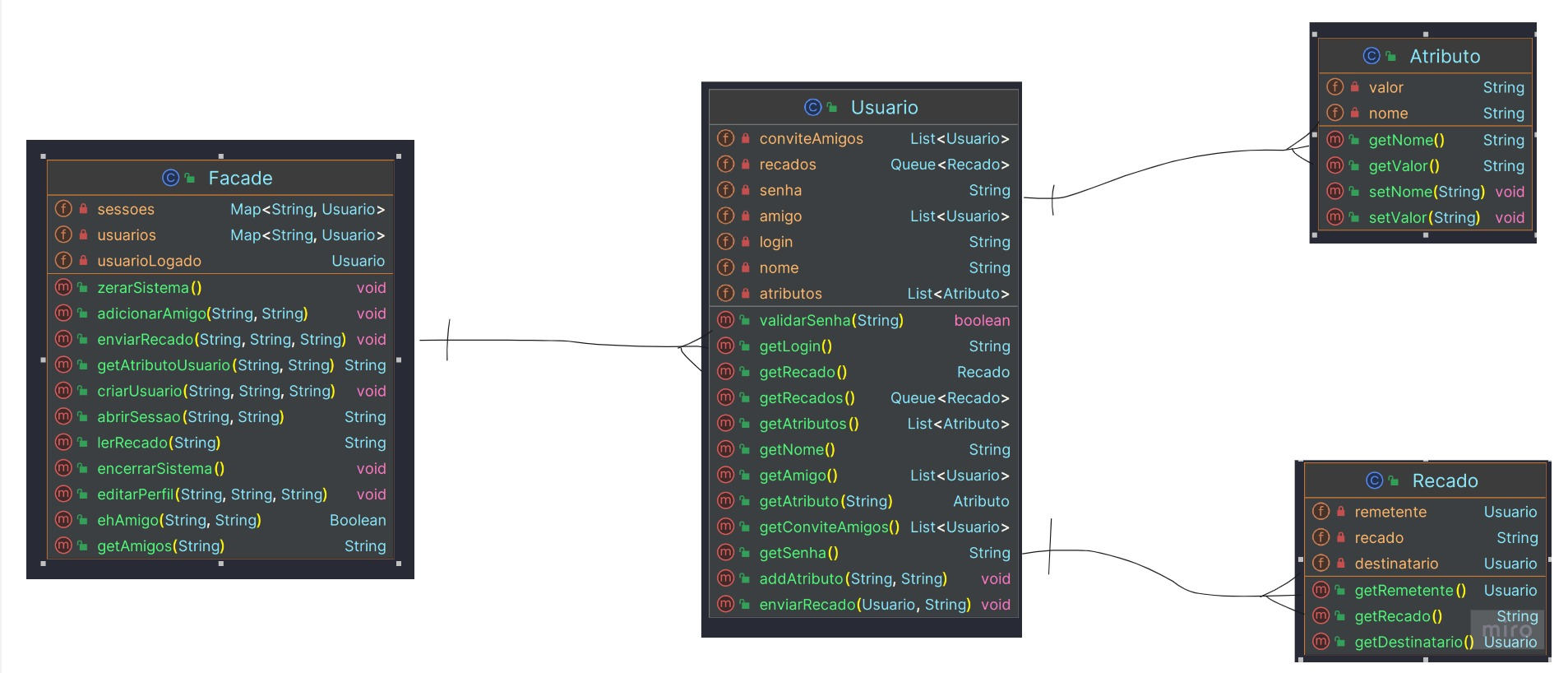
**Relatório de projeto Jackut**

**Alunos**: Ryan Barbalho, Davi Souza.

**Link do acesso ao GitHub**: https://github.com/davisouzal/projectOrkut

Diagrama de relacionamentos do Projeto:

O projeto consiste em uma implementação em java de uma rede social inspirada no orkut e possuindo as funcionalidades mais básicas de uma rede social, como: criação de conta, visualização de perfil, lista de amigos e enviar mensagens. O projeto foi estabelecido de acordo com a arquitetura de software padrão MVC, apesar de não possuir uma interface interativa. Além disso utilizamos alocação de dados em arquivos txt, visto que o primeiro projeto tem como objetivo a utilização dos conceitos da programação orientada a objetos.

O projeto é dividido em diferentes packages sendo eles models, que consistem nos modelos que futuramente serão implementados em um banco de dados, Exceptions, que consiste em um package exclusivo para as exceções que nós criamos e no package raiz (jackut), além dos mencionados agora, está o arquivo Facade que é onde está disposto o conteúdo principal de nossa aplicação e, por fim, a Main que serve apenas para executar o easy\_accept.

No arquivo Facade temos a seguinte composição:

como atributos temos:

private Map<String, Usuario> usuarios;

Map<String,String> de usuários, utilizamos dessa estrutura para definir as classes de usuários que estão manipuladas na execução. A key do map “String” consiste no login de usuario e o value, um objeto da classe usuario.

private Usuario usuarioLogado;

objeto de usuarioLogado que consiste no usuário da sessão atual, que é um objeto da classe Usuario.

private Map<String, Usuario> sessoes;

Map<String,Usuario> sessoes consiste no conjunto de sessoes abertas de usuarios utilizando a rede social. A key String consiste no id do Usuario e o Value, um objeto da classe Usuario.

A classe Facade possui um construtor que é chamado quando o arquivo é executado.

public Facade(){

try{

//verifica se ja arquivo ja existe. se nao, cria um novo

if(!new File("usuarios.txt").exists()){

new File("usuarios.txt").createNewFile();

this.usuarioLogado = null;

this.usuarios = new HashMap<>();

this.sessoes = new HashMap<>();

}

else{

this.usuarioLogado = null;

this.usuarios = new HashMap<>();

this.sessoes = new HashMap<>();

//buffered reader é melhor para ler arquivos grandes

BufferedReader reader = new BufferedReader(new FileReader("usuarios.txt"));

String line;

while((line = reader.readLine()) != null){

String[] user = line.split(";");

Usuario usuario = new Usuario(user[0], user[1], user[2]);

usuario.getAtributos().add(new Atributo("nome", user[2]));

//adiciona os atributos

for(int i = 3; i < user.length; i+=2){

usuario.getAtributos().add(new Atributo(user[i], user[i+1]));

}

usuarios.put(user[0], usuario);

}

reader.close();

}

} catch (IOException *e*) {

*e*.printStackTrace();

}

}

O metodo construtor possui um bloco try, que captura uma IOException caso possua algum erro de fluxo de arquivo, visto que o metodo faz uso de arquivos .txt, se houver erro, é printado um stackTrace do erro para facilitar a identificação da causa. daí, é verificada a condição de existir um arquivo de usuarios.txt, caso não haja, é criado o arquivo de usuarios.txt e os atributos de Facade são declarados. Caso exista um arquivo de usuarios.txt, os atributos de Facade são declarados e é utilizado um bufferedReader para ler a composição do arquivo .txt, onde os dados são dispostos da seguinte forma:

login1;senha1;nome1;atributo1\_nome;atributo1\_valor;atributo2\_nome;atributo2\_valor

login;senha2;nome2;atributo1\_nome;atributo1\_valor;atributo2\_nome;atributo2\_valor

com isso, os dados retirados do arquivo são usados para criar um objeto de Usuario e o objeto é inserido no map usuarios.

para zerar o Sistema é utilizado o seguinte metodo:

public void zerarSistema() {

usuarios.clear();

sessoes.clear();

usuarioLogado = null;

//deleta o arquivo de usuarios

new File("usuarios.txt").delete();

}

onde os maps de usuarios e sessões são esvaziados, o usuarioLogado é definido nulo e o arquivo usuarios.txt é apagado.

Já no metodo de encerrar o sistema:

public void encerrarSistema() {

usuarioLogado = null;

sessoes.clear();

//escreve no arquivo de usuarios

try {

BufferedWriter writer = new BufferedWriter(new FileWriter("usuarios.txt"));

for(Usuario user : usuarios.values()){

writer.write(user.getLogin()+";"+user.getSenha()+";"+user.getNome());

//tambem grava os atributos

for(Atributo atributo : user.getAtributos()){

writer.write(";" + atributo.getNome()+";"+atributo.getValor());

}

writer.newLine();

}

writer.close();

} catch (IOException *e*) {

*e*.printStackTrace();

}

}

o usuarioLogado é definido nulo e o map de sessões é esvaziado, após isso é os dados de usuários manipulados na execução são armazenados no arquivo usuarios.txt.

Tendo sido definida a estrutura básica do Facade, o processo de desenvolvimento do projeto se deu pela solução de erros da bateria de testes “user-stories”.

o user story 1\_1 se inicia com a chamada

criarUsuario login=jpsauve senha=sauvejp nome="Jacques Sauve"

assim, foi criado o metodo criar usuários que recebe as entradas informadas no header e possui duas verificações, sendo a primeira verificação se a entrada de login é vazia, se for, lança uma exceção de “Login inválido”, já a segunda verifica se a entrada de senha é nulo, se for, lança a exceção “Senha inválida.”. Logo em seguida o método verifica se o map de usuarios possui um login já utilizado, se possuir lança a exceção “Conta com esse nome já existe.”, se não houver, é criado um objeto usuario com as entradas e o primeiro atributo nome é inserido na Lista de atributos do usuário e o usuario é inserido no map de usuários.

o método seguinte a ser testado é getAtributoUsuario, onde o usuario é previamente cadastrado.

getAtributoUsuario login=jpsauve2 atributo=nome

No método implementado é instanciado um objeto do tipo Usuário cujos dados são obtidos do map de usuários no sistema usando o login inserido. Assim, é feita a verificação se o map de usuários possui a key do login inserido, se não tiver, lança a exceção “Usuário não cadastrado.”; se tiver verifica se o atributo é igual a nome, se for retorna o nome do usuário; depois verifica se o atributo presente na lista de atributos do usuario é diferente de nulo, se for, retorna o valor do atributo; por fim, se nenhum das duas verificações forem verdadeiras, lança a exceção “Atributo não preenchido.”O método getAtributo, que retorna um objeto de Atributo, que foi implementado na classe Usuario através de um for que percorre os atributos e verifica se é igual a String de atributo fornecida .

Depois de verificar diversas entradas supostamente erradas nos métodos implementados, se encerra o user story 1\_1.

O user story 1\_2 tem como principal funcionalidade exigida a permanência de dados em três a chamadas dos metodos de abrirsessão de um usuario criado no teste anterior seguido da chamada de metodo getAtributo para retornar o atributo usuario. O problema foi solucionado através do armazenamento de dados dos usuários no arquivo usuarios.txt, que é criado ou acessado na execução do sistema.

O user story 2\_1 consiste na implementação da funcionalidade do usuário editar/criar atributos de seu perfil ou preencher atributos inexistentes.

Para resolver os problemas apresentados no user Story e adicionar a funcionalidade de atributos de perfil, foi criada uma classe Atributos com os atributos private String nome e private String valor, dessa forma a classe Usuario possuirá uma lista de Atributos como atributo. A funcionalidade de editar atributos do perfil foi implementada através do método editarPerfil(String id, String atributo, String valor) onde é feita a verificação se o usuarioLogado é nulo, se for, então é lançada a Exceção “Usuário não cadastrado.”, caso contrário, é chamado o método get dos usuarios através do Login do usuarioLogado e assim chamado o método de addAtributo com os parametros fornecidos no header. O método addAtributo é implementado através da criação de um novo Atributo e na adição desse Atributo na lista de Atributos do Usuario.

O método getAtributoUsuario é explicado no último caso do user story 1\_1. No user story 2\_1 são adicionados diversos atributos e chamados os métodos de getAtributoUsuario para cada atributo adicionado.

No user story 2\_2 é verificada a persistência de dados após o sistema ser encerrado no fim do us 2\_1, assim o programa é iniciado e os dados são recuperados do arquivo txt como foi explicado no na sessão do sistema base, com isso são chamados os metodos de getAtributoUsuario chamados na última verificação do us 2-1, após isso o sistema é encerrado novamente.

O user story 3\_1 consiste em implementar a funcionalidade do usuário adicionar outro à sua lista de amigos e poder “aceitar” uma solicitação de amizade.

Isso foi solucionado através da criação de uma List<Usuario> amigo onde ficariam alocados os amigos do Usuario e uma List<Usuario> conviteAmigos. Para adicionar amigos foi implementado o metodo adicionarAmigo(String id, String loginAmigo) onde é criado um objeto de usuario obitdo através do id no map de sessões. Então é verificado se o objeto user é nullo, se for, é lançada a exceção Usuário não cadastrado, então é chamado o método abrirSessão com os atributos do objeto user criado então é criado o objeto amigo do tipo Usuario. Então, é feita a verificação se o userAmigo é nulo, se for, lança a exceção UserNotFound, depois é verificado se o Usuario amigo ja está adicionado (através do metodo ehAmigo, que retorna um valor Boolean se o Usuario for ou não amigo do user), se for, lança a exceção “Usuário já está adicionado como amigo.”, depois é verificado se o usuário ja foi adicionado mas não aceitou ainda o convite, se for, lança a exceção “Usuário já está adicionado como amigo, esperando aceitação do convite.”, depois é verificado se o userAmigo é o proprio Usuario, se for, lança a exceção “Usuário não pode adicionar a si mesmo como amigo.”, só então o userAmigo é adicionado na lista de convites amigos, se o userAmigo também possuir ambos são adicionados na lista de amigos um do outro.

Na user story 3\_1 também é implementado o método get amigos, onde fornecido o login do usuário e a função retorna uma String de uma lista de amigos que o usuário possui, isso foi feito através de um string builder onde cada nome do amigo é adicionado na String a ser retornada.

O user story 3\_2 consiste na verificação de persistencia de dados que é realizada através da alocação de dados no arquivo usuarios.txt. Porfim, o sistema é encerrado.

No user story 4\_1 começa a implementação da funcionalidade de recados, em que usuários podem mandar recados entre si. Foi criada uma classe para recado, no padrão MVC sendo uma model, contendo o remetente e o destinatário, ambos da classe Usuário, e o recado em si, sendo esse uma String e por fim seus getter e setters. Como vários usuários podiam enviar diferentes recados para um só usuário, foi adicionado uma Queue, feature do java.util, permitindo assim fazer uma fila de recados de um determinado usuário. Para esse teste, foi necessária a função enviarRecado, que é feita ao pegar o sender da sessão e tanto o destinatário quanto a mensagem são recebidas nos parâmetros da função. Se usuário não for encontrado cai na exception de userNotFound e se tentar enviar para si mesmo, cai em outra exception. Se tudo der certo, envia o recado. Por fim também foi necessário o lerRecado, que de novo trás apenas uma nova excessão (NaoHaRecados) e e chama o getRecado.

Já para a segunda user\_story da parte 4, como a persistência já foi implementada anteriormente, não foram necessárias alterações.Por fim, o sistema é encerrado.