MC302
Primeiro semestre de 2018

Laboratório 8

Professora: Esther Colombini (esther@ic.unicamp.br)
PEDs: Nathana Facion (nathanafacion@gmail.com), Rafael Tomazela (sohakes@gmail.com), Luis Fernando Antonioli (luisfernandoantonioli@gmail.com))
PAD: Anderson Cotrim (ander.cotrim@gmail.com)

---

1 Descrição Geral

Neste laboratório, continuaremos com o desenvolvimento incremental do sistema de caronas on-line desenvolvido nos laboratórios anteriores.

2 Objetivo

Este laboratório tem como objetivo aprimorar o entendimento sobre os conceitos de interface, tratamento de exceção e manipulação de arquivos.

3 Atividade

Primeiramente, precisamos terminar a implementação de alguns métodos para que o sistema fique coeso. Depois disso, criaremos métodos que permitem que o estado de parte do sistema seja salvo e carregado de arquivos.

3.1 Definições

As atividades que devem ser realizadas estão descritas nessa seção.

3.1.1 Finalizando o sistema

Começaremos com algumas implementações necessárias para um sistema coeso:

1. Anteriormente, criamos os métodos `atribuirNotaCaronante(...)` e `atribuirNotaCaroneiro(...)`, onde um caronante ou um caroneiro podiam, respectivamente, dar uma nota para a carona. A ideia é que essas avaliações sejam usadas para mudar a avaliação do perfil dos outros integrantes da carona. Implemente uma forma de calcular a avaliacação de um perfil através das avaliações recebidas. Você pode calcular da forma que achar melhor. Uma ideia é percorrer todas as caronas que um perfil participa, tanto como caronante quanto como caroneiro, e então usar a média das avaliações dadas por outros integrantes como a avaliação de perfil. Se desejar, você pode remover o atributo `avaliacao` do `Perfil`, e calcular sempre que o `getAvaliacao()` for chamado, ou ainda atualizar o valor do atributo sempre que algum outro integrante da carona faz uma avaliação.

2. Na classe `Grupo` temos o método abstrato `adicionarMembro`, e na classe `Usuario` temos o método `adicionarGrupo`. Por agora, provavelmente não há diferença entre o método `adicionarMembro` nas classes `GrupoPublico` e `GrupoPrivado`, mas a ideia é que somente o dono de um grupo possa adicionar um usuário em um grupo privado, mas qualquer usuário pode se adicionar em um grupo público. Para fazer essa modificação:

   (a) Insira o método `checarPresencaUsuario` na classe Grupo que deve retornar se um determinado usuário faz parte daquele grupo;

   (b) Crie os métodos `criarGrupoPublico` e `criarGrupoPrivado` em `Usuario` para que o usuário possa criar grupos em que ele é o dono;

   (c) Altere o método `adicionarGrupo` da classe Usuario para que este receba um GrupoPublico. O objetivo da mudança é que um usuário só possa se adicionar a um grupo público;

   (d) Insira o método `adicionarUsuarioAUmGrupo` que deve adicionar um usuário a um grupo privado. Note que essa inserção só deve acontecer se o usuário que está tentando inserir o outro usuário for dono do grupo privado. O método retorna *true* em caso de sucesso e `false` caso contrário;

   (e) Garanta que o método `removerGrupo` só remova o usuário do grupo caso ele não seja o dono (no caso de ser um grupo privado).

3. Os grupos permitem que os caronantes possam limitar os caroneiros que podem pedir aquela carona. Para isso, os caronantes podem adicionar grupos públicos em caronas públicas, e grupos privados em caronas privadas, dado que o caronante participe dos grupos que queira adicionar. Para implementar essas restrições, faça as seguintes modificações:

   (a) Garanta que o método `adicionarGrupo` das classes `CaronaPrivada` e `CaronaPublica` só adicionem um grupo caso o Caronante daquela carona pertença ao Grupo.

   (b) Modifique o método `adicionarCaroneiro` de `CaronaPrivada` e `CaronaPublica` para que verifiquem se o usuário que está pedindo carona pertence ao grupo (usando `checarPresencaUsuario`, por exemplo). Caso uma carona pública não tenha grupos associados, considere que qualquer usuário pode ser adicionado (não faça isso para caronas privadas).

3.1.2 Novas funcionalidades

Todas as partes dos laboratórios anteriores que faltavam ser implementadas devem ter sido terminadas. As funcionalidades estão descritas na seção 5, junto com as novas funcionalidades que deverão ser implementadas. A implementação dessas novas funcionalidades está descrita abaixo:

1. Faça com que a classe Perfil implemente a interface `Comparable` definida pela linguagem Java (`https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html`) utilizando, para comparação, a avaliação do Perfil do usuário. Esta interface permite que possamos saber, dado dois Perfis, qual possui melhor avaliação. Isso será usado posteriomente por métodos que aceitam objetos que implementam `Comparable` para ordenar listas.

2. Crie uma interface `Salvavel` para salvar os objetos das classes em arquivos. Essa interface deve conter o método `salvarParaArquivo` (que recebe os parâmetros que preferir), que será implementado por cada classe citada a seguir para salvar os atributos do objeto em arquivos. Lembre-se de tratar as exceções necessárias para a abertura/escrita do arquivo. A escolha do tipo de arquivo a ser salvo deve ser realizada por cada aluno. Os slides da disciplina contém explicações detalhadas de cada um dos tipos de arquivo e das formas de manipulá-los.

(a) Usuario

(b) Perfil

(c) GrupoUsuario

(d) Grupo

(e) GrupoPublico (o ArrayList de caronas não deve ser salvo)

(f) GrupoPrivado (o ArrayList de caronas não deve ser salvo)

(g) Caroneiro (o ArrayList de caronas não deve ser salvo)

(h) Caronante (o ArrayList de caronas não deve ser salvo)

Você pode criar esse método da forma que achar melhor. Primeiramente, é necessário estruturar o arquivo de forma que seja fácil ler e interpretar a informação. Um jeito possível é primeiramente indicar no arquivo o número de usuários, seguido de cada usuário, com seu perfil, caroneiro e caronante, fazer o mesmo com os grupos, e então com o relacionamento entre grupos e usuários, recriando então tais relacionamentos. Você também pode usar o método `useDelimiter` do `Scanner` (`https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html`) para separar atributos e objetos com outro delimitador que não seja espaço, como '*', '|', '<', '>'.

Note que não faz sentido salvar um `Grupo`, e sim um `GrupoPublico` ou um `GrupoPrivado`, então uma ideia é `Grupo` implementar a interface definindo os métodos como abstratos. Para salvar os relacionamentos entre `Usuario` e `Grupo`, você pode guardar o identificador dos objetos para depois reconstruir a relação.

3. Crie uma exceção de nome `SistemaCaronaExcecao`. Modifique o método `adicionarUsuarioAUmGrupo` de `Usuario` para que ao invés de retornar um boolean dizendo se o usuário foi adicionado com sucesso, o método seja void e gere uma exceção caso não seja possível adicionar.

4. Cada um dos métodos citados abaixo tem uma condição que deve ocorrer, caso a condição não ocorra devemos chamar uma exceção. Essa exceção deve ser uma classe SistemaCaronaExcecao, para um melhor entendimento de como fazer isso veja este link `https://goo.gl/TvZdzG`. Então, altere os seguintes métodos para que o sistema se torne consistente e completo, realizando as exceções necessárias:

   (a) Altere o método `adicionarGrupo` da classe Usuario para que este receba um GrupoPublico. O objetivo da mudança é que um usuário só possa se adicionar a um grupo público.

   (b) Insira o método `adicionarUsuarioAUmGrupo` que deve adicionar um usuário a um grupo privado. Note que essa inserção só deve acontecer se o usuário que está tentando inserir o outro usuário for dono do grupo privado. O método retorna *true* em caso de sucesso e *false* caso contrário

   (c) Garanta que os métodos `removerGrupo` só removem o usuário do grupo caso ele pertença ao grupo.

   (d) Insira o método `checarPresencaUsuario` na classe Grupo que deve retornar se um determinado usuário faz parte daquele grupo.

3.1.3 Testes

No main, siga os seguintes passos:

1. Crie 5 usuários (u0, u1, ..., u4) com seus respectivos perfis e faça com que eles sejam tanto Caroneiros quanto Caronantes. Você deve usar um array ou ArrayList para guardar os usuários criados;
2. Crie 1 grupo público (gpu);
3. Crie 1 grupo privado (EC017) e faça u0 ser dono dele;
4. Faça u0 inserir u1 e u2 no grupo EC017 (imprima o estado do grupo EC017 antes e depois dessa inserção);
5. Faça u2 tentar inserir u4 no grupo EC017 (imprima o estado do grupo EC017 antes e depois dessa tentativa de inserção), capture a exceção;
6. Faça u2, u3, e u4 se adicionarem ao grupo público gpu;
7. Faça u3 oferecer uma carona Publica e em sequência anunciar ela (inserí-la) no grupo publico gpu;
8. Faça u0 tentar se retirar do grupo gpu e imprima a saída;
9. Faça u2 oferecer uma carona Privada e em sequência anunciar ela (inserí-la) no grupo privado EC017;
10. Faça u4 oferecer uma carona privada e em sequência tentar anunciar ela (inserí-la) no grupo privado EC017;
11. Faça os usuários u0 e u1 pedirem a carona privada criada pelo usuário u2. Faça com que o u3 tente pedir a mesma carona. Imprima o retorno dos métodos;
12. Adicione os perfis dos usuários u0, u1 e u2 em uma lista, se já não fez;
13. Faça com que u0, u1 e u2 avaliem a carona privada criada por u2, e imprima as avaliações dos perfis, na ordem colocada na lista (escolha valores das avaliações de forma que a lista não esteja ordenada por avaliação);
14. Ordene a lista de acordo com as avaliações
15. Imprima todas as instâncias criadas anteriormente para mostrar o estado do sistema;
16. Salve todos os dados no(s) arquivo(s) apropriado(s).

3.2 Observações

- Não peça, em nenhum momento, dados pela entrada padrão. Construa os objetos com valores inseridos diretamente no código;
- Envie os arquivos *.java*. Jamais envie somente os *.class*;
- Verifique se o código não tem nenhum *warning* antes de enviar;
- Deixe o código organizado e bem indentado. Use comentários quando achar necessário;
- Não coloque acentos no nome das classes. É possível que haja problemas na codificação durante o processo de compressão dos arquivos, e isso vai causar problemas para o compilador;
- Qualquer dúvida, sinta-se livre para pedir ajuda através de e-mail, nos laboratórios, ou na monitoria.

4 Questões

Adicione acima do Main as respostas das questões abaixo. É recomendado que antes de responder seja lido o material de sala de aula. Caso você realize o teste para alguma situação abaixo no código, o mesmo deve ficar comentado, pois não faz parte do sistema de Carona proposto.

Parte I: Sobre interfaces e classes abstratas, responda as seguintes questões:

1. É possível uma interface herdar de uma classe abstrata? É possível uma classe abstrata implementar uma interface?
2. Quais as principais diferenças entre classes abstratas e interfaces?
3. O que acontece se uma classe implementar 2 interfaces diferentes, mas que tem um método com o mesmo nome e assinatura?
4. No caso geral, não é possível implementar métodos em interfaces. Mas a partir do Java 8, é possível um de dois modificadores aos métodos de uma interface para que seja possível implementá-los. Quais são esses modificadores?

Parte II: Sobre arquivos e exceção, responda as seguintes questões:

1. Qual a diferença na leitura através de um objeto `BufferedInputStream` para um `InputStream` (incluindo suas subclasses)?
2. A classe DataOutputStream tem alguns métodos para escrita, com as assinaturas descritas abaixo. Esses métodos dão *throw* em exceções do tipo `IOException`. Por que esse tipo de exceção é jogada? Em que contextos faz sentido jogar uma exceção do tipo `IOException`?

   ```
   public final void writeInt(int i) throws IOException;
   public final void writeFloat(float f) throws IOException;
   public final void writeByte(int b) throws IOException;
   public final void writeChar(int i) throws IOException;
   ...;
   ```

3. Explique a utilidade da interface `Serializable`, como ela pode ser usada com arquivos e cite duas exceções que métodos dessa interface jogam.

5 Requisitos do Sistema que devem estar implementados

1. Deve ser possível criar novos Usuários, Perfis, Caroneiros e Caronantes;
2. Deve ser possível associar Usuários com Perfis, e Perfis com Caroneiros e Caronantes;
3. Um Usuario deve poder criar um Grupo Publico ou um Grupo Privado;
4. Um Usuario dono de um grupo privado deve poder adicionar outros Usuarios em tal grupo.
5. Um Usuario deve poder se adicionar em um grupo publico
6. Um Caronante deve poder criar uma Carona Publica ou uma Carona Privada;
7. Um Caronante deve poder adicionar Grupos em Caronas;
8. Um Caroneiro deve poder pedir uma Carona existente, dado que este Caroneiro pertença a um Grupo associado à Carona, ou que seja uma Carona Publica sem Grupos;

9. Um Caroneiro ou um Caronante deve poder avaliar uma Carona ao qual está associado;
10. Deve ser possível calcular a nota/avaliação de um Perfil de Usuario, baseado nas avaliações dadas por outros Usuarios nas Caronas em que participou;
11. Deve ser possível ordenar os Perfis dos Usuarios de acordo com suas avaliações;
12. Deve ser possível salvar os dados das partes do sistema indicadas em arquivo.

6 Questão bônus

Como atividade bônus referente a este laboratório, espera-se que seja construída uma interface Carregavel que, para as mesmas classes indicadas em Salvavel, carregue os dados dos arquivos apropriados nos objetos da memória.

O bônus será aplicado sobre a média de laboratório e poderá valer até 0,5 ponto.

7 Submissão

Para submeter a atividade utilize o Moodle (`https://www.ggte.unicamp.br/ea`). Salve os arquivos dessa atividade em um arquivo comprimido no formato .tar.gz ou .zip e nomeie-o **Lab8-000000.zip** trocando '000000' pelo seu número de RA. Submeta o arquivo na seção correspondente para esse laboratório no moodle da disciplina MC302.

Datas de entrega

- Dia **28** de Maio, Turma **ABCD** até às 23:55

Informação importante

- A atividade correspondente ao laboratório 8 contará, para nota, como 2 unidades de laboratório.

8 Digrama UML

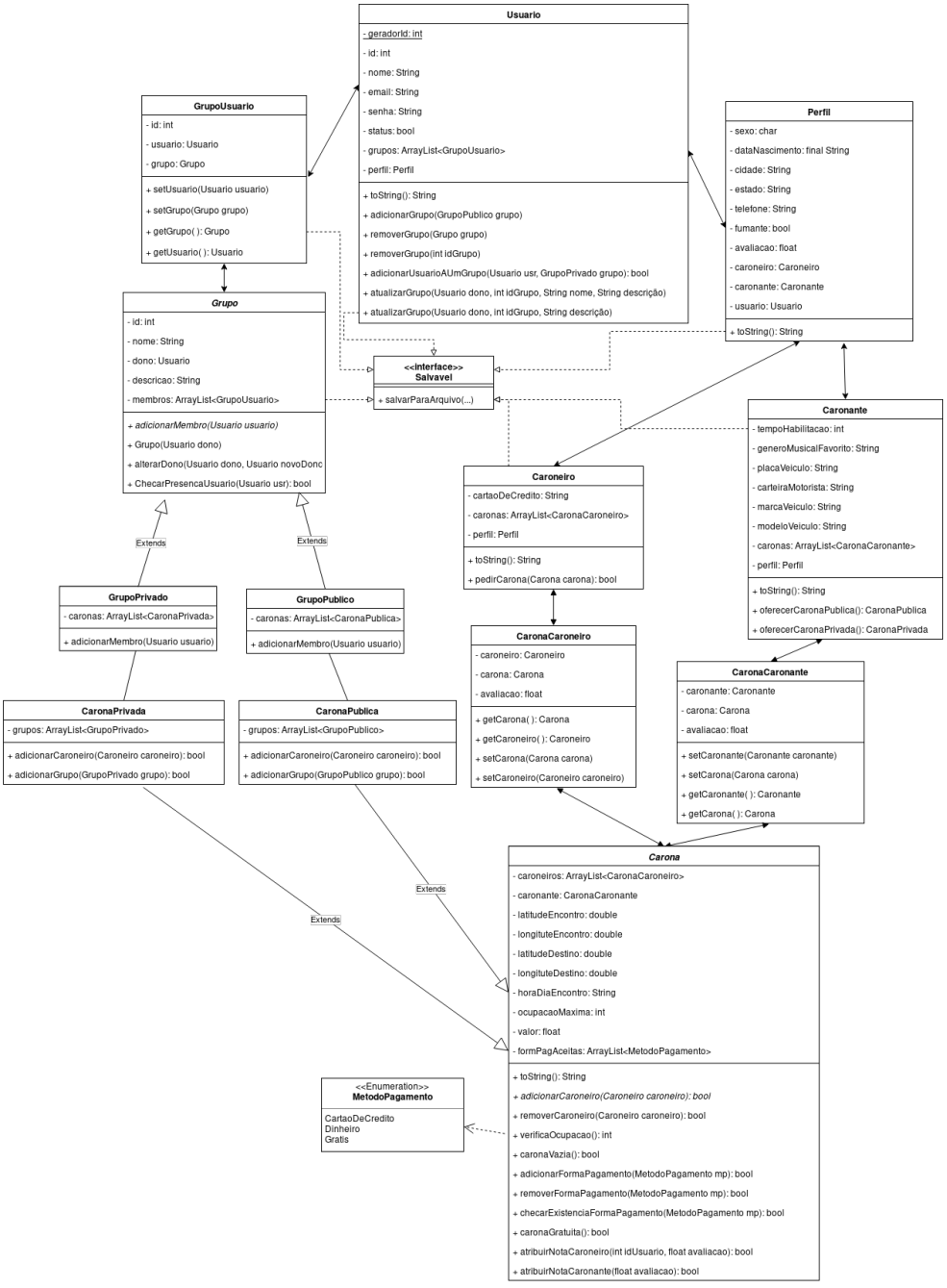

Figura 1: Diagrama UML do sistema