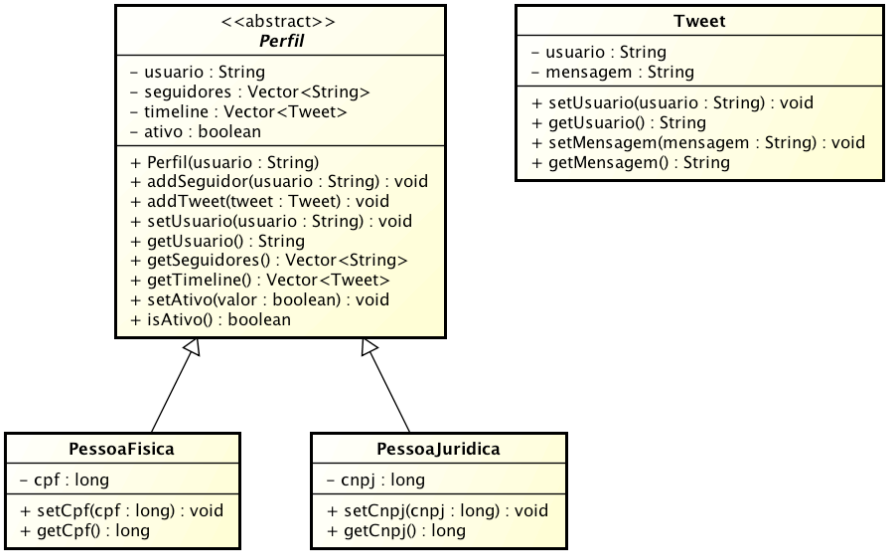
Universidade Federal do Ceará
Departamento de Computação

Disciplina	Técnicas de Programação I - CK0112	**Semestre**	2017/1
Professor	João Bosco Ferreira Filho		
Trabalho Prático Final - INDIVIDUAL		**Data Máxima de Entrega**	19/06/2017

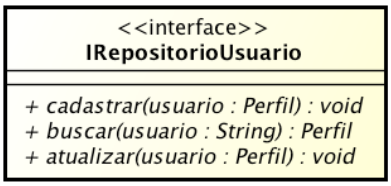
Descrição do Programa: MyTwitter
O MyTwitter é uma versão simplifica do popular serviço de *micro-blogging* Twitter. Os usuários, pessoas ou empresas, podem ser cadastrados no serviço por meio da criação de perfis específicos. Os usuários podem *tweetar*, visualizar seus *tweets*, visualizar sua *timeline*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

1) DESCRIÇÃO FUNCIONAL

Forneça uma implementação para as classes `Perfil`, `PessoaFisica`, `PessoaJuridica` e `Tweet`, conforme a especificação dada no diagrama abaixo. A classe `Perfil` é uma classe abstrata que possui como atributos privados: `usuario` (que guarda o nome de usuário do perfil, por exemplo, @boscoferreira), `seguidores` (que guarda o nome de usuário dos perfis que seguem o usuário em questão), `timeline` (que armazena os *tweets* do perfil em questão e os *tweets* dos usuários que o perfil em questão segue) e `ativo` (que indica se o perfil está ativo ou não). Os métodos, `addSeguidor` e `addTweet` são responsáveis, respectivamente, por adicionar seguidores no atributo `seguidores` e *tweets* no campo `timeline`. Os métodos `setUsuario` e `getUsuario` são, respectivamente, métodos de atribuição e de acesso ao campo `usuario`. Os métodos `getSeguidores` e `getTimeline` são responsáveis, respectivamente, por retornar a lista de nome de usuários dos `seguidores` e *tweets* da `timeline` do usuário em questão. Os métodos `setAtivo` e `isAtivo` são, respectivamente, métodos de atribuição e de acesso ao campo `ativo`. O construtor da classe `Perfil` recebe o nome do usuário como argumento e deve atribuí-lo ao atributo `usuario`, além de inicializar os vetores e tornar o perfil ativo. As classes `PessoaFisica` e `PessoaJuridica` herdam da classe `Perfil` e representam, respectivamente, um perfil comum e um perfil de uma empresa. A classe `Tweet` representa um *tweet* que encapsula o nome de usuário de um perfil e a mensagem de texto postada por este usuário.

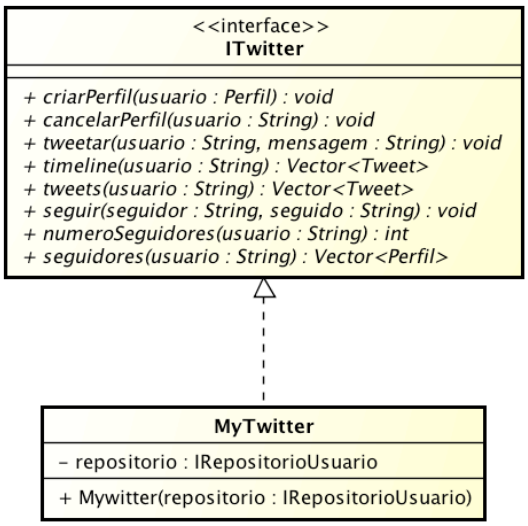


Para que o sistema funcione corretamente, é preciso criar um repositório que permita armazenar, buscar e atualizar perfis de usuários. Dessa forma, você deve fornecer uma implementação para a interface `IRepositorioUsuario` como descrito no diagrama abaixo. Essa implementação pode utilizar como estrutura de armazenamento a bibliotecas de classe do Java, o `Vector<Perfil>`.



- A implementação da interface `IRepositorioUsuario` deve prover os seguintes comportamentos para seus métodos:
    - Método `cadastrar()`: 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;
    - Método `buscar()`: é responsável por procurar o perfil de usuário pelo seu nome de usuário. Deve retornar o perfil de usuário solicitado ou `null` em caso contrário;
    - Método `atualizar()`: é responsável por atualizar o perfil de usuário com base nas informações o perfil passado como argumento, caso o usuário do perfil informado não exista, uma exceção de usuário não cadastrado (`UNCException`) deve ser levantada

As funcionalidades do serviço de *micro-blogging* são acessadas a partir da classe `MyTwitter`, descrita na figura abaixo. Esta classe possui um repositório de usuários como atributo, `repositorio` (`IRepositorioUsuario`), que é passado como argumento para o construtor, no momento da instanciação, e será utilizado na implementação do comportamento da classe `MyTwitter`. Além disso, essa classe implementa a interface `ITwitter` que descreve um contrato comportamental (conjunto de métodos) que devem ser implementados pela classe `MyTwitter`. O comportamento esperado de cada um dos métodos da interface `ITwitter` é explicado a seguir:



- Método `criarPerfil()`: 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.

- Método `cancelarPerfil()`: é responsável desativar o perfil do usuário passado 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.
- Método `tweetar()`: é responsável pela postagem de mensagens no *micro-blog*. Esse método deve utilizar os argumentos passados para instanciar um *tweet* e postá-lo na *timeline* do perfil do usuário e dos seus seguidores.
    - **Restrições**: o perfil do usuário deve existir e estar ativo, o tamanho da mensagem deve ser entre 1 e 140 caracteres e somente o perfil dos seguidores que existirem e estiverem ativos poderão receber a postagem. Caso o perfil do usuário não exista, uma exceção de perfil inexistente (`PIException`) deve ser levantada. Caso a mensagem na limiar entre 1 e 140 caracteres, uma exceção de mensagem fora do padrão (`MFPException`) deve ser levantada.
- Método `timeline()`: é responsável por recuperar todos os *tweets* da *timeline* do perfil do 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.
- Método `tweets()`: é responsável por recuperar todos os *tweets* postados pelo perfil do 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.
- Método `seguir()`: é responsável por incluir o perfil do usuário ***seguidor*** na lista de seguidores do perfil do usuário ***seguido***.
    - **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.
- Método `numeroSeguidores()`: é responsável por retornar o número de seguidores do perfil do usuário informado como argumento.
    - **Restrições**: o perfil do 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.
- Método `seguidores()`: é responsável por recuperar todos os seguidores do perfil do usuário passado como argumento.
    - **Restrições**: o perfil do 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.

2) CRITÉRIOS DE AVALIAÇÃO

O trabalho final é individual e vale 10 (dez) pontos, devendo ser entregue até o dia 19 de Junho de 2017. A entrega será realizada nas aulas que seguem o dia da entrega.

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

- Etapa 1: funcionalidade, cobertura e corretude
    - Nessa etapa serão aplicados testes funcionais automatizados ao programa com o propósito de verificar o quanto das funcionalidades solicitadas foram implementadas e se a implementação de cada funcionalidades foi feita de forma correta.
- Etapa 2: domínio, legibilidade e boas práticas
    - Nessa etapa o aluno fará uma apresentação do código fonte do programa onde perguntas serão feitas a respeito das soluções de programação dadas. Além disso, a legibilidade do código fonte bem como o uso de boas práticas serão considerados. A duração dessa etapa deve ser de no máximo 20 minutos.

BONIFICAÇÕES. Será concedido pontos extra para os alunos que utilizarem técnicas avançadas de programação em Java na implementação do trabalho: utilização de arquivos e desenvolvimento com interface gráfica Java Swing. Dependendo do quanto de cada técnica for **BEM** utilizada dentro do projeto, a bonificação pode chegar a até 2,0 pontos extras.