MC302
Primeiro semestre de 2018

Laboratório 6

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, vamos continuar com a construção do sistema de carona on-line, reaproveitando o que foi feito nos dois laboratórios anteriores.

2 Objetivo

Este laboratório tem como objetivo tratar de relacionamentos entre classes. Isso já foi feito anteriormente, mas agora vamos tratar de relacionamentos utilizando classes associativas.

3 Atividade

Nos laboratórios anteriores, há um relacionamento direto entre a Carona, com o *Caroneiro* e o *Caronante*. Mas uma carona pode ter vários caroneiros, e cada caroneiro pode estar em várias caronas. Até agora isso foi suficiente, mas se quiséssemos adicionar um atributo para o relacionamento, o que poderia ser feito? Um possível modo de resolver o problema é usando uma classe associativa que guarda informações do relacionamento, nesse caso a classe *CaronaCaroneiro*, que será criada. Faremos o mesmo com o relacionamento entre *Carona* e *Caronante*, criando a classe *CaronaCaronante*.

3.1 Definições

Com a mudança da classe *Carona*, *Caroneiro* e *Caronante*, além da criação das classes *CaronaCaroneiro* e *CaronaCaronante*, mais modificações em *Perfil*, temos o diagrama UML apresentado na Figura 1 (escondemos as partes não necessárias como os grupos ou a herança da carona).

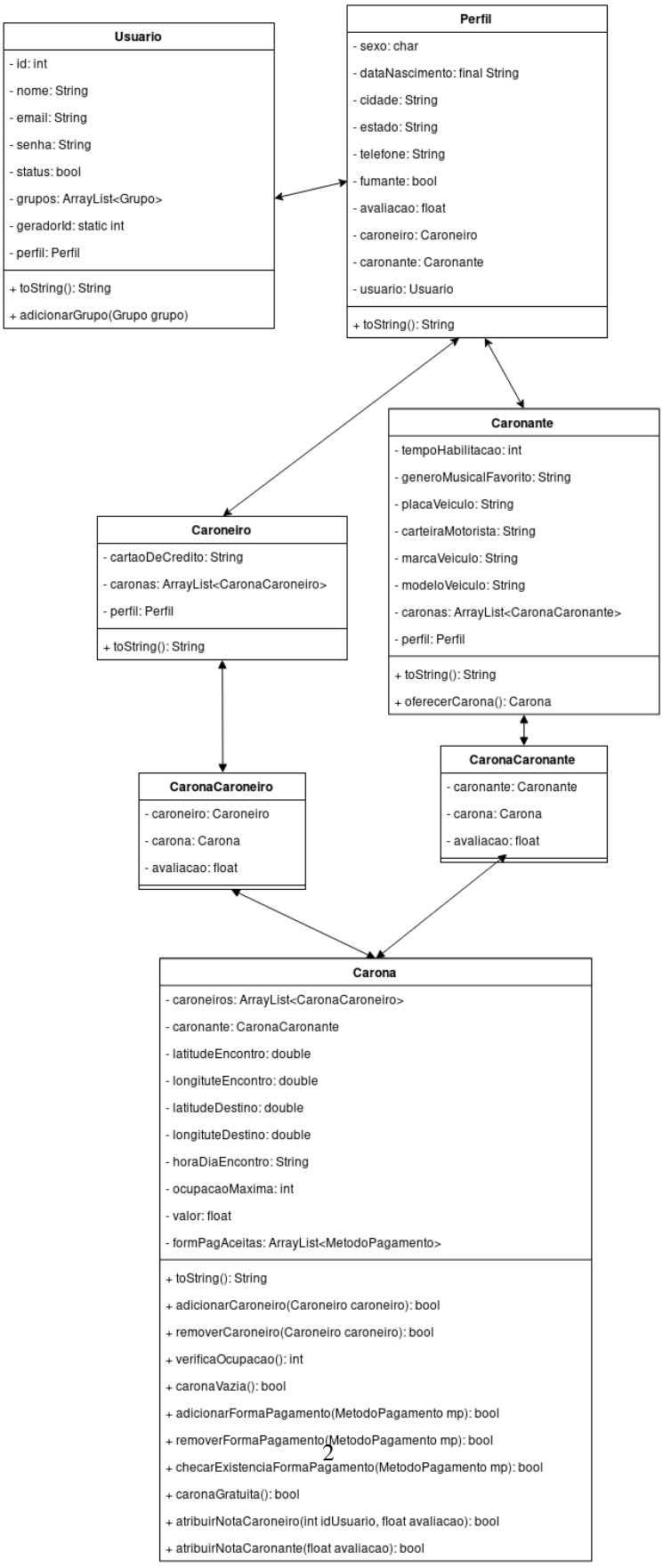

Figura 1: Diagrama UML de parte do sistema

Como tarefa:

1. Implemente a nova classes conforme apresentado no diagrama UML 1. Os *getters*, *setters* e construtores foram omitidos. Perceba a adição do atributo *usuario* na classe *Perfil*. Isso possivelmente vai causar algum problema pelo relacionamento bi-direcional, talvez no construtor ou no método *toString*. Corrija os problemas como achar melhor;

2. O método *atribuirNotaCaroneiro* da classe *Carona* deve receber um inteiro, que indica o *id* de um usuário e um *float* que indica a avaliação da carona. Então o método deve percorrer pela lista de objetos *CaronaCaroneiro*, verificando se o *id* de algum deles é o passado (através do relacionamento *Carona->CaronaCaroneiro->Caroneiro->Perfil->Usuario->id*), e então modificar o valor da avaliação correta. O método deve retornar verdadeiro caso o usuário tenha sido encontrado, e falso caso contrário. O método *atribuirNotaCaronante* tem a mesma função, mas como só há um caronante não é necessário passar um *id*, e deve retornar falso somente se, por algum motivo, não houver objeto atribuído do tipo *CaronaCaroneiro*;

3. Modifique o método *adicionarCaroneiro* da classe *Carona* para criar um objeto da classe associativa. Além disso, modifique o método *oferecerCarona* da classe *Caroneiro* para que receba os parâmetros da carona, criando os objetos das classes *Carona* e *CaronaCaronante*, e fazendo os relacionamentos necessários;

4. No main, crie quatro usuários com diferentes IDs, cada um associado com um diferente perfil. Associe o perfil de um dos usuários com um objeto do tipo Caronante, e os outros três usuários com objetos do tipo Caroneiro. Crie uma carona a partir do método *oferecerCarona* e adicione os três caronantes. Agora atribua as notas que cada caroneiro e o caronante deram para a carona. Além disso, tente atribuir a nota de um caroneiro para um *id* inexistente. Imprima o resultado de cada uma das chamadas de atribuição de nota, e verifique se o retorno é o esperado;

5. Imprima as informações de cada usuário (é suficiente imprimir só os atributos da classe *Usuário*, sem cascatear os relacionamentos) e da carona, incluindo a nota que cada usuário deu para a carona (identifique cada usuário pelo seu ID, e se ele era caroneiro ou caronante). A forma de imprimir as informações vai depender de como você tratou o método *toString()*.

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*.
- 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.

4 Questões

Sobre a atividade realizada, responda como comentário no início do código da classe que contém o método *main*.

- Há diversos relacionamentos bi-direcionais em todo o sistema, como é o caso entre Carona e CaronaCaroneiro, por exemplo. Quais são as vantagens e desvantagens de se utilizar relacionamentos bi-direcionais?

- O relacionamento entre a classe Carona e Caronante não é de muitos para muitos. Considerando esse fato, foi realmente necessário criar a classe *CaronaCaronante*? Qual seria uma alternativa se não quiséssemos criar a classe associativa, mas ainda assim guardar informação do relacionamento (nesse caso, a nota)? Quais as vantagens e desvantagens dessa abordagem alternativa, comparando com a atual?

- Em um sistema real, o funcionamento dos métodos criados (caso nenhuma modificação adicional tenha sido feita) são suficientes para garantir consistência do sistema? É possível que haja algum problema com um relacionamento entre as classes caso não seja feito alguma verificação adicional? Por exemplo, poderíamos chegar em um estado onde um caronante se relaciona com uma carona, mas tal carona não tem referência a esse caronante? Como poderíamos impedir que algo assim ocorra? Caso no seu sistema não seja possível criar esse tipo de inconsistência, então "Não é possível"é uma resposta válida (mas iremos verificar se é verdade).

5 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 **Lab6-000000.[tar.gz | 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 **23** de Abril, Turma **ABCD** até às 23:55