MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TRIÂNGULO MINEIRO
CAMPUS UBERLÂNDIA CENTRO

TECNOLOGIA EM SISTEMAS PARA INTERNET
Turma: **3º PERÍODO**
Unidade Curricular: **PROGRAMAÇÃO ORIENTADA A OBJETOS**
Professor: **WILL ROGER PEREIRA**

LISTA 1-9

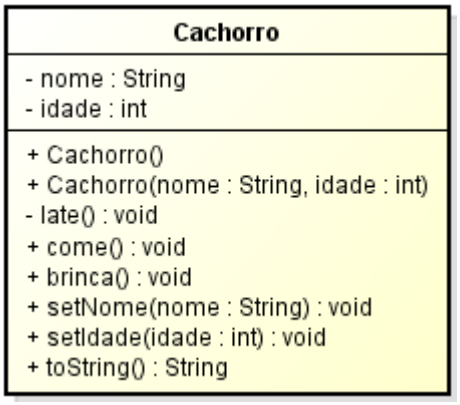
Obs: Para todos os exercícios, crie pelo menos 2 (dois) objetos, inicialize os atributos utilizando os construtores (exceto quando especificado no diagrama), e execute todos os métodos públicos para demonstrar suas funcionalidades.

Obs2: As especificações e/ou restrições para os valores dos atributos sempre se encontrarão neles!!! Caso este valor esteja fora das especificações dentro de um método, sempre mostre uma mensagem de erro. No caso dos construtores, caso aconteça algum problema com os atributos, atribua valores padrões.

Obs3: O levantamento de restrições também é de sua responsabilidade. Portanto, sempre que encontrar alguma irregularidade na execução de um método, informe este erro.

Obs4: LEIA, NA ÍNTEGRA, A DESCRIÇÃO DE TODOS OS ATRIBUTOS E MÉTODOS.

1ª Questão

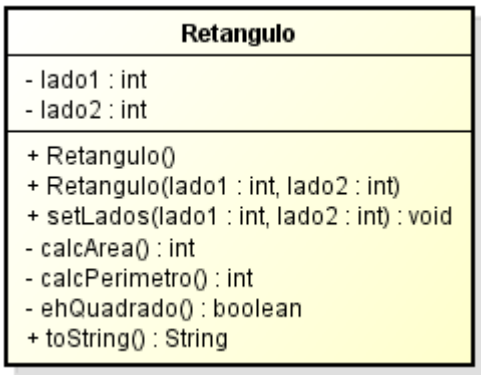


Classe Cachorro:
- nome : String → Nome do Cachorro. Não pode ser uma String vazia.
- idade : int → Idade do Cachorro. Deve ser um valor Natural.

+ Cachorro() → Construtor aridade 0.
+ Cachorro(nome : String, idade : int) → Construtor aridade total.

- late() : void → Mostra na tela que o Cachorro latiu uma vez. O latido será “au au”.
+ come() : void → Mostra na tela que o Cachorro comeu ração. Depois de comer ele late.
+ brinca() : void → Mostra na tela que o Cachorro brincou com o brinquedo. Depois de brincar ele late.
+ setNome(nome : String) : void → Modifica o nome de acordo com o argumento.
+ setIdade(idade : int) : void → Modifica a idade de acordo com o argumento.
+ toString() : String → Retorna as informações do Cachorro, para ser mostrado na tela.

2ª Questão

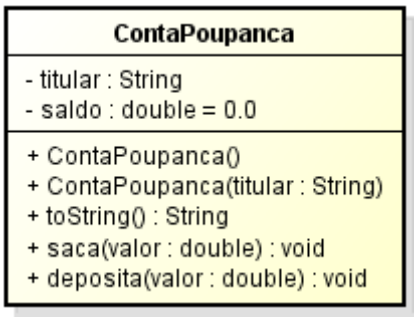


Classe Retangulo:
- lado1 : int → Lado horizontal. Deve ser um valor positivo.
- lado2 : int → Lado vertical. Deve ser um valor positivo.

+ Retangulo() → Construtor de aridade 0.
+ Retangulo(lado1 : int, lado2 : int) → Construtor completo.

+ setLados(lado1 : int, lado2 : int) : void → Modifica os lados do retângulo, baseado respectivamente nos argumentos do método.
- calcArea() : int → Retorna a área do Retangulo.
- calcPerimetro() : int → Retorna o perímetro do Retangulo.
- ehQuadrado() : boolean → Retorna **true** se o Retangulo for um quadrado. Caso contrário, retorna false. Pesquise sobre o quadrado ser um tipo especial de retângulo.
+ toString() : String → Retorna as informações do Retangulo na tela. Além dos atributos, mostra sua área, perímetro e se este retângulo é ou não um quadrado.

3ª Questão

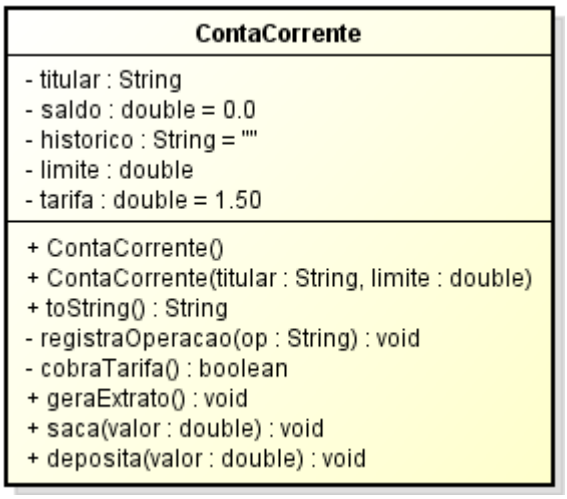


Classe ContaPoupanca:
- titular : String → Nome do titular da ContaPoupanca. Não pode ser uma String vazia.
- saldo : double = 0.0 → Saldo da ContaPoupanca. Não pode ser um número negativo.

+ ContaPoupanca() → Construtor de aridade 0.
+ ContaPoupanca(titular : String) → Construtor completo.

+ toString() : String → Retorna as informações da ContaPoupanca.
+ saca(valor : double) : void → Saca determinado valor da ContaPoupanca. O argumento não pode ser negativo.
+ deposita(valor : double) : void → Deposita determinado valor na ContaPoupanca. O argumento não pode ser negativo.

4ª Questão

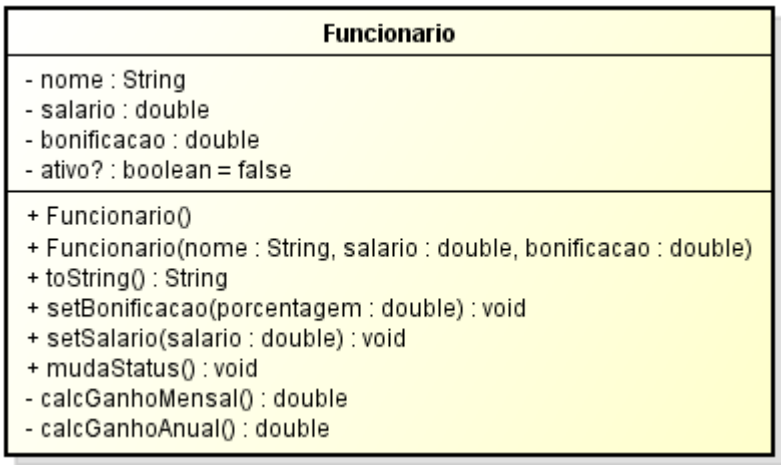


Classe ContaCorrente:
- titular : String → Nome do titular da ContaCorrente. Não pode ser uma String vazia.
- saldo : double = 0.0 → Saldo da ContaCorrente. Deve ser sempre maior que o negativo de limite.
- histórico : String = "" → String que armazenará o histórico de operações da ContaCorrente.
- limite : double → Limite da ContaCorrente. Não pode ser um valor negativo.
- tarifa : double = 1.50 → Tarifa cobrada nas operações de saque e geração de extrato.

+ ContaCorrente() → Construtor de aridade 0.
+ ContaCorrente (titular : String, limite : double) → Construtor completo.

+ toString() : String → Retorna as informações da ContaCorrente, exceto o histórico.
- registraOperacao(op : String) : void → Registra determinada operação(op) realizada no histórico, dependendo da operação realizada.
- cobraTarifa() : boolean → Cobra tarifa por operação, dependendo se há saldo disponível na ContaCorrente. Retorna **true** se a tarifa for cobrada com sucesso, e **false** caso contrário.
+ geraExtrato() : void → Mostra na tela, o histórico da conta, somente se a tarifa puder ser cobrada com sucesso. Operação tarifada. Necessita de registro no histórico.
+ saca(valor : double) : void → Retira determinado valor da ContaCorrente. O argumento não pode ser negativo. Operação tarifada. Necessita de registro no histórico em caso de sucesso. Além de considerar o valor do saque, deve-se considerar se a tarifa também pode ser cobrada sem que o saldo fique inválido.
+ deposita(valor : double) : void → Deposita determinado valor na ContaCorrente. O argumento não pode ser negativo. Necessita de registro no histórico em caso de sucesso.

5ª Questão

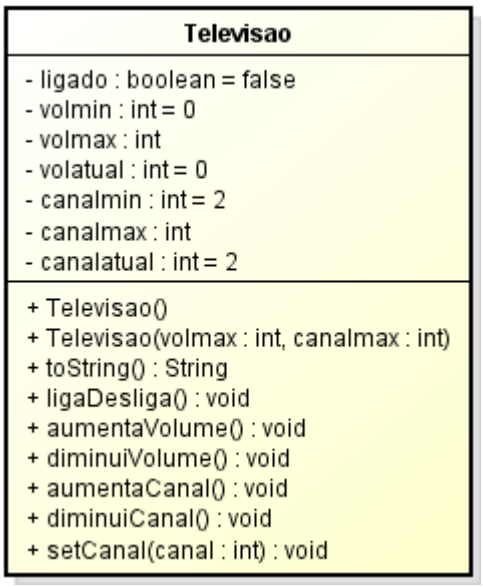


Classe Funcionario:
- nome : String → Nome do Funcionario. Não pode ser uma String vazia.
- salario : double → Importância recebida como parte da remuneração do Funcionario. Não pode ser menor que o salário mínimo(faça uma pesquisa para saber o valor vigente).
- bonificacao : double → Porcentagem utilizada para compor a remuneração do Funcionario. Não pode ser um valor negativo.
- ativo? : boolean = false → Atributo que diz se o Funcionario está trabalhando ou afastado de suas funções.

+ Funcionario() → Construtor de aridade 0.
+ Funcionario(nome : String, salário : double, bonificação : double) → Construtor completo.

+ toString() : String → Retorna as informações do Funcionario, acrescidos de seu ganho mensal e ganho anual.
+ setBonificacao(porcentagem : double) : void → Modifica a bonificacao baseado no argumento. Somente funcionários que não estão afastados poderão ter sua bonificação modificada.
+ setSalario(salario : double) : void → Modifica o salario do Funcionario baseado no argumento. O salário sempre deverá aumentar quando modificado. Somente funcionários que não estão afastados poderão ter seu salário modificado.
+ mudaStatus() : void → Afasta o Funcionario caso ele esteja trabalhado e vice-versa.
+ calcGanhoMensal() : double → Calcula o ganho mensal do Funcionario, baseado no salário e na bonificação aplicada sobre o salário.
+ calcGanhoAnual() : double → Calcula o ganho anual, baseando-se no ganho mensal do ano inteiro, considerando a gratificação natalina.

6ª Questão

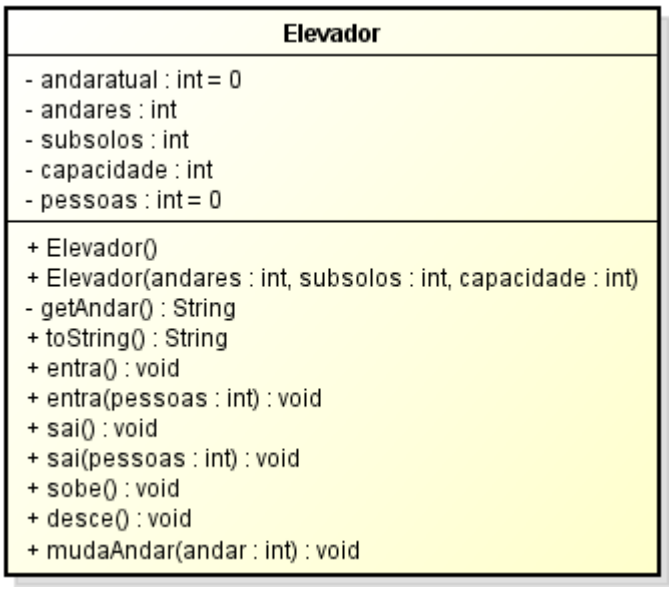


Classe Televisao:
- ligado : boolean = false → Atributo que diz se a Televisao esta ligada ou não. Não deve ser possível mudar o volume e/ou o canal se o televisor estiver desligado.
- volmin : int = 0 → Valor mínimo que o volume atual do televisor pode estar.
- volmax : int → Valor máximo que o volume atual do televisor pode estar.
- volatual : int = 0 → Volume atual do televisor, tal que volmin ≤ volatual ≤ volmax.
- canalmin : int = 2 → Canal mínimo que o televisor pode estar sintonizado.
- canalmax : int → Canal máximo que o televisor pode estar sintonizado, tal que, canalmin ≤ canalatual ≤ canalmax.
- canalatual : int = 2 → Canal atual que o televisor está sintonizado.

+ Televisao() → Construtor de aridade 0.
+ Televisao(volmax : int, canalmax : int) → Construtor completo.

+ toString() : String → Retorna as informações da Televisao.
+ ligaDesliga() : void → Liga ou desliga a TV, dependendo do estado atual, e.g. se está ligada desliga, e vice-versa.
+ aumentaVolume() : void → Aumenta em uma unidade o volume atual do televisor.
+ diminuiVolume() : void → Diminui em uma unidade o volume atual do televisor.
+ aumentaCanal() : void → Aumenta em uma unidade o canal atual do televisor. Se atingir o limite máximo, deve voltar para canalmin, de forma a haver uma alteração cíclica no canal atual.
+ diminuiCanal() : void → Diminui em uma unidade o canal atual do televisor. Se atingir o limite mínimo, deve voltar para canalmax, de forma a haver uma alteração cíclica no canal atual.
+ setCanal(canal : int) : void → Se o argumento for um valor válido, deve mudar o canal atual para o argumento utilizando os métodos aumentaCanal ou diminuiCanal.

7ª Questão

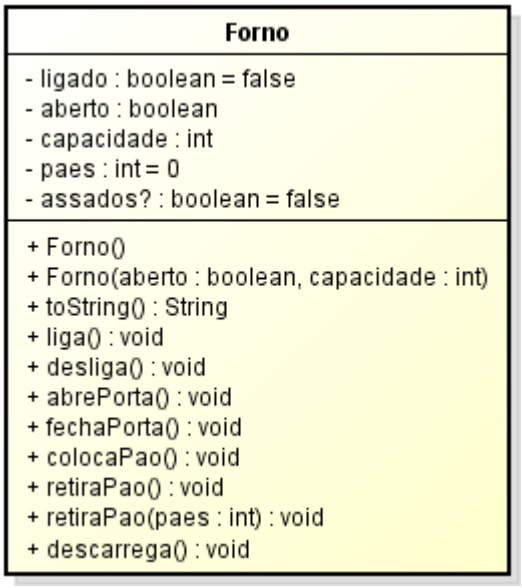


Classe Elevador:
- andaratual : int = 0 → Andar em que o elevador de encontra, tal que –subsolos ≤ andaratual ≤ andares
- andares : int → Quantidade de andares existentes no prédio acima do térreo. Deve ser um valor positivo.
- subsolos : int → Quantidade de andares existentes no prédio abaixo do térreo. Deve ser um valor positivo.
- capacidade : int → Capacidade do elevador em número de pessoas. Deve ser um número maior que 3.
- pessoas : int = 0 → Número de pessoas existentes no elevador. Respeite a capacidade do elevador.

+ Elevador() → Construtor de aridade 0.
+ Elevador(andares : int, subsolos : int, capacidade : int) → Construtor completo.

- getAndar() : String → Retorna uma string contendo a informação de onde o Elevador está. Sempre ao fornecer informações ou exibir mensagens, um elevador não está/chegou ao andar 1, 2, etc, chamam-se "1o. andar", "2o. andar", etc; Idem para andar 0, chama-se "térreo"; Idem para andar -1, -2, etc, chamam-se "subsolo 1", "subsolo 2", etc.
+ toString() : String → Retorna as informações do Elevador. O andar atual deve ser retornado com o método correspondente.
+ entra() : void → Para que uma pessoa entre no elevador. Ao entrar uma pessoa, um aviso deve ser mostrado, além da quantidade de pessoas existente no elevador.
+ entra(pessoas : int) : void → Para que uma quantidade presente no argumento de pessoas entre no elevador. Utilize o método entra anterior neste método. O argumento deve ser positivo.
+ sai() : void → Para que uma pessoa saia do elevador. Ao sair uma pessoa, um aviso deve ser mostrado, além da quantidade de pessoas existente no elevador.
+ sai(pessoas : int) : void → Para que uma quantidade presente no argumento de pessoas saia do elevador. Utilize o método sai anterior neste método. O argumento deve ser positivo.
+ sobe() : void → Para subir um andar. Ao subir um andar, um aviso deve ser mostrado, além do andar atual quando chegar.
+ desce() : void → Para descer um andar. Ao subir um andar, um aviso deve ser mostrado, além do andar atual quando chegar.
+ mudaAndar(andar : int) : void → Deve mudar para o andar desejado, utilizando os métodos sobe ou desce. Antes de sair, mostre o andar de origem. Ao chegar, mostre o andar de destino.

8ª Questão

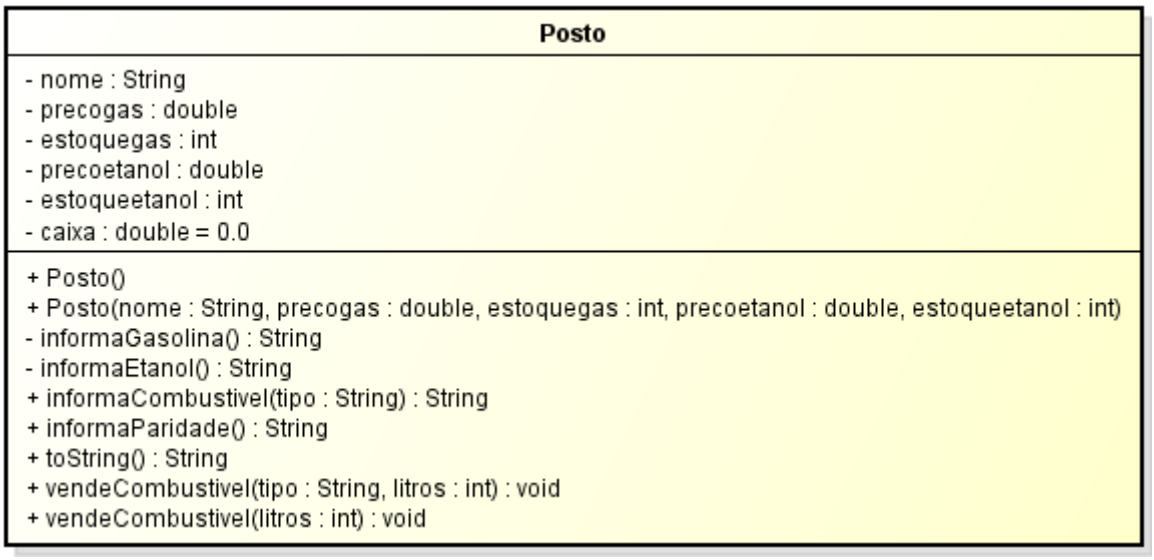


Classe Forno:
- ligado : boolean = false → Define se o forno está ligado ou não.
- aberto : boolean → Define se o forno está aberto ou não.
- capacidade : int → Capacidade do forno, em número de pães. Deve ser um valor positivo.
- paes : int = 0 → Quantidade de pães presente no forno. Deve ser sempre ≥ 0.
- assados? : boolean = false → Define se existem pães assados no Forno.

+ Forno() → Construtor de aridade 0.
+ Forno(aberto : boolean, capacidade : int) → Construtor completo.

+ toString() : String → Retorna as informações do Forno.
+ liga() : void → Para ligar o Forno. Um forno nunca é ligado de porta aberta, vazio ou se existirem pães assados dentro dele, dentre outras restrições.
+ desliga() : void → Para desligar o Forno. Quando o forno é desligado os pães em seu interior são considerados assados, dentre outras restrições.
+ abrePorta() : void → Para que a porta seja aberta. Se a porta for aberta com o aparelho ligado, o padeiro se queimará, dentre outras restrições.
+ fechaPorta() : void → Para que a porta seja fechada. Atenção às restrições;
+ colocaPao() : void → Para colocar um pão cru no Forno. Pães crus não podem ser misturados com pães assados, dentre outras restrições.
+ retiraPao() : void → Para retirar um pão. Preste atenção nas restrições. Quando todos os pães estiverem assados, e saírem do forno, não existirão pães assados no Forno.
+ retiraPao(paes : int) : void → Para retirar uma quantidade determinada de pães. Deve-se utilizar o método retiraPao anterior. Preste atenção nas restrições.
+ descarrega() : void → Para descarregar o forno. Deve-se utilizar um dos métodos retiraPao. Não se pode descarregar um forno vazio. Preste atenção nas restrições.

9ª Questão



Classe Posto:
- nome : String → Nome do Posto. Não pode ser uma String vazia.
- precogas : double → Preço da gasolina, em reais. Não pode ser menor que R$1,00.
- estoquegas : int → Estoque de gasolina, em litros.
- precoetanol : double → Preço do etanol, em reais. Não pode ser menor que R$1,00.
- estoqueetanol : int → Estoque de etanol, em litros.
- caixa : double = 0.0 → Quantia presente no caixa do Posto, proveniente da venda de combustível. Deve ser um valor ≥ 0.0.

+ Posto() → Construtor de aridade 0.
+ Posto(nome : String, precogas : double, estoquegas : int, precoetanol : double, estouqueetanol : int) → Construtor completo.

- informaGasolina() : String → Para retornar, informações acerca da gasolina, ou seja, preço e quantidade disponível. Ex. de retorno: Gasolina – Preço: R$2.89 – Estoque: 500 litros.
- informaEtanol() : String → Para retornar, informações acerca do etanol, ou seja, preço e quantidade disponível. Ex. de retorno: Etanol – Preço: R$1.89 – Estoque: 500 litros.
+ informaCombustivel(tipo : String) : String → Para retornar, informações do combustível relativo ao argumento passado, utilizando o respectivo método. Caso a string do argumento seja inválida, retorne um aviso.
+ informaParidade() : String → Para retornar, a paridade, em porcentagem, entre o valor do etanol e o valor da gasolina, além de dizer se é melhor abastecer com gasolina ou com etanol, naquele posto. Cálculo da paridade:

$$Paridade = \frac{PREÇO_{etanol}}{PREÇO_{gasolina}} * 100$$

Condições para saber se é melhor abastecer com um combustível ou com outro:
- Paridade maior que 70%: Preferência à gasolina;
- Paridade menor que 70%: Preferência ao etanol;
- Paridade igual a 70%: Preferência ao etanol devido ao seu menor impacto ambiental.

Ex. de retorno: **Paridade: 68% – Dê preferência ao etanol**.
+ toString() : String → Retorna as informações do Posto. Porém, ao retornar as informações dos combustíveis, utilize os métodos de informação. Em ao menos um dos combustíveis deve ser utilizado o método informaCombustivel. Retorne também a paridade e o combustível que é melhor abastecer no posto.
+ vendeCombustivel(tipo : String, litros : int) : void → Para que se venda o combustível presente no primeiro argumento, com a quantidade presente no segundo argumento. O Posto ganha dinheiro com a venda de combustível. Atenção às restrições. Após a venda, mostre as informações do combustível vendido e a receita atualizada do posto.
+ vendeCombustivel(litros : int) : void → Vende a quantidade presente no argumento, do combustível que possuir maior estoque. Se os estoques estiverem iguais, escolha vender etanol.

10ª Questão

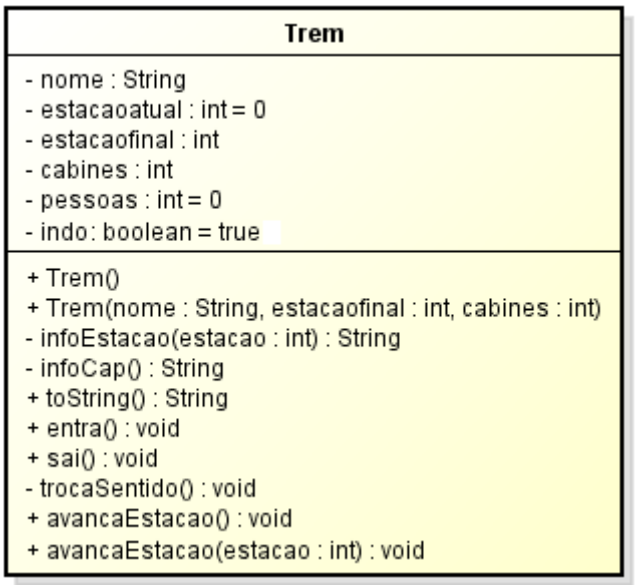


Tabela de estações do Trem

NÚMERO	NOME DA ESTAÇÃO
0	Luz
1	Prefeitura
2	Moema
3	Barra
4	Copacabana

Classe Trem:
- nome : String → Nome do trem.
- estacaoatual : int = 0 → Estação em que o Trem se encontra. Deve obedecer à tabela.
- estacaofinal : int → Estação limite que o Trem irá. Deve obedecer à tabela e ser ≥ 0.
- cabines : int → Quantidade de cabines presente no Trem. Cada cabine ocupa até 2 pessoas. Um trem deve ter no mínimo 2 cabines.
- pessoas : int = 0 → Quantidade de pessoas presente no Trem. Deve ser sempre ≥ 0. Respeite a capacidade do Trem.
- indo : boolean = true → Define se o trem está indo ou voltando. Se o trem estiver indo, seu caminho entre as estações é crescente. Caso contrário, é decrescente.

+ Trem() → Construtor de aridade 0.
+ Trem(nome : String, estacaofinal : int, cabines : int) → Construtor completo.

- infoEstacao(estação : int) : String → Para informar em qual o número e nome da estação de acordo com um argumento. O argumento deve ser uma das estações da tabela. Ex de retorno: 1 – Prefeitura.
- infoCap() : String → Para informar qual a capacidade do trem, em número de pessoas e a quantidade de cabines do trem. Ex de retorno: Cabines: 1 – Capacidade: 2 pessoas.
+ toString() : String → Retorna as informações do Trem. A estação atual e a capacidade devem ser retornadas utilizando os métodos correspondentes.
+ entra() : void → Para que uma pessoa entre no trem. Ao entrar uma pessoa, um aviso deve ser mostrado, além da quantidade de pessoas no trem, juntamente com a quantidade de cabines e a capacidade do trem.
+ sai() : void → Para que uma pessoa saia do trem. Ao sair uma pessoa, um aviso deve ser mostrado, além da quantidade de pessoas no trem, juntamente com a quantidade de cabines e a capacidade do trem.
- trocaSentido() : void → Para trocar o sentido do Trem. Isto deve acontecer caso o trem alcance a estação 0 ou a estação final.
+ avancaEstacao() : void → Para ir para a próxima estação. Preste atenção nas restrições. A próxima estação irá depender do sentido do trem. Se o trem chegar a alguma das extremidades do trecho, seu sentido deve ser trocado. Uma mensagem informando a estação de onde o trem está partindo e qual estação ele chegou deve ser mostrada.
+ avancaEstacao(estação : int) : void → Deve mudar para a estação desejada, utilizando o método avancaEstacao, chamado quantas vezes forem necessárias. A estação informada deve estar na rota do trem, dependendo da estação atual, do sentido do trem e da estação final do trem.
Ex: Um trem que está na estação 2, sentido crescente, pode ir para as estações 3 e 4, mas não pode ir para as estações 0 e 1, pois está fora do seu sentido, e nem para a estação 2, pois já está nela.