seguintes links:
https://canaltech.com.br/carreira/como-gravar-videoaulas-para-alunos/
https://blog.hotmart.com/pt-br/como-fazer-videoaulas-atraentes/

4) Dos Requisitos Funcionais

ATENÇÃO! *Testes unitários deverão ser feitos para todas as classes do sistema evidenciando a implementação das funcionalidades requeridas*.

Classe	`Tweet`
Construtor	O construtor dessa classe deve receber como parâmetro o nome de usuário e armazená-lo no atributo `usuario`, o texto da mensagem da postagem e armazená-lo no atributo `mensagem`, atribuir automaticamente a hora e data[2] da criação do *tweet* ao atributo `data_postagem` e, usando a função `next()` sobre uma função geradora[3] passada como argumento `gerador_id`, gerar o identificador do *tweet* e armazená-lo no atributo `id`.
Atributos	• `id` (privado): identificador único do *tweet*. • `usuario` (privado): que guarda o nome de usuário que faz a postagem. • `mensagem` (privado): que guarda o texto da postagem. • `data_postagem` (privado): que guarda a data e hora da postagem.
Métodos	• `get_id` (público): responsável por acessar o atributo `id`. • `get_usuario` (público): responsável por acessar o atributo `usuario`. • `get_mensagem` (público): responsável por acessar o atributo `mensagem`. • `get_data_postagem` (público): responsável por acessar o atributo `data_postagem`.

Classe	`Perfil`
Construtor	O construtor da classe recebe o nome do usuário como argumento e deve atribuí-lo ao atributo `usuario`, além de inicializar os vetores e tornar o perfil ativo.
Atributos	• `usuario` (privado): que guarda o nome de usuário do perfil, por exemplo, @lincolnsrocha. • `seguidos` (privado): que guarda os perfis que são seguidos pelo usuário em questão. • `seguidores` (privado): que guarda os perfis que seguem o usuário em questão. • `tweets` (privado) que armazena os *tweets* do perfil em questão. • `ativo` (privado) que indica se o perfil em questão está ativo ou não.
Métodos	• `add_seguidor` (público): responsável por adicionar seguidores no atributo `seguidores` • `add_seguidos` (público): responsável por adicionar perfís seguidos no atributo `seguidos` • `add_tweet` (público): responsável por adicionar *tweets* no campo `tweets`.

[2] Dica! use `datetime.today()` do módulo `datetime` do Python para obter data e hora correntes do sistema operacional. Detalhes no link: https://docs.python.org/pt-br/3.13/library/datetime.html

[3] Essa função geradora deve ser definida em um contexto global do sistema.

	• `get_tweets` (público): responsável por recuperar todos os *tweets* do perfil. OBS: *tweets* dessa lista devem ser ordenados pela data de postagem. • `get_tweet` (público): responsável por recuperar um *tweets* específico do perfil. Ele recebe como parâmetro o id do *tweet*. • `get_timeline` (publico): responsável por criar uma lista de *tweets* que incluem os *tweets* do perfil e dos perfís seguidos. OBS: *tweets* dessa lista devem ser ordenados pela data de postagem. • `set_usuario` e `get_usuario` (públicos): são, respectivamente, métodos de atribuição e de acesso ao campo `usuario`. • `set_ativo` e `is_ativo` (públicos): são, respectivamente, métodos de atribuição e de acesso ao campo `ativo`.

CLASSE	`PessoaFisica` (subclasse de `Perfil`)
CONSTRUTOR	O construtor da classe recebe dois parâmetros (i) o nome do usuário que deve ser repassado ao construtor da super classe; e (ii) `cpf`, que representa o CPF associado ao perfil e que deve ser atribuído ao atributo `cpf`.
ATRIBUTOS	• `cpf` (privado): CPF associado ao perfil.
MÉTODOS	• `get_cpf` (público): responsável acessar o atributo `cpf`.

CLASSE	`PessoaJuridica` (subclasse de `Perfil`)
CONSTRUTOR	O construtor da classe recebe dois parâmetros (i) o nome do usuário que deve ser repassado ao construtor da super classe; e (ii) `cnpj`, que representa o CNPJ associado ao perfil e que deve ser atribuído ao atributo `cnpj`.
ATRIBUTOS	• `cnpj` (privado): CNPJ associado ao perfil.
MÉTODOS	• `get_cnpj` (público): responsável acessar o atributo `cnpj`.

CLASSE	`RepositorioUsuarios`
CONSTRUTOR	O construtor da classe inicializar um vetor onde será armazenado todos os perfis criados no sistema.
ATRIBUTOS	• `usuarios` (privado): uma lista de perfís.
MÉTODOS	• `cadastrar` (público): é responsável por cadastrar perfis de usuários. Restrição, não devem ser cadastrados usuários com o mesmo nome de

	usuário, caso isso ocorra uma exceção de usuário já cadastrado (`UJCException`) deve ser levantada. ● `buscar` (público): é responsável por procurar o perfil de usuário pelo seu nome de usuário. Deve retornar o perfil de usuário solicitado ou `None` em caso contrário. ● `atualizar` (público): é responsável por atualizar o perfil de usuário com base nas informações do `perfil` passado como argumento, caso o usuário do perfil informado não exista, uma exceção de usuário não cadastrado (`UNCException`) deve ser levantada.

CLASSE	`MyTwitter`
CONSTRUTOR	O construtor da classe inicializar o `repositorio` onde será armazenado todos os perfis criados no sistema.
ATRIBUTOS	● `repositorio` (privado): é um objeto da classe `RepositorioUsuarios`. onde os usuários são armazenados.
MÉTODOS	● `criar_perfil()`: é responsável por cadastrar o perfil passado como argumento no repositório de usuários. ○ **Restrição**: não pode existir mais de um perfil com o mesmo nome de usuário. Nesse caso, se já existir um perfil com o mesmo nome de usuário, uma exceção de perfil existente (`PEException`) deve ser levantada. ● `cancelar_perfil()`: é responsável desativar o perfil do nome de usuário passado como argumento. ○ **Restrições**: o perfil do nome de usuário deve existir e estar ativo. Caso o perfil do nome de usuário informado não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada. ● `tweetar()`: é responsável pela postagem de mensagens no *micro-blog*. Esse método deve utilizar os argumentos passados (nome de usuário do perfil dono da postagem e o texto da postagem) para instanciar um *tweet* e adicioná-lo nos *tweets* do perfil do usuário em questão. ○ **Restrições**: o perfil do nome de usuário deve existir e estar ativo, o tamanho da mensagem deve ser entre 1 e 140 caracteres. Caso o perfil do nome de usuário informado não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Caso a mensagem não esteja no limiar entre 1 e 140 caracteres, uma exceção de mensagem fora do padrão (`MFPException`) deve ser levantada. ● `timeline()`: é responsável por recuperar todos os *tweets* da *timeline* do perfil do nome de usuário informado como argumento. ○ **Restrições**: o perfil do nome de usuário informado deve existir e estar ativo. Caso o perfil do nome de usuário informado não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada.

	• `tweets()`: é responsável por recuperar todos os *tweets* postados pelo perfil do nome de usuário informado como argumento.
	  ○ **Restrições**: o perfil do usuário deve existir e estar ativo. Caso o perfil do usuário não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada.
	• `seguir()`: é responsável por incluir o perfil do nome de usuário ***seguidor*** na lista de seguidores do perfil do nome de usuário ***seguido***, e, adicionalmente, incluir o perfil do usuário ***seguido*** na lista de seguidos do perfil ***seguidor***.
	  ○ **Restrições**: o perfil dos usuários ***seguido*** e ***seguidor*** devem existir e estarem ativos e um usuário não pode seguir a si mesmo. Caso o perfil dos usuários ***seguido*** e/ou ***seguidor*** não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil dos usuários ***seguido*** e/ou ***seguidor*** exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada. Caso o nome de usuário do ***seguidor*** seja o mesmo do ***seguido***, uma exceção de seguidor inválido (`SIException`) deve ser levantada.
	• `numero_seguidores()`: é responsável por retornar o número de seguidores do perfil do nome de usuário informado como argumento.
	  ○ **Restrições**: o perfil do nome de usuário passado como argumento deve existir e estar ativo e só devem ser levados em consideração os seguidores cujos perfis existam e estejam ativos. Caso o perfil do usuário não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada.
	• `seguidores()`: é responsável por recuperar todos os seguidores do perfil do nome de usuário passado como argumento.
	  ○ **Restrições**: o perfil do nome de usuário passado como argumento deve existir e estar ativo e só devem ser levados em consideração os seguidores cujos perfis existam e estejam ativos. Caso o perfil do nome de usuário não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do nome de usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada.
	• `seguidos()`: é responsável por recuperar todos os seguidos pelo perfil do nome de usuário informado como argumento.
	  ○ **Restrições**: o perfil do nome de usuário passado como argumento deve existir e estar ativo e só devem ser levados em consideração os seguidos cujos perfis existam e estejam ativos. Caso o perfil do nome de usuário não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Porém, caso o perfil do nome de usuário exista, mas esteja inativo, uma exceção de perfil desativado (`PDException`) deve ser levantada.