MC302
Primeiro semestre de 2018

Laboratório 4

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 laboratórios anteriores.

2 Objetivo

O objetivo desta atividade consiste na refatoração do código construído até o momento e a familiarização com o conceito de *Herança*.

3 Atividade

3.1 Refatoração

Nesta parte da atividade realizaremos a refatoração do código construído nos laboratórios passados para atender as demandas das novas fases do projeto. Para isso, as seguintes classes deverão ser ajsutadas:

3.1.1 Usuário

Descrição dos atributos:

- *id* é um identificador único de cada usuário.
- *nome* é o nome completo do usuário.
- *emai*l é o login do usuário.
- *senha* é usada para acessar o sistema de caronas.
- *status* é o responsável por dizer a disponibilidade do usuário, caso seja verdadeiro o usuário está disponível, caso contrário não.
- *grupos* são comunidades que agrupam usuários com interesses similares em caronas.
- *perfil* é onde teremos informações adicionais do usuário, entre elas: se é fumante, a cidade,o estado.
- geradorId é o responsável por gerar o id do usuário. Como o id tem que ser único, essa variável é estática (não se esqueça de incrementar seu valor no construtor e atribuir ao id).

Descrição dos métodos:

- *toString()* onde devemos ter os atributos concatenados e retornados como uma String (como nos labs anteriores)
- *adicionarGrupo()* onde devemos adicionar os grupos que o usuário participa.
- *getXX() e setXX()* também devem ser implementados de acordo com a necessidade dos atributos.

Altere a classe usuário de acordo com as descrições acima e o UML apresentado na Figura 6:

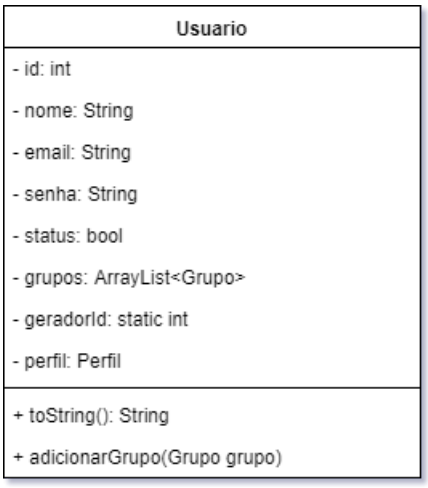

Figura 1: Diagrama UML da classe Usuário

3.1.2 Perfil

Descrição dos atributos:

- *sexo* é o sexo do usuário.
- *dataNascimento* é a data de nascimento do usuário.
- *cidade* é a cidade em que o usuário habita.
- *estado* é a estado em que o usuário habita.
- *telefone* é o telefone para com contato com usuário.
- *fumante* serve para informar se o usuário fuma(true) ou não(false).
- *avaliacao* é a avaliação média que ele tem como caroneiro, caronante ou ambos.
- *caroneiro* é o perfil dele associado com caroneiro, com os dados para ser caroneiro.
- *caronante* é o perfil dele associado com caronante, com os dados para ser caronante.

Descrição dos métodos:

- *toString()* onde devemos ter os atributos concatenados e retornados como uma string (como nos labs passados)

- *get() e set()* também devem ser implementado de acordo com a necessidade dos atributos.

Altere a classe perfil de acordo com as descrições anteriores e o diagrama apresentado na Figura 2.

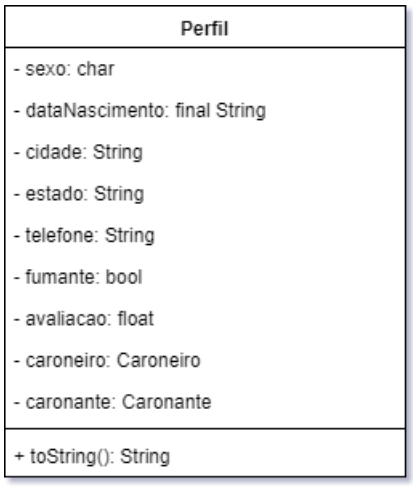

Figura 2: Diagrama UML da classe Perfil

3.1.3 Caroneiro

Descrição dos atributos:

- *cartaoDeCredito* representa o número do cartão de crédito que poderá ser usado para pagar a carona, caso esta seja cobrada.
- *caronas* por enquanto não será necessário adicionar este atributo. Nos laboratórios adiante faremos uso dele.
- *perfil* é preciso adicionar o atributo perfil no caronante pra que seja possível então encontrar seu usuário e seu grupo.

Descrição dos métodos:

- *toString()* onde devemos ter os atributos concatenados e retornados como uma String (como nos labs passados)
- *get() e set()* também devem ser implementados de acordo com a necessidade dos atributos.

Altere a classe Caroneiro de acordo com a descrição anterior e o modelo da Figura 3.

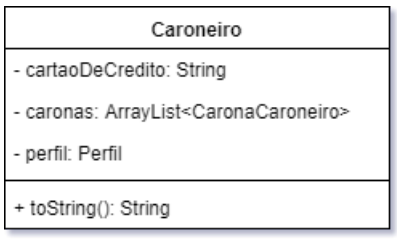

Figura 3: Diagrama UML da classe Caroneiro

3.1.4 Caronante

Descrição dos atributos:

- *tempoHabilititicao* define o tempo (em anos) de experiência do caronante.
- *generoMusicalFavorito* é o gênero musical favorito do caronante.
- *placaVeiculo* é a placa do veículo do caronante.
- *carteiraMotorista* é o número de registro da carteira de motorista
- *marcaVeiculo* é a marca do veículo do caronante.
- *modeloVeiculo* é o modelo do veículo do motorista.
- *caronas* este atributo será utilizado apenas em laboratórios fututos. Não adicione no momento.
- *perfil* é preciso adicionar o atributo perfil no caronante pra que seja possível então encontrar seu usuário e seu grupo.

Descrição dos métodos:

- *oferecerCarona()* cria uma nova carona com as especificações de acordo com o interesse do caronante.
- *toString()* onde devemos ter os atributos concatenados e retornados como uma string (como nos labs passados)
- *get() e set()* também devem ser implementados de acordo com a necessidade dos atributos.

Altere a classe Caronante de acordo com a descrição anterior e o diagrama UML da Figura 4.

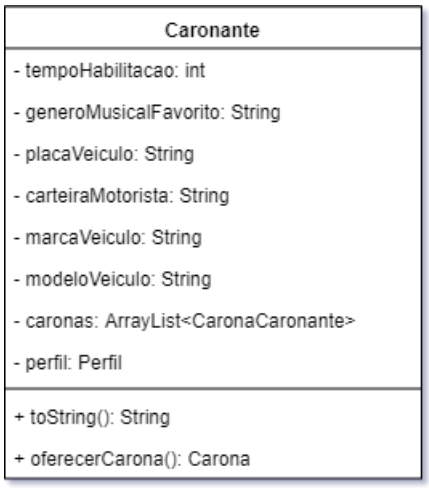

Figura 4: Diagrama UML da classe Caronante

3.2 Herança

Nesta parte do laboratório, adicionaremos 3 novas classes ao projeto: **Grupo**, **GrupoPublico** e **GrupoPrivado**. Por meio dessas classes vamos demonstrar o funcionamento de herança em Orientação a Objetos.

- *Grupo* é a classe mãe das quais as demais classes herdarão.
- *GrupoPublico e GrupoPrivado* são classes filhas e portanto são mais especializadas. Estas são classes de um tipo de Grupo. Cada uma possui objetivos distintos já que *GrupoPublico* será utilizada para um grupo visível a todos os usuários e *GrupoPrivado* representará um grupo visível apenas para quem for convidado ou autorizado a participar do mesmo.

3.2.1 Grupo

Descrição dos atributos:

- *id* é um identificador único de cada grupo.
- *nome* é o nome do grupo.
- *descricao* é a descrição sobre o grupo.
- *geradorId* é o responsável por gerar o id do grupo. Como o id tem que ser único, essa variável é estática.(não se esqueça de incrementar seu valor no construtor e atribuir ao id)
- *membros* é uma lista que contém todos os usuários que fazem parte deste grupo.

Descrição dos métodos:

- *adicionarMembro()* usado para adicionar novos usuários a este grupo.
- *toString()* onde devemos ter os atributos concatenados e retornados como uma String (como nos labs passados).
- *get() e set()* também devem ser implementado de acordo com a necessidade dos atributos.

3.2.2 GrupoPublico e GrupoPrivado

Descrição dos atributos:

- *caronas* contém as caronas associadas a este grupo. Por enquanto, não deve ser adicionado pois será construída em laboratórios posteriores.

Descrição dos métodos:

- *adicionarMembro()* usado para adicionar novos usuários a este grupo.
- *toString()* onde devemos ter os atributos concatenados e retornados como uma String.
- *get() e set()* também devem ser implementados de acordo com a necessidade dos atributos.

A Figura 5 apresenta o digrama correspondente as classes da hierarquia de grupos.

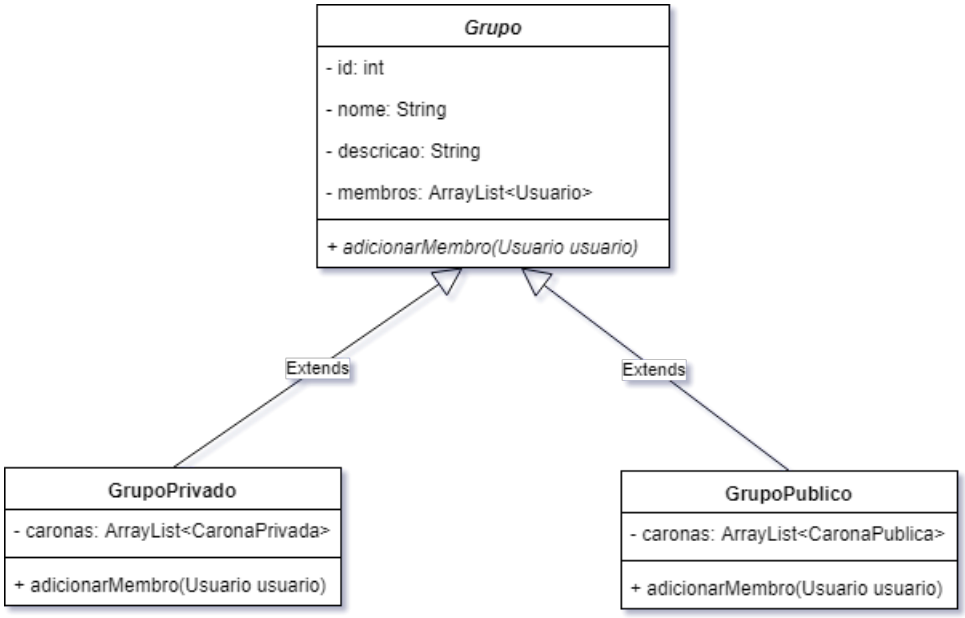

Figura 5: Diagrama UML da Herança. Como podemos ver os extends representam a herança que existe de Grupo com GrupoPrivado e GrupoPublico.

4 Atividades

4.1 Construtores

Apesar dos métodos construtores não terem sido mencionados, faz parte do trabalho de cada aluno definir aqueles métodos necessários a implementação da sua solução para o porblema.

4.2 Implementação

Para cada classe alterada e/ou criada neste laboratório:

1. Crie um objeto no main.
2. Invoque os métodos implementados para entender o funcionamento dos mesmo.
3. Verifique o uso das variáveis estáticas ao longo da execução das chamadas anteriores (Não se esqueça de usar o nome da classe para acessar a variável).

Finalmente, apresente na tela todos os usuários alocados em cada grupo criado.

5 Questões

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

- Qual o principal benefício da herança?
- Adicione final na classe *Grupo* , o que aconteceu com o código? Por que isso aconteceu? Em vez de *Grupo* ser final e se definirmos GrupoPublico como final?
- Crie uma variável estática em *Grupo* e *GrupoPublico* com o nome testStatic do tipo inteiro. No main, instancie 3 objetos: **a** da classe *Grupo*, **b** e **c** da classe *GrupoPublico*. Faça a=b e b=c.
  - Se você imprimir a variável testStatic qual das duas classes foi chamada para **a**, **b** e **c**?
  - Altere as variáveis testStatic das duas classes, removendo o static. Não esqueça de adicionar o get neste caso, pois você terá que acessar o atributo por meio de instância de objeto. O resultado é o mesmo? O que mudou?

6 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 **Lab4-000000.[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 **2/4/2018** Turma **ABCD** até às 23:55

7 Dica

Como vocês já entenderam o funcionamento de alguns métodos e tem uma maior familiaridade com Java, existe uma forma de implementar alguns conceitos mais rapidamente.

Para fazer uma implementação de forma mais rápida para sets(), gets(), construtores e toString() podemos usar funcionalidades do eclipse. Para isso:

1. Clique com o botão direito do mouse no código.
2. Selecione Source e a opção do método que quer implementar.

Imagem para ilustrar os passos:

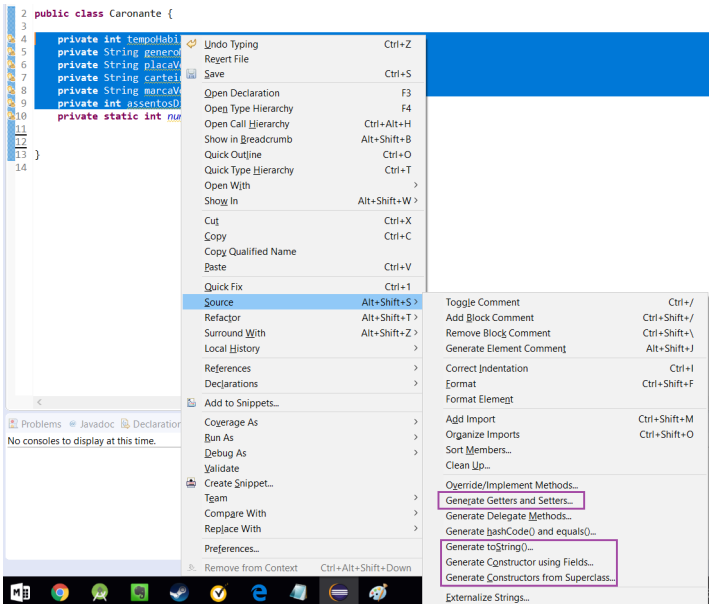

Figura 6: Criando métodos de forma automática