Engenharia de Telecomunicações
POO29004 – Programação Orientada a Objetos

03/06/2019

Lista 7: Polimorfismo, Enum e Tipos genéricos

Professor: Emerson Ribeiro de Mello http://docente.ifsc.edu.br/mello/poo

Nota:

- *Antes de iniciar, sincronize seu repositório local com o seu repositório remoto "Listas de exercícios" do Github. É possível que o professor tenha feito um commit por lá após a entrega da lista de exercício anterior;*
- *Crie um diretório com o nome* `lista-07` *na raiz do repositório e coloque as soluções dessa lista dentro desse diretório;*
- *Edite o arquivo* `Readme.md` *que está na raiz do repositório e adicione um novo item com o nome* `lista-07` *e este deverá ter um link o qual deverá apontar para o subdiretório com o nome* `lista-07`*.*

Para cada um dos exercícios apresentados a seguir, crie um subdiretório e dentro desse crie um projeto Java+gradle com o IntelliJ. O nomes dos subdiretórios deverão ser: `exercicio-01`, `exercicio-02` e `exercicio-03`.

1 Veículos

Faça a implementação das classes e interfaces apresentadas no diagrama da Figura 1.

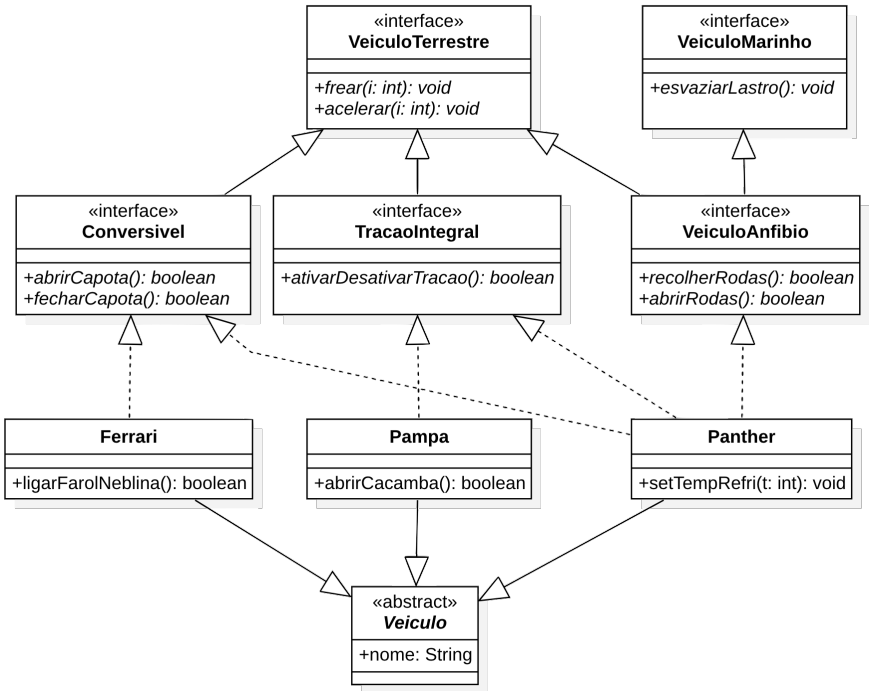

Figura 1: Diagrama de classes

Atenção: Se necessário, modifique a hierarquia de classes para melhorar o projeto. Faça testes de unidade para todos os métodos de todas as classes. A implementação deverá respeitar as seguintes regras:

- Na implementação de cada método não devem ser colocadas instruções de entrada e saída para interação com o usuário (e.g. `Scanner` e `System.out`). Tais instruções devem estar presentes somente no método `main` da classe `Principal`.
  - Na classe `Principal`, faça um `ArrayList` e neste adicione pelo menos uma instância de cada uma das classes concretas do digrama da Figura 1, já respeitando a modificação que venha fazer.
  - Faça pelo menos uma interação (invoque algum método) com cada instância presente no `ArrayList`.
- Para as ações abaixo, retorne para o usuário (imprima uma mensagem na tela), se conseguiu ou não realizar a ação.
  - Só é possível abrir a caçamba se o carro estiver parado
  - Só é possível ativar ou desativar a tração integral se o veículo estiver parado
  - Só é possível ativar a tração integral se o veículo anfíbio estiver com as rodas abertas
  - Só é possível abrir ou fechar a capota se o carro estiver parado, com exceção da Ferrari, pois essa permite fazer isso se a velocidade estiver abaixo dos 20km/h
  - Ao recolher as rodas é necessário esvaziar o lastro

2 Baralho francês

- Desenvolva um conjunto de classes que possibilite representar um baralho de cartas. Para essa modelagem é obrigatório o uso de `Enum`.
- Um baralho francês possui 52 cartas, sendo 13 cartas para cada um dos 4 naipes (paus, ouros, copas e espadas)
- Cartas são: Ás, 2, 3, 4, 5, 6, 7, 8, 9, 10, Valete (J), Dama (Q) e Rei (K)

Figura 2: Baralho frânces

3 Classe que permita comparações

A interface `Comparable<T>`[1] possui o método `int compareTo(<T>)` que tem por objetivo permitir a comparação entre o objeto em questão com aquele que fora passado como parâmetro. Esse método é usado para fazer ordenação de elementos em uma Coleção, por exemplo, quando se desejar ordenar elementos de uma *ArrayList*. O método deve retornar um número inteiro negativo, zero ou número inteiro positivo se o objeto atual for menor, igual ou maior que o objeto que fora passado como parâmetro, respectivamente.

A classe `String` implementa a interface `Comparable<String>`[2] e classe `LocalDate`, que serve para manipular data e hora, implementa a interface `Comparable<LocalDate>`. Sendo assim, ambas as classes provêem implementação para o método `int compareTo(<T>)`. Abaixo é apresentado um trecho de Java mostrando o uso do método `compareTo`.

```
1  String s1 = "Ama";
2  String s2 = "Ana";
3  String s3 = "Ana";
4
5  System.out.println(s1.compareTo(s2)); // imprime -1
6  System.out.println(s2.compareTo(s1)); // imprime 1
7  System.out.println(s2.compareTo(s3)); // imprime 0
8
9  LocalDate d1 = LocalDate.of(1990, 01, 01);
10 LocalDate d2 = LocalDate.of(1980, 01, 01);
11
12 System.out.println(d1.compareTo(d2)); // imprime 10
```

Para esse projeto, copie na íntegra as classes Pessoa, Telefone, Agenda e Email que foram desenvolvidas na **Lista de exercícios 05**. Se você não fez a lista 05, então terá que prover as implementações para essas classes.

- Modique a classe `Pessoa` para implementar a interface `Comparable<Pessoa>`. A lógica do método `compareTo` permitirá ordenar objetos dessa classe dentro de uma coleção de forma ascendente, respeitando a seguinte ordem:
  - nome, sobrenome e data de nascimento
- Faça uma classe Principal, instancie uma coleção (e.g. `ArrayList`) e adicione alguns objetos da classe pessoa. Imprima o conteúdo coleção, ordene e imprima novamente. A ordenação de uma coleção pode ser feita da seguinte forma com Java8:

```
13 List<Pessoa> pessoas = new ArrayList<>();
14
15 pessoas.add(new Pessoa ("Joao", "Silva" , "1990-01-01"));
16 pessoas.add(new Pessoa ("Ana",  "Paula" , "1989-01-01"));
17 pessoas.add(new Pessoa ("Joao", "Santos", "1991-01-01"));
18 pessoas.add(new Pessoa ("Joao", "Silva" , "1980-01-01"));
19
20 System.out.println(pessoas); // imprimirá exatamente igual a ordem que foram adicionados
21
22 Collections.sort(pessoas); // ou ainda,  pessoas.sort(Comparator.naturalOrder());
23
24 System.out.println(pessoas);
25 // Imprimirá algo assim:
26 // Ana,  Paula,  1989-01-01
27 // Joao, Santos, 1991-01-01
28 // Joao, Silva,  1980-01-01
29 // Joao, Silva,  1990-01-01
```

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
[2] https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#compareTo-java.lang.String-

Atenção:

- ***Para receber nota 10:*** *Os commits para essa entrega deverão ser encaminhados (push) para o Github Classroom até o dia 14/06/2020;*
- ***Para receber nota 5:*** *Os commits deverão ser encaminhados (push) para o Github Classroom até o dia 15/06/2020;*
- *Qualquer entrega após o dia **15/06/2020** não será contabilizada e receberá nota 0.*