Relatório - Jackut

Alunos: João Lucas e Mateus dos Santos

Introdução:

O Jackut é um sistema que se assemelha a de uma rede social (como o *orkut)*, gerenciando usuários, seus interesses, relacionamentos, etc. A tarefa proposta foi a de projetar e implementar esse sistema, utilizando de um processo semelhante ao TDD (*Test-Driven Development)*, porém, no caso, construindo o sistema a partir dos testes já dados pelo instrutor. Dessa forma, implementamos de forma incremental o projeto, desenvolvendo cada pedaço do código após os erros levantados pelos testes.

Nesse relatório, iremos nos aprofundar sobre o nosso design do sistema, detalhando o porquê das nossas decisões do padrão que construímos. Também criamos o diagrama de classes que será usado de base para nossa explicação do projeto.

É importante destacar, também, que nesse projeto, escolhemos construí-lo simulando um banco de dados através de arquivos de texto, pois decidimos que, neste milestone o uso de um não se provaria necessário, porém acreditamos que a mudança, além de fácil feita, é possível para as próximas milestones que o sistema terá.

Desenvolvimento:

No Jackut, criamos essencialmente 5 classes principais:

- **Facade:**

Facade é a classe que irá fazer a comunicação com o easyAccept, contendo todos os métodos que o easyAccept reconhece até esse

“milestone 1”. Além disso, a classe possui uma instância da classe Sistema, que iremos abordá-la logo após. A maioria dos métodos desta classe apenas chamam outros métodos das classes que irão desempenhar aquela determinada ação. Porém, podemos destacar certas funções que apenas a classe Facade desempenha

- ***Facade:***

  Em seu construtor, a classe Facade desempenha a função de recuperar do banco de dados (arquivos de texto), as informações de usuários, amizades e recados, e as coloca na instância de Sistema, em seus devidos lugares.

- ***zerarSistema:***

  Apesar do Facade neste método chamar a função *zerarSIstema* da classe Sistema, ele também faz a ação de zerar o nosso “banco de dados”. Logo após ele deletar os arquivos, ele irá chamar o método *zerarSistema* da instância de Sistema.

- ***encerrarSistema:***

  O método de encerrarSIstema que o Facade desempenha, realiza a ação de salvar no banco de dados, aqueles dados que estão na instância de Sistema. Portanto é neste método que os dados que estavam sendo mantidos na instância de sistema, são persistidos no nosso “banco”.

- **Sistema:**

A classe Sistema é responsável por gerenciar os usuários cadastrados e as sessões que os usuários logados terão ao entrar no sistema. É nela que usuários que foram validados a estarem logados, terão suas sessões salvas no sistema.

Nesta classe, temos duas propriedades que valem a pena serem destacadas: ***usuarios***, que é um HashMap entre o login do usuário e a instância do mesmo, servindo como uma tabela de usuários cadastrados do sistema atualmente; e ***sessões****,* que é outro HashMap que mapeia o id único do usuário (falaremos dele logo após) com a instância do mesmo que está logado.

Podemos destacar as seguintes funções da classe Sistema:

- ***setSessaoUsuario:***

É nesse método que é atribuído àquele usuário que foi corretamente logado, o seu id de sessão. Optamos nesse projeto em utilizar, em cada usuário que está com sua sessão ativa, um id no formato UUID *(Universally Unique IDentifier)*, uma vez que, para cada usuário logado no sistema, ele deve possuir um id único que irá identificá-lo no sistema. Então, após ser gerado um UUID aleatório, atrelamos ele a instância do usuário logado.

- **Usuário:**

A classe de Usuário é a classe onde guardamos as informações mais importantes do usuário. É nela que guardamos seu nome, login, senha, seu perfil, seus amigos, como também suas solicitações de amizades e seus recados. Nesses últimos, é importante destacar que: o ***perfil*** é uma classe à parte, para ser mais fácil de gerenciar seus atributos, que podem ser quaisquer; os ***amigos*** é um ArrayList de usuários; tanto as ***solicitacaoEnviadas*** como ***solicitacoesRecebidas*** *são ArrayLists de usuários também; e os* ***recados*** é uma fila de Recados, que também é uma classe à parte.

  - ***enviarSolicitacao:***

    No método de ***enviarSolicitacao***, o usuário que enviará o solicitação de amizade, irá não só colocar em seu ArrayList de ***solicitacaoEnviadas***, o usuário que ele quer adicionar como amigo, como se adicionar no ArrayList de ***solicitacoesRecebidas*** do futuro amigo.

  - ***aceitarSolicitacao:***

    É nesse método que será efetivada a adição de amigos pelos dois usuários. O usuário que aceita a amizade, é retirado de sua lista de ***solicitacoesRecebidas*** e é adicionado o amigo, enquanto o "amigo", é retirado de sua lista de ***solicitacaoEnviadas***, e é adicionado o usuário.

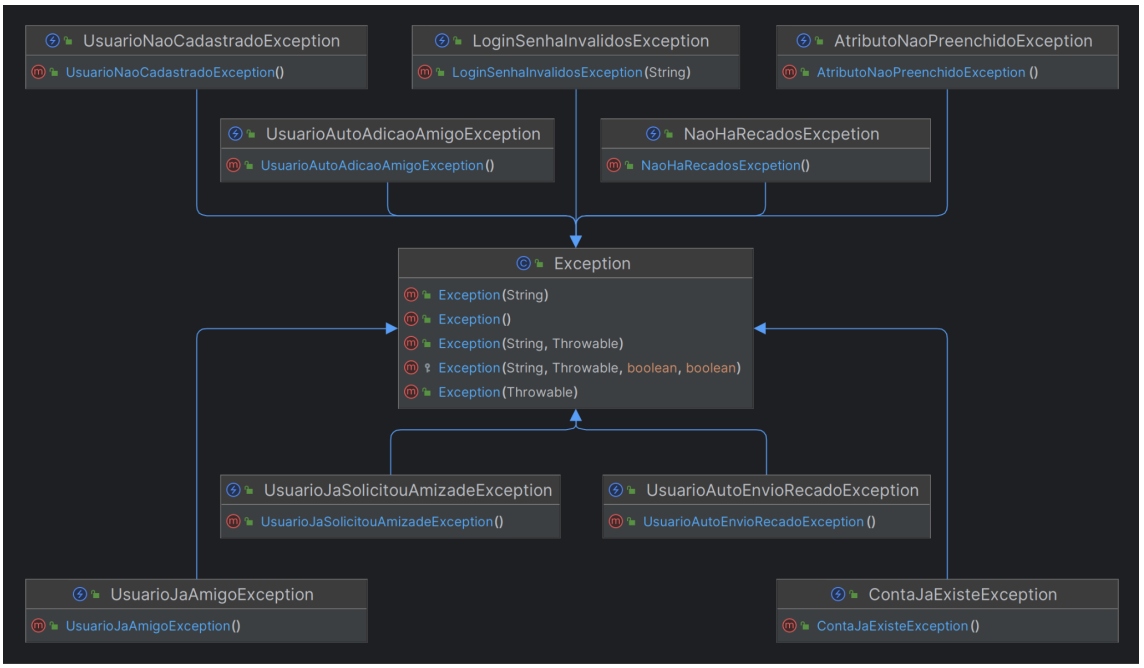
- **Perfil:**

A classe Perfil é utilizada pelo usuário, responsável por guardar seus atributos, que pela estrutura solicitada, pode ter qualquer chave e valor. Portanto, foi criada uma classe para gerenciar essa relação entre os atributos e o usuário. Seus métodos são simples getters e o seu construtor.

- **Recado:**

A classe de recado é utilizada pelo ***Usuário***, de forma que o usuário possui uma fila de *Recados*, onde cada recado possui seu *remetente*, *destinatário* e *recado.* Nós optamos por utilizar da estrutura de dados ***fila*** pois, uma vez que a estrutura do easyAccept é a de, após adicionar n recados, ler os recados a partir do primeiro que foi adicionado, esse é o clássico padrão de FIFO *(First In, First Out)*, portanto, escolhemos de fazer uma Queue de recados para o usuário. Esta classe possui como métodos, apenas getters para o uso da instância do usuário.

Exceções:



As exceções criadas nesse projeto, são simples exceções herdadas de RuntimeException que realizam a exceção específica que é às dada. Porém, uma exceção em específico que seria feita três ao todo, pôde ser resumida em apenas uma:

- **LoginSenhaInvalidosException:**

  Essa exceção cuida de 3 exceções ao mesmo tempo: “Login inválido.”, “Senha inválida.” e “Login ou senha inválidos.” Para fazer isso, colocamos uma string de tipo para determinar qual exceção seria feita ao usuário, sendo “login” para a de login inválido, “senha” para a de senha inválida, e “any” para as duas, uma vez que, no contexto da aplicação, foi pedido que, ao errar qualquer um dos campos no ***abrirSessao***, não ser informado qual foi o inválido, por isso o uso do “any”.