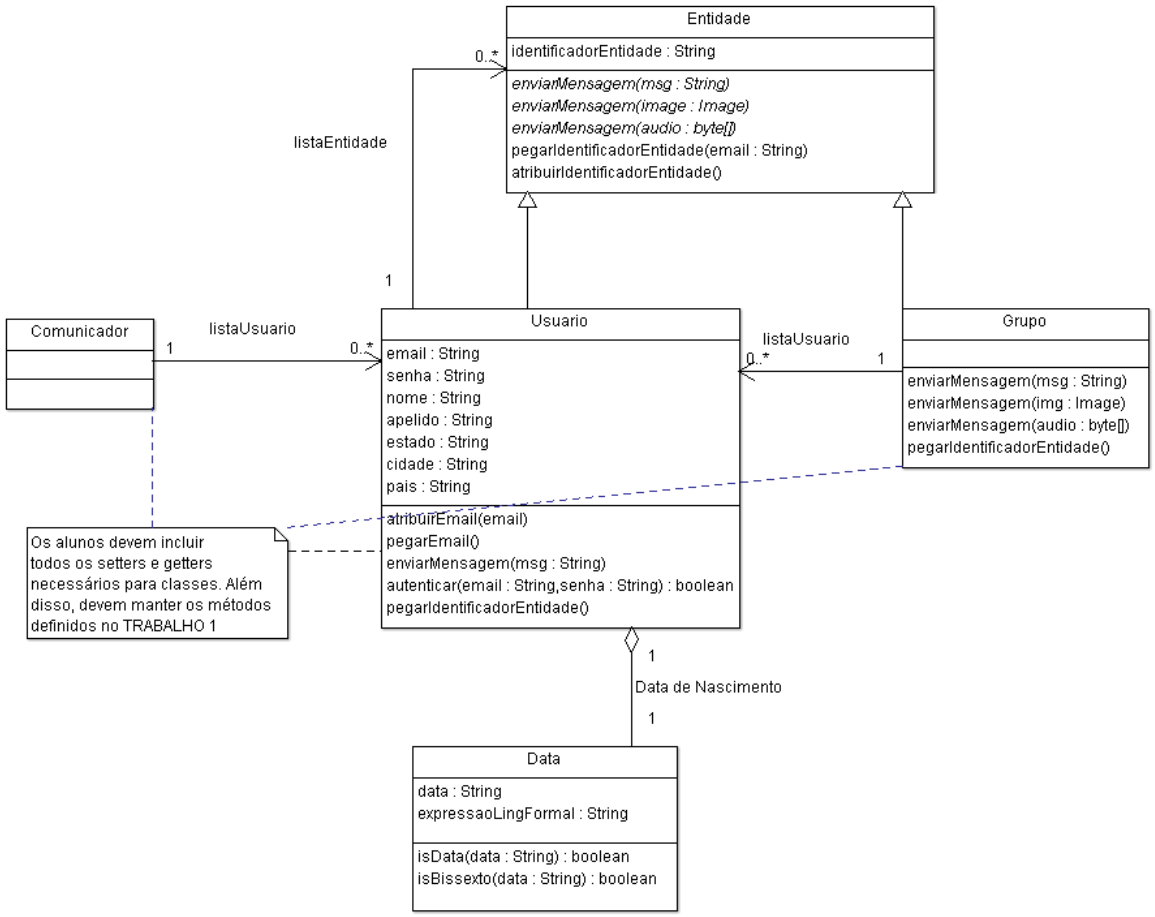
SCC 204 – Programação Orientada a Objetos
Trabalho 2

Todos os trabalhos da disciplina são parte de um projeto que envolve o desenvolvimento de um Comunicador Instantâneo. Esse trabalho apresenta o Diagrama de Classes em UML (Unified Modelling Language) abaixo:



Em primeiro lugar, ajuste seu programa de acordo com o diagrama acima. Em seguida, atualize o diagrama a fim de contemplar suas classes, variáveis membros, métodos e funcionalidades que utilizou no primeiro trabalho e que continuará utilizando neste segundo. Para editar o diagrama de classe acima, utilize a ferramenta **ArgoUML** e abra o arquivo chamado **trabalho2.zargo**.

Observe que este diagrama contém a classe **Entidade** e heranças para as classes **Usuario** e **Grupo**.

Além disso, neste trabalho, deve-se desenvolver as seguintes funcionalidades:

1) Validar a senha de um usuário dentro do método **autenticar(email, senha)** o qual está localizado

dentro da classe **Usuario**. Esse método recebe email e senha, verifica se usuário existe por meio de busca binária em arquivo de indexação **usuarios.idx** e carrega os demais dados do usuário contidos no arquivo binário de dados **usuarios.dat** para dentro do objeto da classe **Usuario**.

Na autenticação de um usuário, deve-se observar os seguintes casos:

Caso um usuário não exista na base **usuarios.idx** , deve-se imprimir a seguinte mensagem:

```
printf("Usuario nao encontrado.\n");  // sem acentuação
```

Caso o usuário exista na base **usuarios.idx**, deve-se imprimir:

```
printf("Usuario encontrado.\n");  // sem acentuação
```

Em seguida, deve-se verificar se a senha do usuário está correta. Caso esteja correta, imprima:

```
printf("Senha correta.\n");
```

Caso esteja incorreta, imprima:

```
printf("Senha incorreta.\n");
```

2) Criar uma classe **Data** para gerenciar datas no sistema (por exemplo, data de nascimento de usuários). Essa classe deve aceitar somente datas válidas. Deve-se suportar, também, anos bissextos. No caso da linguagem Java, procure mais informações sobre a classe **Calendar** e **Date** já disponíveis. No caso da linguagem C++, procure um diretório **data** disponibilizado nos exemplos de aula e procure mais sobre como descobrir quando um ano é bissexto e, como dica, defina o número esperado de dias para cada mês do ano.

Caso uma data esteja incorreta, imprima como saída:

```
printf("Data incorreta.\n");
```

Caso uma data esteja correta, imprima como saída:

```
printf("Data correta.\n");
```

3) De acordo com o novo diagrama de classes apresentado anteriormente, deve-se desenvolver uma classe **Entidade** a partir da qual as classes **Usuario** e **Grupo** herdam. Observe que cada usuário tem um vetor de entidades, as quais podem tanto ser outros usuários quanto grupos. Quando um usuário A deseja enviar uma mensagem para uma entidade B, esse usuário a procura em seu vetor de contatos. Em seguida invoca o método **enviarMensagem(msg)**. Caso essa entidade seja outro usuário, a mensagem será enviada para ele. Caso seja um grupo, a mensagem será enviada a todos os usuários associados a tal grupo.

Dessa maneira, para testar seu programa, encaminharemos o comando:

enviarmensagem teste@teste.com entityname "mensagem mensagem mensagem"

o qual será responsável por selecionar o usuário da classe **Comunicador** que tem como e-mail

teste@teste.com. O segundo parâmetro pode ser um e-mail de usuário ou o nome do grupo. Deve-se utilizar esse parâmetro para recuperar, a partir do vetor de entidades do usuário teste@teste.com, o usuário ou o grupo destino. Para isso, deve-se utilizar o método virtual **pegaIdentificadorEntidade()** definido na classe **Entidade** que irá retornar um nome de grupo no caso de **Grupo** e um e-mail no caso de **Usuario**. Portanto, tanto a classe **Usuario** quanto **Grupo** sobrescrevem (polimorfismo de métodos virtuais) tal método a fim de retornar o valor desejado.

Em seguida, o terceiro parâmetro será enviado como mensagem para o usuário ou o grupo. Esse parâmetro pode ser apenas uma palavra, nesse caso não haverá aspas duplas iniciando e terminando tal parâmetro, ou um conjunto de palavras entre aspas duplas.

Caso a mensagem seja corretamente enviada, imprima:

```
printf("Mensagem\n");
printf("De: %s\n", email);
printf("Para: %s\n", entidade);
printf("Texto: %s\n", texto);
```

Caso a mensagem não possa ser enviada, pois o usuário destino ou o grupo destino, ou ainda o próprio usuário que deseja enviar a mensagem não exista, imprima:

```
printf("Problemas ao enviar a mensagem.\n");
```

4) Criar um arquivo de índice para armazenar os relacionamentos de grupos com usuários o qual será denominado **grupos.idx**. Esse arquivo irá conter o nome do grupo e o e-mail do usuário. Observe que esse arquivo contém uma lista repetindo o nome do grupo todas as vezes que houver mais de um usuário nele contido.

#nome do grupo, e-mail do usuário
nomegrupo1, teste@teste.com
nomegrupo1, joao@teste.com
nomegrupo1, paulo@teste.com
nomegrupo2, joao@teste.com
nomegrupo2, carlos@teste.com
nomegrupo2, antonio@teste.com

inserirrelacionamento teste@teste.com joao@teste.com
inserirrelacionamento nomegrupo1 joao@teste.com
inserirrelacionamento joao@teste.com nomegrupo1

Observe que os usuários que serão utilizados nas inserções dos relacionamentos devem constar (existir) na base de dados **usuarios.idx/usuarios.dat**.

No caso em que um grupo A esteja relacionado com outro grupo B e seja solicitada a inserção de um relacionamento invertido, ou seja, a inserção de um relacionamento do grupo B com grupo A, chamado de loop ou reentrância, deve-se imprimir:

```
printf("Relacionamento existente e reentrante.\n");
```

No caso da inserção do um novo relacionamento, essa já exista na base (repetido), deve-se imprimir:

```
printf("Relacionamento existente.\n");
```

No caso de inserção de um novo relacionamento os usuários (emails) que serão utilizados nessa inserção deverão existir na base de dados **usuarios.idx** e **usuarios.dat**. Após a verificação da existência dos usuários na base de dados, e caso **não** seja localizado, deve-se imprimir:

```
printf("Relacionamento inserido com sucesso.\n");
```

No caso da inserção de um relacionamento em que **não** tenha pelo menos um dos seus os usuários cadastrados na base de dados **usuarios.idx** e **usuarios.dat** e seja realizada com sucesso, deve-se imprimir:

```
printf("Erro ao inserir o relacionamento.\n");
```

Observe que não há necessidade de um arquivo de dados para grupos, pois o arquivo de índice possui as informações suficientes do relacionamento do usuário para o processo de troca de mensagens.

5) A remoção de relacionamentos deverá remover o relacionamento de um usuário com um grupo ou um email ou todos seus relacionamentos através do método **removerRelacionamento** (classe **Comunicador**), mas se for removido um grupo ou um usuário deve-se remover todas as ocorrências desse grupo ou usuário existente no arquivo **grupos.idx**.

Se for necessário remover todas as ocorrências de um grupo da base **grupos.idx**, deve-se utilizar o seguinte comando:

removerrelacionamento nomegrupo1

Se for necessário remover todas as ocorrências de um usuário da base **grupos.idx**, deve-se utilizar o seguinte comando:

removerrelacionamento joao@teste.com

Se for necessário remover um único relacionamento de um usuário com outro usuário ou um grupo da base **grupos.idx**, deve-se utilizar o seguinte comando:

removerrelacionamento joao@teste.com carlos@teste
ou
removerrelacionamento joao@teste.com monegrupo2

Caso **não** exista o grupo ou usuário ou relacionamento na base de índices **grupos.idx**, deve-se imprimir:

```
printf("Relacionamento inexistente.\n");
```

Caso exista o grupo ou usuário ou relacionamento específico na base de índices **grupos.idx**, e seja removido com sucesso, deve-se imprimir:

```
printf("Relacionamento removido com sucesso.\n");
```

6) A busca de relacionamentos deverá procurar por um grupo ou um e-mail das seguintes maneiras:

- **buscarRelacionamento** → recebe uma string com o email do Usuário ou de um grupo e busca por um ou mais registros no arquivo binário de índices **grupos.idx**. Em seguida, carrega esses dados, cria e retorna uma lista com os relacionamentos.

Vale lembrar que o arquivo de indexação **grupos.idx** deve conter os emails e os grupos relacionados aos registros de usuários existentes no arquivo de dados **usuarios.dat**, ou seja, deve-se verificar a existência de cada usuário que participará de algum relacionamento.

Sobre este trabalho

Deve-se implementar este trabalho na linguagem C++ criando todas as classes manualmente. Deve-se, também, implementar o mesmo trabalho na linguagem Java. Haverá dois itens no sistema de submissão de trabalhos, um para submeter em linguagem C++ e outro para Java. Ambos são obrigatórios.

Para submeter o trabalho

1) Acesse o site: https://ssp.icmc.usp.br
2) Cadastre-se na opção "Cadastro"
3) Clique na opção "Login" e entre no sistema
4) Clique em "Matricular em disciplina" e selecione "SCC0204 - Programação Orientada a Objetos (Ano: 2013/Semestre: 1- Turma: B)" e matricule-se
5) Clique em "Login" novamente e, em seguida, na opção "Listar disciplinas matriculadas"
6) Clique na opção "Submeter exercícios" da disciplina
7) Escolha o exercício e, em seguida, defina a linguagem de programação como "Zip/Makefile"
8) Selecione seu arquivo zipado contendo o código e o Makefile do programa
9) Submeta seu programa
10) Clique novamente na opção "Login" e, em seguida, na opção "Listar disciplinas matriculadas" e depois em "Resultados" para tomar ciência dos resultados da correção automática

Lembre-se que as notas dependem, ainda, de uma avaliação posterior.

Para criar seu projeto em C++

O projeto zipado deve conter, em seu diretório raíz, um arquivo Makefile. Internamente podem haver outros diretórios. Por exemplo:

```
projetoc++.zip
./Makefile
./meuprojeto/meuprograma.h
./meuprojeto/meuprograma.cpp
```

Dentro do arquivo Makefile deve SEMPRE haver duas targets (all e run). LEMBRE-SE QUE O NOME DO ARQUIVO Makefile DEVE TER A PRIMEIRA LETRA EM MAIÚSCULO! Exemplo de Makefile:

Arquivo Makefile

```
all:
	@g++ -o ./meuprojeto/meuprograma ./meuprojeto/meuprograma.cpp

run:
	@./meuprojeto/meuprograma
```

Observe que há um arroba (@) antes dos comandos contidos no Makefile. Isso é obrigatório, caso contrário a linha de comando será enviada para o sistema de correção automática e ocorrerá erro.

Para criar seu projeto em Java

O projeto zipado deve conter, em seu diretório raíz, um arquivo Makefile. Internamente podem haver outros diretórios. Por exemplo:

projetojava.zip
./Makefile
./meuprojeto/Meuprograma.java

Dentro do arquivo Makefile deve SEMPRE haver duas targets (all e run). LEMBRE-SE QUE O NOME DO ARQUIVO Makefile DEVE TER A PRIMEIRA LETRA EM MAIÚSCULO! Exemplo de Makefile:

Arquivo Makefile

```
all:
	@javac meuprojeto/Meuprograma.java

run:
	@java -cp meuprojeto Meuprograma
```

Observe que há um arroba (@) antes dos comandos contidos no Makefile. Isso é obrigatório, caso contrário a linha de comando será enviada para o sistema de correção automática e ocorrerá erro.

A opção -cp (ou classpath) define o diretório em que se encontram as classes compiladas (arquivos extensão .class).

Saída esperada para o programa

Algumas opções serão encaminhadas para o programa. Essas opções são:

1) *autenticarusuario* → ao receber essa opção, o seu programa deve capturar os dados referentes a autenticação do usuário na ordem:

- Email do usuário

- Senha do usuário

Através do método de busca verificar se usuário existe na base de usuário e na sequencia verificar se a senha está correta. Pode-se obter as seguintes mensagens conforme o caso:

```
printf("Usuario nao encontrado.\n");  \\ sem acentuação
                                        ou
printf("Usuario encontrado.\n");  \\ sem acentuação
                                         e
printf("Senha correta.\n");
                                        ou
printf("Senha incorreta.\n");
```

2) *enviarmensagem* → ao receber essa opção, seu programa deve capturar os dados referentes ao usuário na ordem:

- Email do usuário
- Email do usuário alvo ou nome do grupo
- Mensagem à ser enviada

em seguida, invocar o método sendMessage de um objeto da classe Comunicador. Ao final imprima:

```
printf("Mensagem.\n");
printf("De: %s.\n", email);
printf("Para: %s.\n", entidade);
printf("Texto: %s.\n", texto);

                                        ou

printf("Problemas ao enviar a mensagem.\n");
```

3) *inserirusuario* → ao receber essa opção, seu programa deve capturar os dados referentes ao usuário na ordem:

- Email do usuário
- Senha do usuário
- Nome completo
- Apelido ou nickname para o Comunicador Instantâneo
- Cidade
- Estado
- País
- Data de nascimento

em seguida, criar um objeto da classe **Usuario** e depois invocar o método **inserirUsuario** de um objeto da classe **Comunicador**. Ao final imprima:

```
printf("Usuario inserido com sucesso.\n"); \\ sem acentuação
```

ou

printf(“Falha ao inserir usuario.\n”); \\ sem acentuação

caso um usuário com mesmo email já esteja cadastrado no sistema.

4) *inserirrelacionamento* → ao receber essa opção, seu programa deve substituir o primeiro usuário pelo o **identificadorEntidade**, (email) do usuário autenticado ou pelo nome do grupo, à ser relacionado e, em seguida, solicitar o email ou nome do grupo que será relacionado no segundo usuário. Busque por ambos usuários utilizando **buscarBinUsuario** do objeto da classe **Comunicador**, o qual utiliza o arquivo **grupos.idx**. Em seguida, caso não exista, adicione o segundo usuário na lista de contatos do primeiro e vice-versa. Se um dos usuários não existir exiba a seguinte mensagem na tela:

Caso de inserção de um novo relacionamento do grupo B com o grupo A e já exista no arquivo de índice **grupos.idx** o relacionamento do grupo A com o grupo B, deve-se imprimir:

printf(“Relacionamento existente e reentrante.\n”);

Caso exista o relacionamento no arquivo de índice, deve-se imprimir:

printf(“Relacionamento existente.\n”);

Caso o relacionamento seja cadastrado com sucesso, deve-se imprimir:

printf(“Relacionamento inserido com sucesso.\n”);

Caso o identificador do usuário (email) não existir na base de usuários (**usuários.idx**/**usuários.dat**) no momento da inserção do relacionamento, deve-se imprimir:

printf(“Erro ao inserir o relacionamento.\n”);

5) *removerrelacionamento* → ao receber essa opção, seu programa deve ler o email do usuário ou grupo e chamar o método **removerRelacionamento** do objeto da classe Comunicador. Esse método é responsável por removê-lo do sistema. Ainda dentro do método **removerRelacionamento**, faça:

Caso **não** exista o grupo ou usuário ou relacionamento, imprimia:

printf(“Relacionamento inexistente.\n”);

Caso exista o grupo ou usuário ou relacionamento específico e seja removido com sucesso, imprima:

printf(“Relacionamento removido com sucesso.\n”);

6) *buscarrelacionamento* → ao receber essa opção, seu programa deve ler o email do usuário ou nome do grupo e chamar o método **buscarRelacionamento** do objeto da classe **Comunicador**. Receba o

lista retornado, o qual será da classe **Usuario**, e chame o método imprimir e, em seguida, **imprimirRelacionamento**.

Exemplo de entrada para seu programa

inserirusuario
joao@teste.com
senha1
"Joao da Silva"
jsilva
"Sao Carlos"
SP
Brasil
12/02/1973
inserirusuario
carlos@teste.com
senha2
"Carlos da Silva"
csilva
Recife
PE
Brasil
01/07/1962
autenticarusuario
joao@teste.com
senha1
inserirrelacionamento
joao@teste.com
carlos@teste.com
inserirrelacionamento
amigos
carlos@teste.com
buscarrelacionamento
joao@teste.com
buscarelacionamento
amigos
removerrelacionamento
amigos
removerrelacionamento
joao@teste.com
quit

Exemplo de saída esperada para a entrada anterior:

Usuario criado com sucesso
Usuario criado com sucesso
Usuario encontrado

Senha correta
Relacionamento criado com sucesso
Relacionamento criado com sucesso
amigos
carlos@teste.com
1
joao@teste.com jsilva
carlos@teste.com
1
Relacionamento removido com sucesso
Relacionamento removido com sucesso

Outro exemplo de entrada para seu programa

autenticarusuario
joao@teste.com
senha1
inserirrelacionamento
joao@teste.com
carlos@teste.com
inserirrelacionamento
amigos
carlos@teste.com
buscarrelacionamento
joao@teste.com
buscabinrelacionamento
amigos
enviarmensagem
joao@teste.com
amigos
"Teste de envio para amigos"
enviarmensagem
joao@teste.com
carlos@teste.com
"Teste de envio para carlos"
removerrelacionamento
amigos
removerrelacionamento
joao@teste.com
quit

Outro exemplo de saída esperada para a entrada anterior:

Usuario encontrado
Senha correta
Relacionamento criado com sucesso
Relacionamento criado com sucesso
amigos
carlos@teste.com

1
joao@teste.com jsilva
carlos@teste.com
1
Mensagem
De: joao@teste.com
Para: amigos
Texto: Teste de envio para amigos
Mensagem
De: joao@teste.com
Para: carlos@teste.com
Texto: Teste de envio para carlos
Relacionamento removido com sucesso
Relacionamento removido com sucesso