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 2-1

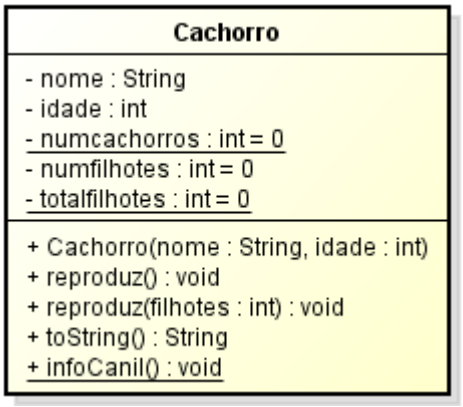
Obs: Para todos os exercícios, crie pelo menos 2 (dois) objetos, inicialize os atributos utilizando os construtores (exceto quando especificado no diagrama), e execute todos os métodos públicos para demonstrar suas funcionalidades.

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

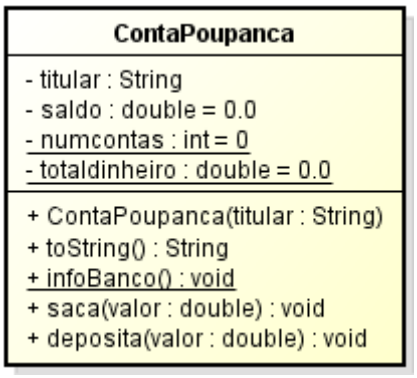


Classe Cachorro:
- nome : String → Nome do Cachorro. Não pode ser uma String vazia.
- idade : int → Idade do Cachorro. Deve ser um valor Natural.
- numcachorros : int = 0 → Total de Cachorros criado. A cada Cachorro criado, incremente este atributo de classe.
- numfilhotes : int = 0 → Quantidade de filhotes que um Cachorro gerou.
- totalfilhotes : int = 0 → Total de filhotes gerados por todos os Cachorros.

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

+ reproduz() : void → O Cachorro gera um filhote. Mostre na tela que isto ocorreu, identificando o Cachorro no processo. Incremente as quantidades de filhotes gerados.
+ reproduz(filhotes : int) : void → O Cachorro gera uma quantidade de filhotes presente no argumento. Utilize o método reproduz anterior no processo.
+ toString() : String → Retorna as informações do Cachorro, para ser mostrado na tela. Não retorne as informações da classe.
+ infoCanil() : void → Mostra na tela as informações da classe.

2ª Questão

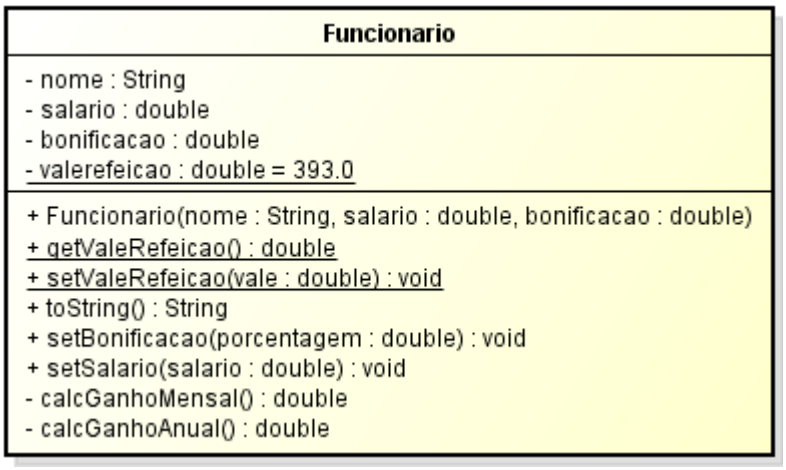


Classe ContaPoupanca:
- titular : String → Nome do titular da ContaPoupanca. Não pode ser uma String vazia.
- saldo : double = 0.0 → Saldo da ContaPoupanca. Não pode ser um número negativo.
- numcontas : int = 0 → Total de ContasPoupanca criadas no banco. A cada conta criada, incremente este atributo de classe.
- totaldinheiro : double = 0.0 → Total de saldo presente no banco. Quando entrar dinheiro ou sair dinheiro do banco, modifique este atributo de classe.

+ ContaPoupanca(titular : String) → Construtor.

+ toString() : String → Retorna as informações da ContaPoupanca. Não mostre informações referentes a todas as ContasPoupanca.
+ infoBanco() : void → Mostra na tela as informações referentes ao conjunto de ContasPoupanca, ou seja, o banco: Número de contas criadas e total de saldo presente no banco.
+ 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.

3ª Questão

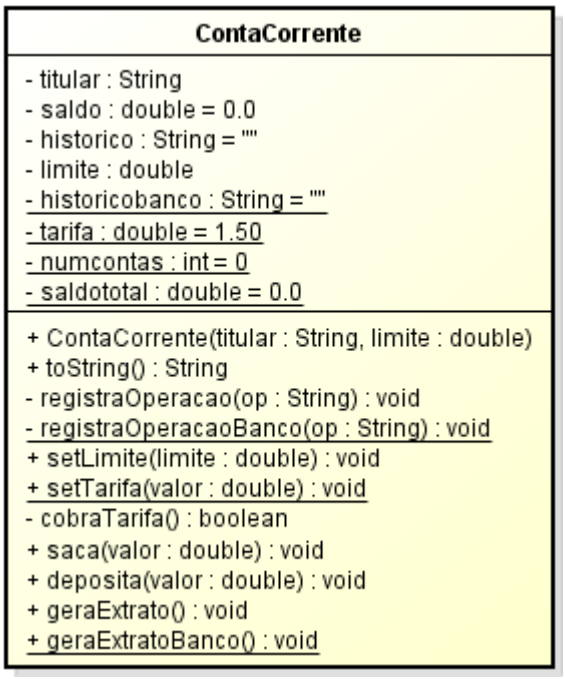


Classe Funcionario:
- nome : String → Nome do Funcionario. Não pode ser uma String vazia.
- salario : double → Importância recebida como parte da remuneração do Funcionario. Não pode ser menor que o salário mínimo(faça uma pesquisa para saber o valor vigente).
- bonificacao : double → Porcentagem utilizada para compor a remuneração do Funcionario. Não pode ser um valor negativo.
- valerefeicao : double = 393.0 → Vale-refeição recebido por todos os Funcionarios para alimentação.

+ Funcionario(nome : String, salário : double, bonificação : double) → Construtor.

+ getValeRefeicao() : double → Retorna o valor do vale-refeição dos funcionários.
+ setValeRefeicao(vale : double) : void → Modifica o valor do vale-refeição dos funcionários, baseado no argumento. O vale-refeição nunca poderá diminuir.
+ toString() : String → Retorna as informações do Funcionario, acrescidos de seu ganho mensal e ganho anual, além do vale-refeição de todos os funcionários.
+ setBonificacao(porcentagem : double) : void → Modifica a bonificacao baseado no argumento.
+ setSalario(salario : double) : void → Modifica o salario do Funcionario baseado no argumento. O salário sempre deverá aumentar quando modificado.
+ calcGanhoMensal() : double → Calcula o ganho mensal do Funcionario, baseado no salário e na bonificação aplicada sobre o salário. Também deve ser levado em consideração o vale-refeição.
+ calcGanhoAnual() : double → Calcula o ganho anual, baseando-se no ganho mensal do ano inteiro, considerando a gratificação natalina.

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. Nunca poderá ser menor que o negativo de saldo.
- historicobanco : String = “” → String que armazenará o histórico de operações de todas as ContasCorrente.
- tarifa : double = 1.50 → Tarifa cobrada nas operações de saque e geração de extrato.
- numcontas : int = 0 → Total de ContasCorrente criadas no banco. A cada conta criada, incremente este atributo de classe.
- saldototal : double = 0.0 → Total de saldo presente no banco. Quando entrar dinheiro ou sair dinheiro do banco, modifique este atributo de classe.

+ ContaCorrente (titular : String, limite : double) → Construtor.

+ 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.
- registraOperacaoBanco(op : String) : void → Registra determinada operação(op) realizada no histórico, dependendo da operação realizada. Esta operação será registrada no histórico do banco e deverá sempre acompanhar o nome do titular.
+ setLimite(limite : double) : void → Modifica o limite de acordo com o argumento.
+ setTarifa(valor : double) : void → Modifica a tarifa do banco de acordo com o argumento.
- 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.
+ saca(valor : double) : void → Retira determinado valor da ContaCorrente. O argumento não pode ser negativo. Operação tarifada. Necessita de registro no histórico e no histórico do banco 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 e no histórico do banco em caso de sucesso.
+ 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 e no histórico do banco.
+ geraExtratoBanco() : void → Mostra na tela, o histórico do banco.
