Engenharia de Software 2 – Trabalho Semipresencial 02

Professor: Leonardo F. B. S. Carvalho

Nome(s):

Informações

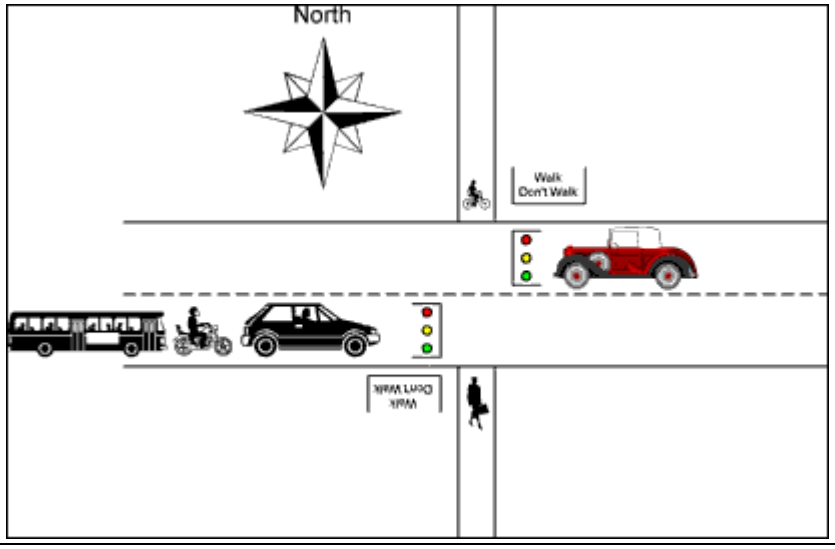
- A atividade consiste de exercícios práticos relacionados a padrões de Projeto de Software.
- A atividade deve ser feita e entregue **preferencialmente em dupla** até as **23h55m** do sábado dia **17/11/18**

As questões do trabalho cobrem apenas as classes e objetos que tratam da simulação. Não serão feitas interfaces gráficas. A abordagem a ser seguida é orientada a objetos e define classes para os carros, luzes de sinalização e pista de espera. Padrões de projeto serão usados para estruturar as relações entre as classes e objetos da simulação. É recomendado que antes de iniciar as questões sejam listadas as classes que compõem o projeto. Elas serão refatoradas à medida que você realiza as questões.

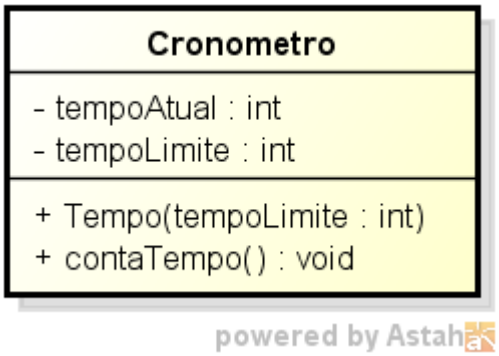
No seu projeto, para cada questão, crie um pacote para armazenar as classes criadas para ela. Caso a proposição de uma questão exija alterar uma classe criada para uma questão anterior, ambas as versões da classe devem ser mantidas. Com a nova versão da classe sendo colocada no pacote da questão atual.

Os diagramas de classe para cada proposição mostram a estrutura básica (não completa) das classes para o sistema. Altere as estruturas conforme a sua necessidade para a implementação dos requisitos ou padrões indicados.

Ao final do exercício você terá um programa que simula o tráfico em um cruzamento, com dois semáforos para liberar e interromper o fluxo de carros em sentidos opostos de uma pista cortada por uma faixa de pedestres.



1. **Padrão Singleton**: crie a classe Cronometro que será usada por diferentes classes para nosso sistema. A classe não é um relógio real, mas simula um relógio. Deve haver apenas um cronometro para o nosso sistema, para garantir que o tempo marcado seja o mesmo para todas as nossas classes.



a. O cronometro deve ter um atributo tempoAtual de tipo int inicializado com zero.

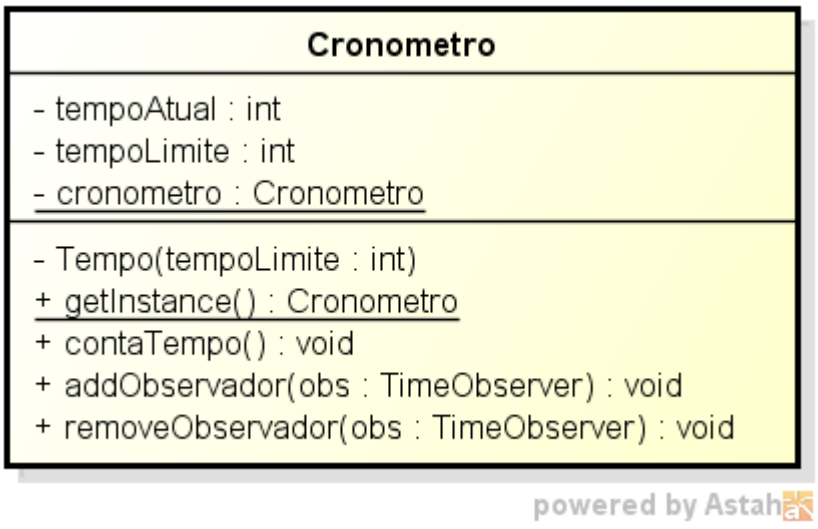
b. O atributo tempoLimite deve ter um método get, mas não um set.

c. A classe deve ter um atributo tempoLimite de tipo int, que armazena o total de segundos para o cronometro do nosso relógio. Inicie este parâmetro no construtor da classe

d. O método run() incrementa o valor de tempoAtual em 1 unidade

2. **Padrão Observer**: implemente também o padrão Observer na classe Cronometro da questão anterior. Nosso cronometro será um objeto observável (Subject) que conta com uma lista de seus observadores

a. A classe deve contar as unidades de tempo e, a cada unidade, deve notificar os observadores registrados usando seu método de notificação

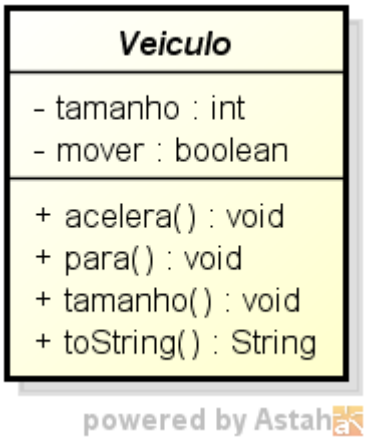


b. O método contaTempo() incrementa o valor de tempoAtual e notifica cada observador registrado sobre a mudança.

c. Escreva a interface ObservadorTempo que possui um único método de atualização, atualizaTempo(), como no diagrama. Esta será o observador abstrato implementado pelos observadores concretos.

d. Identifique ao longo das próximas proposições quais serão os observadores concretos.

3. **Abstract Factory**: Escreva uma classe abstrata Veículo com a estrutura do diagrama abaixo.



a. Escreva três classes concretas para Veículo: Carro, Onibus e Caminhao de acordo com os parâmetros:

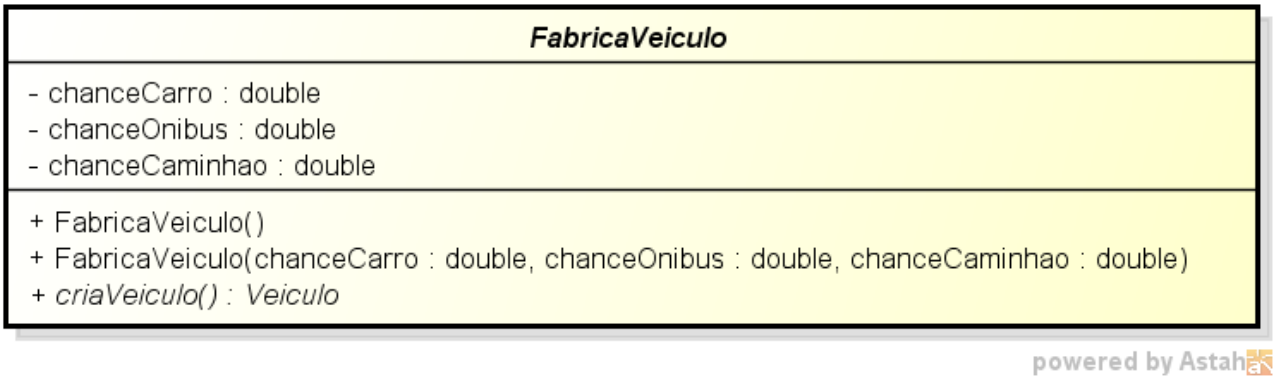
i. Carro: 6 metros de comprimento

ii. Onibus: 18 metros de comprimento

iii. Caminhao: 14 metros de comprimento

b. Para as 3 classes é preciso apenas definir o construtor que define os valores indicados

4. **Abstract Factory**: Escreva uma classe abstrata FabricaVeiculo que cria de forma aleatória um carro de passeio, ônibus ou caminhão



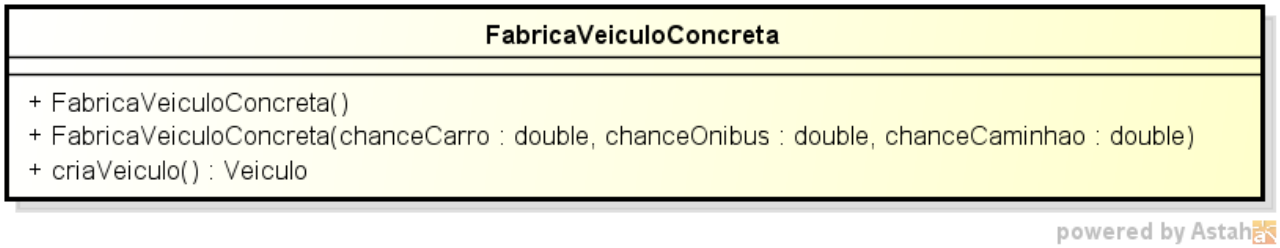
a. A fábrica, por meio do seu método *criaVeiculo*, retorna instâncias concretas de veículos de acordo com as diferentes probabilidades definidas nos atributos da classe.

b. O construtor sem argumentos define valores padrão para os atributos da classe, enquanto que o construtor com argumentos passa para os atributos os valores recebidos como parâmetros

i. Os valores padrão são: 0,8 carro; 0,12 para ônibus; 0,8 para caminhões

ii. Implemente uma restrição para que a soma dos atributos não ultrapasse 1,0 (100%). Da mesma forma, nenhum dos atributos pode ser negativo

5. **Abstract Factory:** Para a classe abstrata da questão anterior, escreva a subclasse concreta abaixo.

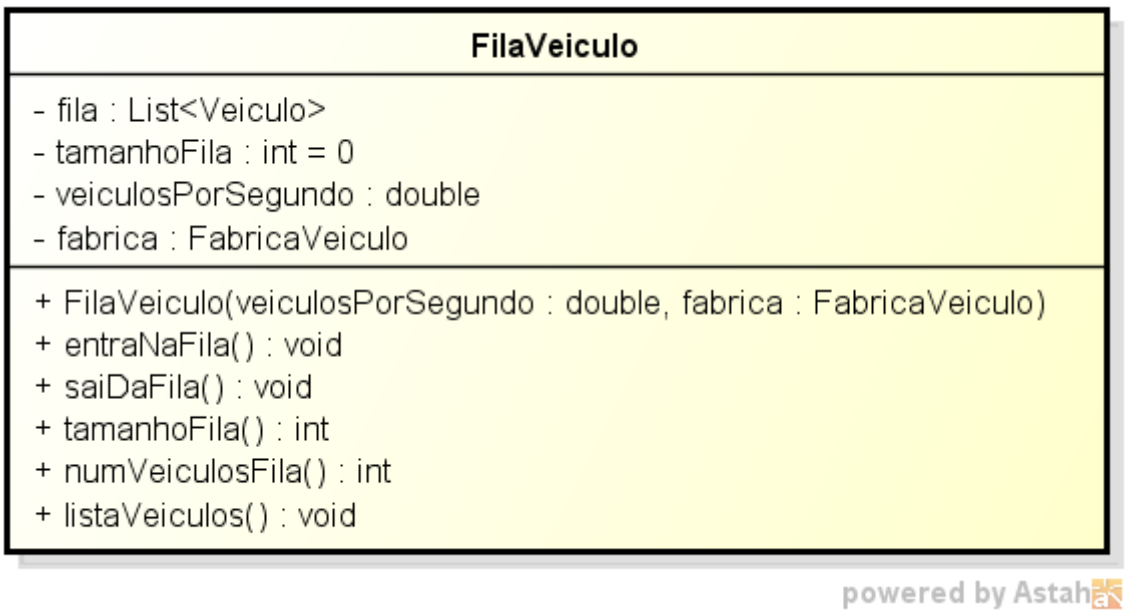


a. Implemente o método *criaVeiculo* para que retorne as instâncias concretas de veículos de acordo com as diferentes probabilidades definidas nos atributos da classe.

6. **Abstract Factory**: escreva uma classe que demonstra veículos em uma fila. A estrutura básica da classe é dada como segue.

a. A classe representa uma fila, na qual um veículo pode entrar a qualquer momento

b. O objeto theQueue armazena a lista de veículos da fila.



a. A chance de um veículo (qualquer que seja ele) entrar na fila a cada segundo é dada pelo atributo

vehiclesPerSecond. Um número real entre zero e um que determina a chance de um veículo aparecer na fila em um dado momento.

b. Caso seja sorteado que um veículo entrará na fila, o tipo de veículo a entrar será decidido pelo atributo theFactory, objeto da classe de fábrica que irá gerar um veículo para a fila dadas as probabilidades indicadas na questão 4.

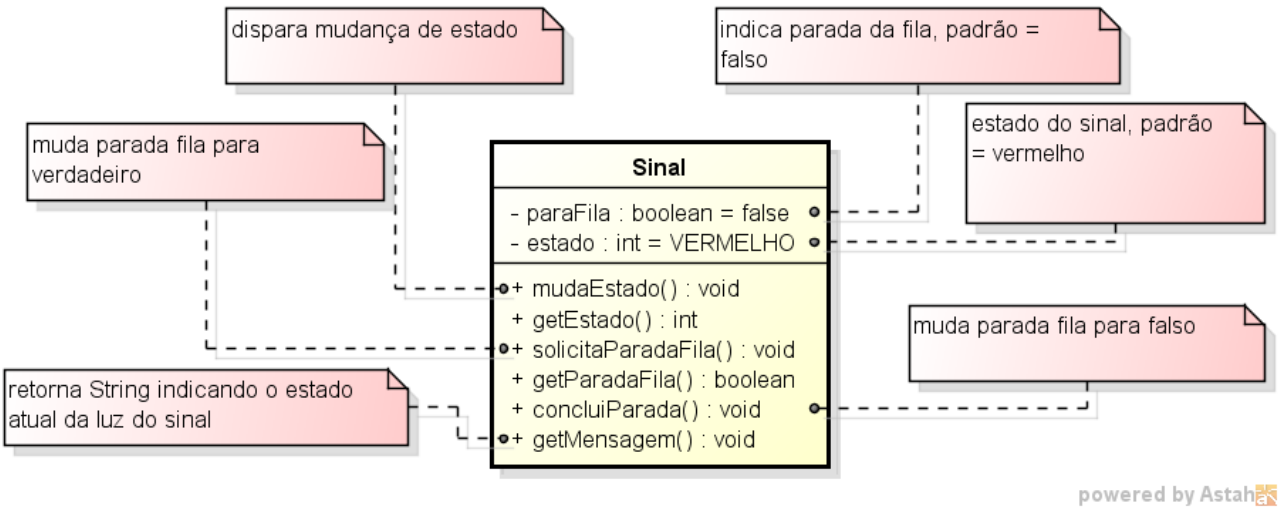
c. Demais métodos da classe seguem como:

- tamanhoFila(): tamanho total da fila em metros (soma do tamanho de todos os veículos)
- numVeiculosFila(): retorna o número de veículos na fila.
- saiDaFila(): chamado tantas vezes quanto possível quando a luz do semáforo está verde, cada chamada remove um veículo da fila.
- entraNaFila(): chamado a cada segundo, dependendo do valor de vehiclesPerSecond e do valor aleatório sorteado a cada chamada, resulta em um veículo entrar ou não na fila
- listaVeiculo(): imprime a lista de veículos da fila, a forma de armazenar a lista de veículos (Vector, LinkedList, ArrayList, etc.) é escolhida pela equipe.

7. Escreva classes para os semáforos. Eles serão o sinal para que possamos liberar veículos da fila. Escreva a classe Signal que satisfaça as proposições abaixo.

a. Define 2 estados para um sinal a partir de constantes nomeadas VERMELHO, AMARELO e VERDE.

b. Um sinal está sempre em um desses 3 estados. Os estados sempre mudam na ordem VERMELHO → VERDE → AMARELO → VERMELHO.



a. O atributo haltLine indica quando alguém solicitou passagem pela pista (true).

b. Sempre que o estado do sinal for VERDE, paradaFila deve ser definido automaticamente como

falso.

8. **Classe principal do programa (método main)**: o programa principal deverá assegurar que a luz está verde por um período mínimo de tempo. Não há tempo máximo para que a luz esteja verde, mas, quando uma parada for solicitada, ela deve respeitar o período mínimo antes de mudar para vermelho.
    - **a.** A simulação não requer entrada direta do usuário. A duração da simulação e o percentual de carros, ônibus e caminhões será informada diretamente no código (hardcoded).
    - **b.** A simulação irá executar por 600 segundos (10 minutos) que na prática corresponde a um loop de 600 interações para o programa (um for ou um while).
    - **c.** Para cada interação há uma chance de 10% de se solicitar a parada do sinal e de 50% para um veículo entrar na fila.
    - **d.** O tempo do sinal fechado é de 20 segundos.
    - **e.** Uma vez aberto, o sinal não pode ser fechado novamente por, pelo menos, 30 segundos.
    - **f.** Caso o sinal esteja aberto por pelo menos 30 segundos e um pedido de parada foi gerado, ocorre a transição de estado do sinal (item 7).
    - **g.** Armazene em um contador o total de solicitações de parada até que a parada ocorra de verdade. Zere o contador enquanto o sinal estiver vermelho
    - **h.** A cada segundo (iteração) são gerados como saída no console o tempo atual, o tamanho da fila de veículos (em metros), o número de veículos na fila e o total de solicitações de parada.