PO24CP - Programação Orientada a Objetos
Profa. Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Trabalho Prático - Parte 1 - Simulador de Corrida de Veículos

Data de entrega: 28/04/2023, via moodle.
O trabalho poderá ser individual ou em grupo de até 2 estudantes.

Um simulador de corrida de veículos irá controlar veículos através de seu centro de comandos. Os veículos estarão competindo no estilo corrida.

Cada veículo criado possuirá uma identificação única (que deverá ser um número inteiro, gerado automaticamente) e uma quantidade de **4** rodas (cujos pneus estarão calibrados ou não de acordo com um sorteio[1]). Além disso, os veículos terão uma quantidade inicial de combustível, 2,5 litros, seu valor de venda e se o IPVA (Imposto sobre a Propriedade de Veículos Automotores) do veículo estará pago ou não.[2]

Os veículos podem ser abastecidos e consomem combustível à medida que se deslocam. Eles apenas se movimentam se há combustível suficiente para tal, se os pneus das rodas estiverem todos calibrados e se o IPVA estiver pago. Assume-se que para mover um "*bloco*"[3] de espaço, o veículo gaste 0,55 listros de combustível. Portanto, um veículo **não** deve se mover caso:

- não possua a quantidade de combustível suficiente,
- se um ou mais pneus não estiverem calibrados e,
- se o IPVA não estiver pago.

Os veículos devem ser desenhados em modo texto.[4] Cada veículo se move sempre na horizontal da esquerda para direita, deslocando-se cinco (5) "*blocos*" de espaço (unidade de movimento) a cada vez que se movimenta.

Com base no detalhamento anterior, faça:

I. Descreva o diagrama UML das classes do simulador tomando como modelo o esboço apresentado na Figura 1 (gerar o arquivo pdf do diagrama).

II. Com base do diagrama UML elaborado acima, desenvolva um aplicativo Java com um **menu interativo** que permita a utilização de um objeto da classe Simulador de corrida de veículos com no máximo 20 veículos:

(1) Incluir veículo

[1]Por exemplo, para cada roda, sorteia-se um valor booleano *true* ou *false*, se for sorteado *true*, calibra-se o iésimo pneu, caso contrário, não.

[2]Também neste caso, sorteia-se um valor booleano, se for *true*, o IPVA estará pago, caso contrário, não.

[3]Entende-se por bloco de espaço uma coluna (de n linhas) de espaço, que devem ser considerados ao desenhar o percurso do veículo na pista.

[4]Deve ser utilizado, para cada veículo, o desenho em código *ascii* fornecido no arquivo `desenhaCarro.java` disponível no moodle.

[Gerar um *id* (inteiro) automático e único para cada veículo e assumir que cada pneu está vazio ou não de acordo com um sorteio. O IPVA estará pago ou não, também de acordo com um sorteio.]

(2) Remover veículo

[deve-se informar o *id* do veículo]

(3) Abastecer veículo

[deve-se informar o *id* do veículo e a quantidade de combustível em litros]

(4) Movimentar um veículo

(5) Movimentar todos os veículos.

(6) Imprimir todos os dados de um veículo

(7) Imprimir todos os dados de todos os veículos

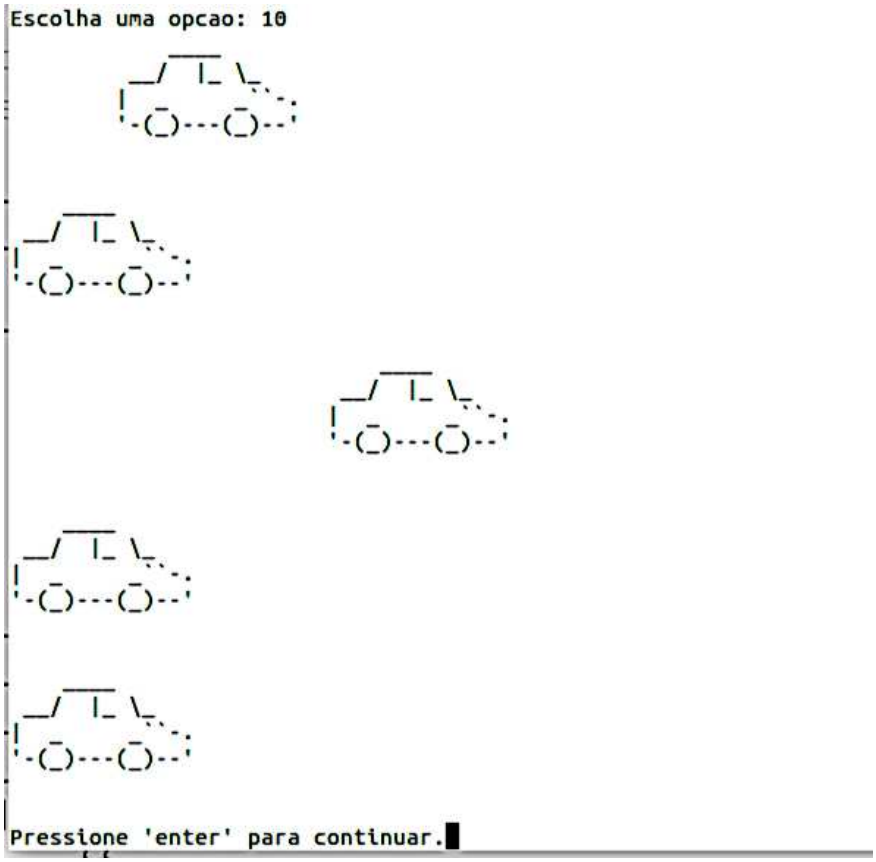
(8) Esvaziar/calibrar um pneu específico,

[Solicitar o *id* do veículo, a ação (esvaziar/encher) e qual pneu será esvaziado/calibrado (primeiro, segundo,...), caso não seja dada uma entrada correta, repetir a leitura.]

(9) Esvaziar/calibrar todos os pneus de um veículo específico

(10) Imprimir pista de corrida

[imprime na ordem em que estão no *array* os veículos com seus respectivos *"blocos"* de espaços percorridos. Para desenhar cada veículo, utilize o desenho *ascii* contido no arquivo `desenhaCarro.java`. Por exemplo, a pista de corrida abaixo descreve 4 veículos no array, onde o segundo contem o maior deslocamento até o momento.]



(11) Gravar veículos em arquivo

(12) Ler veículos do arquivo

(13) Sair da aplicação

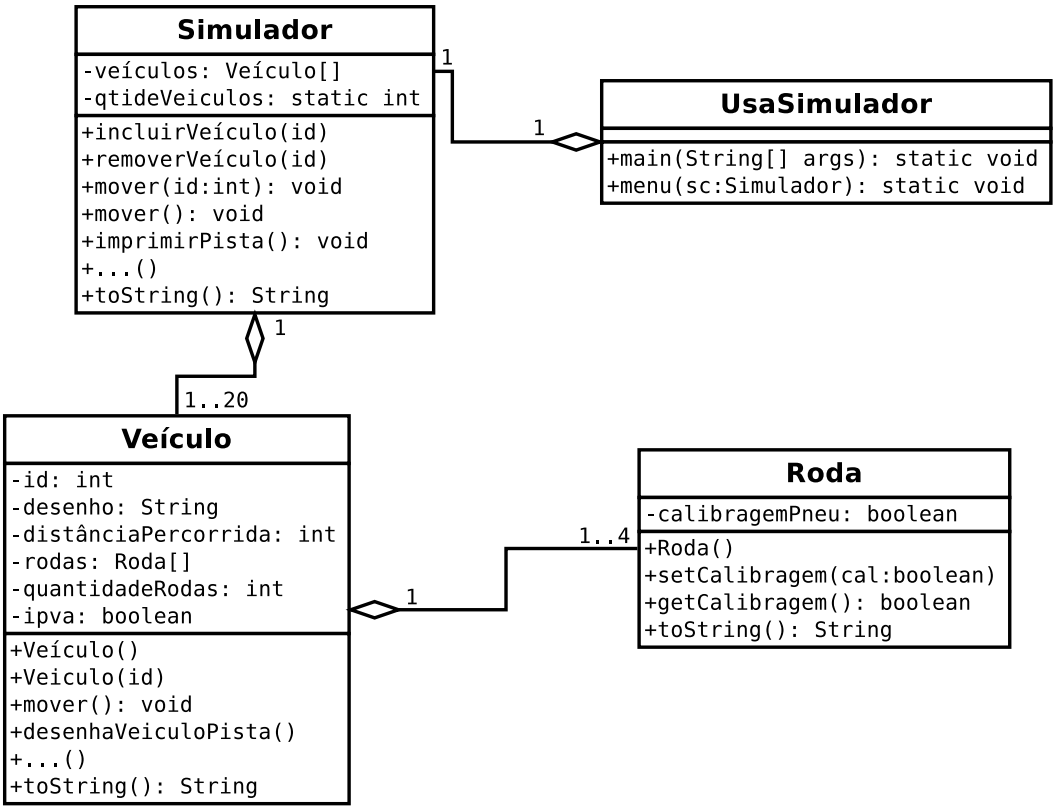


Figura 1: Esboço do diagrama UML a ser seguido.

Avaliação:

A atividade será avaliado em função da:

- Correção (o aplicativo cumpre com as exigências?);
- Documentação (o aplicativo está devidamente comentado?);
- Paradigma orientado a objetos (o aplicativo está seguindo os princípios da programação OO: -encapsulamento, -associação de classes?)
- Modularidade (o aplicativo está bem estruturado onde necessário, com métodos (funções) parametrizados?);
- Robustez (o aplicativo trava em tempo de execução?).

Detalhamento de itens a serem avaliados:

Item	Atendeu?
Respeitar o princípio do encapsulamento de dados	
Usar modificadores de acesso adequados (`private` e `public`)	
Criar getters e setters que forem necessários	
Criar métodos construtores parametrizados	
Fazer sobrecarga de pelo menos um método (qualquer um)	
Fazer uso da palavra reservada `this`	
Ter pelo menos um atributo `static`	
Criar relacionamento entre classes (Agregação ou Composição)	
Não apresentar erro em tempo de execução	
Apresentar o diagrama UML	