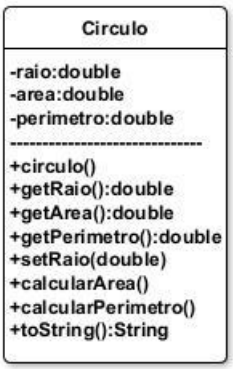
1) Construa uma classe **Círculo** e uma classe consumidora para validar o funcionamento de todos os métodos da respectiva classe.



Equações matemáticas:

- **Área = raio * raio * 3.14**
- **Perímetro = 2 * 3.14 * raio**

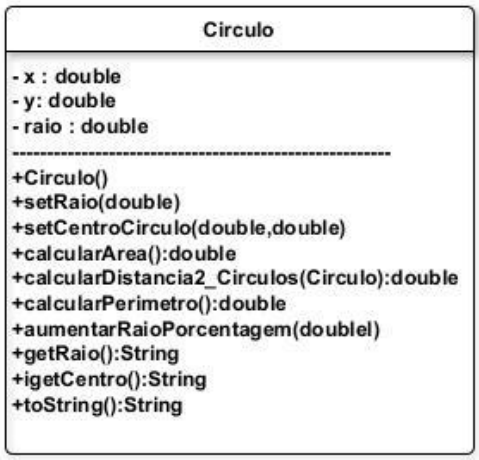
A classe deve ter:

- Os atributos raio, área e perímetro. Todos do tipo ponto flutuante e privados.
- Construtor vazio.
- Métodos públicos:
    - **getRaio ()**: retorna o valor do atributo raio do tipo ponto flutuante do objeto;
    - **getArea ()**: retorna o valor do atributo área do tipo ponto flutuante do objeto;
    - **getPerimetro ()**: retorna o valor do atributo perímetro do tipo ponto flutuante do objeto;
    - **setRaio(double)**: recebe o valor do raio e coloca o mesmo no respectivo atributo.
    - Construa um método público **calculaArea ()** que calcula o valor da área do objeto e coloque a mesma no respectivo atributo.
    - Construa um método público **calculaPerimetro ()** que calcula o valor do perímetro do objeto e coloque o mesmo no respectivo atributo.
    - Construa um método público **toString ()** que mostra todos os atributos do objeto no formato de uma String.

Para testar o programa você deve: Instanciar a classe criando um objeto;

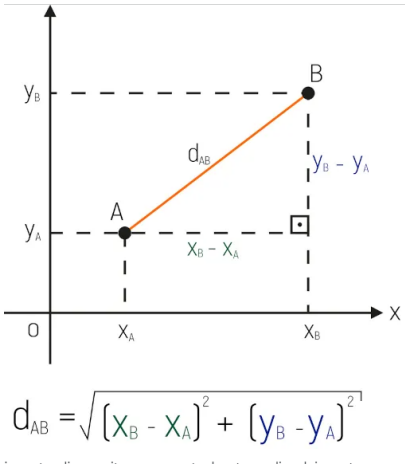
- Atribuir o valor 5.0 ao raio;
- Chamar o método que calcula a área;
- Chamar o método que calcula o perímetro;
- Mostrar os valores da classe pelo método toSring ();
- Mudar o raio para 10.0;
- Chamar o método que calcula a área;
- Chamar o método que calcula o perímetro;
- Mostrar os valores da classe pelo método toSring ();

2) Definir uma classe que represente um círculo.



Equações matemáticas:

- **Área = raio * raio * 3.14**
- **Perímetro = 2 * 3.14 * raio**
- **Distância entre dois pontos:**



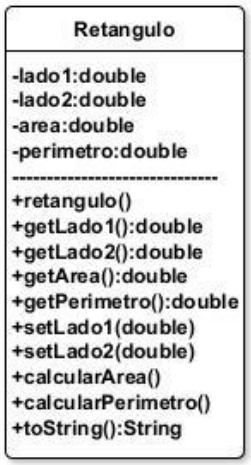
$$d_{AB} = \sqrt{(X_B - X_A)^2 + (y_B - y_A)^2}$$

A classe possui os atributos privados x, y, raio.
E métodos Públicos para:

- Construtor vazio.
- Definir o raio do círculo, dado um número real;
- Definir o centro do círculo, dada uma posição (X,Y);
- Calcular a área do círculo;
- Calcular a distância entre os centros de 2 círculos;
- Calcular o perímetro do círculo.
- Aumentar o raio do círculo, dado um percentual de aumento;
- Construa um método getRaio () que retorna o raio do círculo;
- Construa o método getCentro () que retorna uma string no padrão (X,Y) do centro do círculo.
- Construa um método público **toString ()** que mostra todos os atributos do objeto no formato de uma String.

Criar um programa principal para testar a classe.

3) Construa uma classe **Retângulo** e uma classe consumidora para validar o funcionamento de todos os métodos da respectiva classe.



Atributos:

- **lado1**: privado do tipo ponto flutuante.
- **lado2**: privado do tipo ponto flutuante.
- **área**: privado do tipo ponto flutuante.
- **perímetro**: privado do tipo ponto flutuante.

Construtor:

- O **construtor** deve ser vazio.

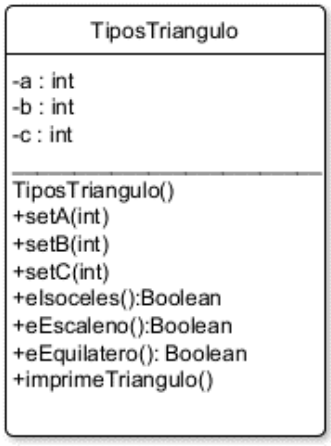
- **Métodos públicos:**
  - getLado1(): publico, retorna o valor do tipo ponto flutuante que está no atributo lado1.
  - getLado2(): publico, retorna o valor do tipo ponto flutuante que está no atributo lado2.
  - getArea (): publico, retorna o valor do tipo ponto flutuante que está no atributo área.
  - getPerimetro (): publico, retorna o valor do tipo ponto flutuante que está no atributo perímetro.
  - setLado1(double): publico, coloca o valor do tipo ponto flutuante no atributo lado1.
  - setLado2(double): publico, coloca o valor do tipo ponto flutuante no atributo lado2.
  - Construa um método público **calculaArea ()** que calcula o valor da área do objeto e coloque a mesma no respectivo atributo.
  - Construa um método público **calculaPerimetro ()** que calcula o valor do perímetro do objeto e coloque o mesmo no respectivo atributo.
  - Construa um método público **toString ()** que mostra todos os atributos do objeto no formato de uma String.

Para testar o programa você deve:

- Instanciar a classe criando um objeto vazio;
- Atribuir o valor 10.0 ao lado1
- Atribuir a valor 7.0 ao lado2
- Chamar o método que calcula a área;
- Chamar o método que calcula o perímetro;

- Mostrar o valores da classe pelo método toSring();
- Mudar o lado para 8.0;
- Chamar o método que calcula a área;
- Chamar o método que calcula o perímetro;
- Mostrar o valores da classe pelo método toSring();

4) Com base no diagrama abaixo construa a classe TiposTriangulo e uma classe consumidora para testar o funcionamento de todos os métodos e atributos.



Instruções

Um triângulo pode ser classificado com relação aos seus lados em:

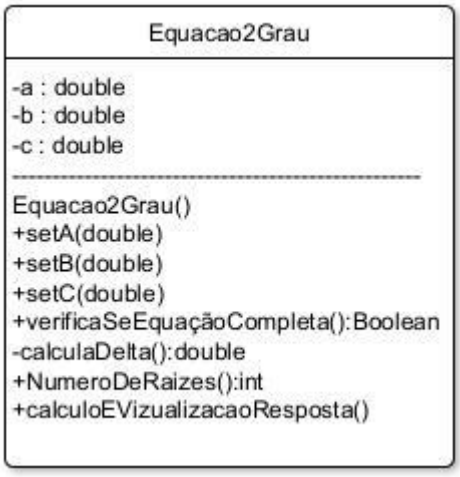
- equilátero: todos os lados do mesmo tamanho;
- isósceles: dois lados do mesmo tamanho;
- escaleno: todos os lados com medidas distintas.

Detalhamento:

a) A classe deve ter três atributos do tipo inteiro;
b) O construtor a ser implementado é vazio, ou seja, não inicializa os parâmetros.
c) Os métodos setA, setB e SetC só devem ser atualizados se os valores de a, b, c se os valores recebidos forem positivos.
d) O método eIsoceles retorna **true** quando o triângulo for isósceles e **false** caso contrário.
e) O método eEscaleno retorna **true** quando o triângulo for escaleno e **false** caso contrário.
f) O método eEquilatero retorna **true** quando o triângulo for Equilátero e **false** caso contrário.
g) O método imprimeTriangulo chama um método da Classe **JOptionPane** para mostrar o tamanho dos lados do triângulo e o tipo de triangulo.

A classe consumidora deve instanciara a classe **TiposTriangulo** e testar todos os métodos.

5) Com base no diagrama abaixo construa a classe Equacao2Grau e uma classe consumidora para testar o funcionamento de todos os métodos e atributos.



Equação a ser utilizada

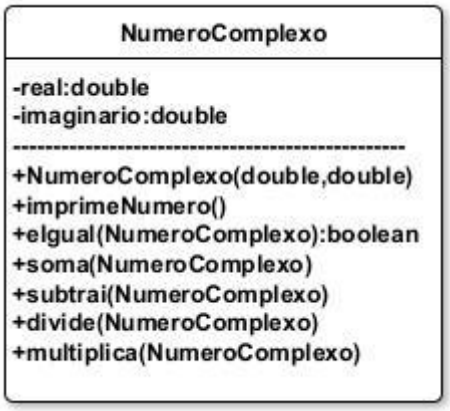
$$x = \frac{-b \pm \sqrt{\Delta}}{2a}; \quad \Delta = b^2 - 4ac$$

Detalhamento:

a) A classe deve ter três atributos do tipo ponto flutuante;
b) O construtor a ser implementado é vazio e não inicializa atributos.
c) Os métodos setA, setB e SetC só devem atualizar os atributos a, b, c se os valores recebidos forem diferentes de zero.
d) O método verificaSeEquacaoCompleta retorna true quando todos os atributos forem diferentes de zero e false caso o contrário.
e) O método calculaDelta faz o cálculo do delta conforme a equação acima e retorna um valor do tipo double.
f) O método numeroDeRaizes calcula a quantidade de raízes e retorna um valor do tipo inteiro.
g) O método CalculoEVisualizacaoResposta faz o cálculo das raízes conforme a equação acima e chama um método da Classe JOptionPane para mostrar a resposta.

   A **classe consumidora** deve instanciara **a classe Equacao2Grau** e testar todos os métodos.

6)



• **Construtor**, que recebe dois valores como argumentos para inicializar os campos da classe (parte real e imaginária);

Escreva em Java a classe NumeroComplexo que represente um número complexo. A classe deverá ter os seguintes métodos:

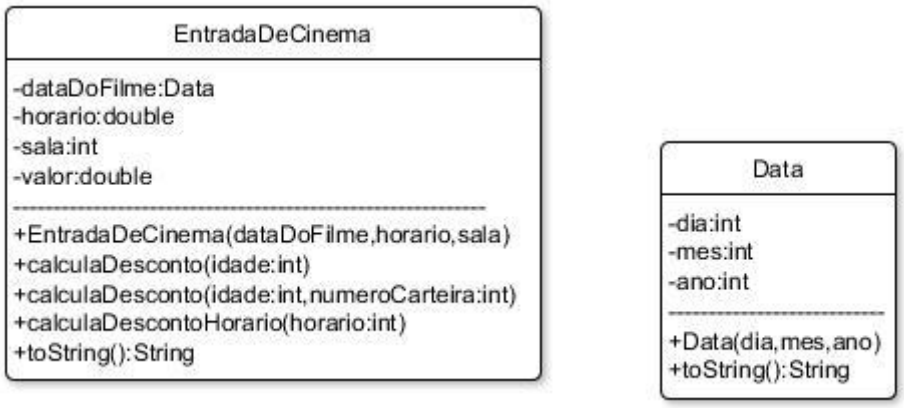
Métodos

a) **imprimeNumero**, que deve imprimir o número complexo encapsulado usando a notação **a + bi** onde **a** é a parte real e **b** a imaginária;
b) **eIgual**, que recebe outra instância da classe NumeroComplexo e retorna true se os valores dos campos encapsulados forem iguais aos da instância passada como argumento e false caso contrario;
c) **soma**, que recebe outra instância da classe NumeroComplexo e soma este número complexo com o encapsulado usando a fórmula (a + bi) + (c + di) = (a + c) + (b + d)i;
d) **subtrai**, que recebe outra instância da classe NumeroComplexo e subtrai o argumento do número complexo encapsulado usando a fórmula
(a + bi) − (c + di) = (a − c) + (b − d)i;
e) **multiplica**, que recebe outra instância da classe NumeroComplexo e multiplica este número complexo com o encapsulado usando a fórmula
(a + bi) ∗ (c + di) = (ac − bd) + (ad + bc)i;
f) **divide**, que recebe outra instância da classe NumeroComplexo e divide o número encapsulado pelo passado como argumento usando a fórmula
(a+bi)/(c+di) = (ac+bd) /(c^2+d^2) + (bc−ad)/(c^2+d^2)i

Classe consumidora

Deve ser construído uma classe consumidora com o método **public static main** que a partir da classe NumeroComplexo que instancie um objeto e mostre o funcionamento de todos os seus métodos;

7)



Escreva em Java as classes EntradaDeCinema e Data com a estrutura de dados e métodos mostrado acima.

Características da classe Data

- O construtor recebe dia, mês e ano.
- O método toString() retorna a data no formato dia/mês/ano

Características da classe EntradaDeCimena.

- O construtor recebe a data do filme, o horário e a sala. O preço do bilhete é atribuído internamente no valor de R$ 20,00.
- **Métodos:**
  a) **CalculaDesconto**: que deve receber como parâmetro a idade do cliente (do tipo int) e caso seja menor de 12 anos, deve ser dado um desconto de 50% no valor normal.

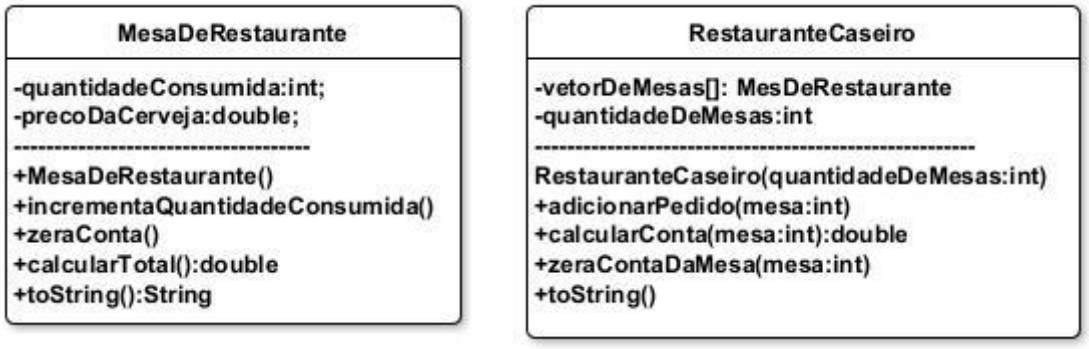
  b) **CalculoDesconto:** O método recebe dois parâmetros: a idade do cliente (do tipo **int**) e o número de sua carteira de estudante (do tipo **int**). Conforme as seguintes condições:
    - Se o cliente tiver menos de 12 anos, será aplicado um desconto de 50% no valor normal.
    - Se o cliente tiver entre 12 e 15 anos, será aplicado um desconto de 40% no valor normal.
    - Se o cliente tiver entre 16 e 20 anos, será aplicado um desconto de 30% no valor normal.
    - Se o cliente tiver mais de 20 anos, será aplicado um desconto de 20% no valor normal.

  c) CalculoDescontoHorario: esse método deve dar um desconto de 10% sobre o valor aferido após todas as outras opções de desconto, caso o horário do filme seja antes das 16 horas.

  d) toString(): que deve imprimir todos os dados do ingresso.

Classe consumidora: Deve ser construído uma classe consumidora com o método **public static main** que a partir da classe Data e EntradaDeCinema que instancie os objetos e mostre o funcionamento de todos os seus métodos;

8) Escreva em Java as classes mostradas abaixo:



A classe MesaDeRestaurante possui dois atributos privados:

- quantidadeConsumida: inicializado como zero construtor.
- precoDaCerveja: Inicia com 9.90 no construtor.

O construtor é vazio, não recebe nenhum parametro.

Métodos:

- incrementarQuantidadeConsumida(): incrementa em 1 o atributo quantidadeConsumida.
- zerarConta(): zera o atributo quantidadeConsumida.
- calcularTotal(): multiplica quantidadeConsumida pelo precoDaCerveja e retorna o resultado.
- toString(): que deve imprimir todos os dados da classe.

A classe RestauranteCaseiro possui dois atributos privados:

- vetorDeMesas: um vetor onde cada posição armazena um objeto do tipo MesaDeRestaurante.
- quantidadeDeMesas: quantidade máxima de objetos a serem armazenados no vetor.
- toString(): mostra os dados da classe.

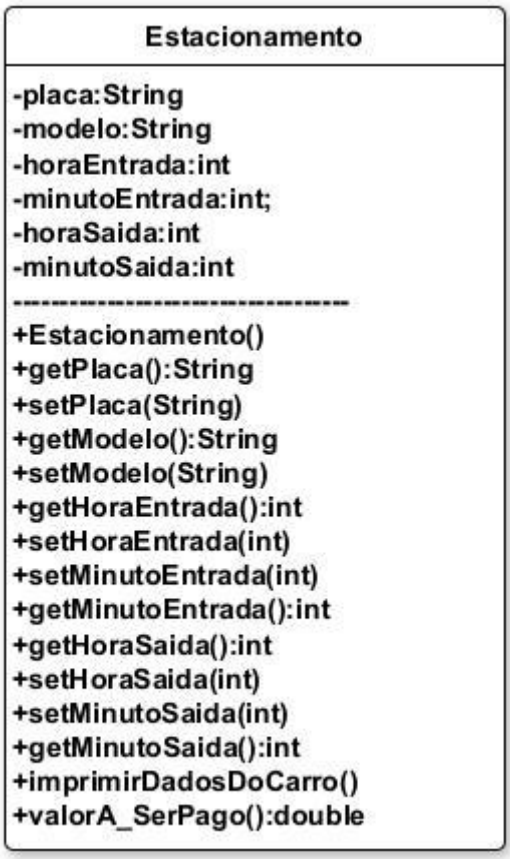
O construtor recebe a quantidade máxima de objetos a serem armazenados no vetor e inicializa os atributos necessários.

Métodos:

- adicionarPedido(mesa): adiciona uma cerveja na mesa passada como parâmetro.
- calcularConta(mesa): calcula a conta de uma determinada mesa do restaurante retornando o valor.
- zeraContaDaMesa(mesa): zera a conta de uma determinada mesa do restaurante.
- toString(): mostra os dados de todas as mesas do restaurante.

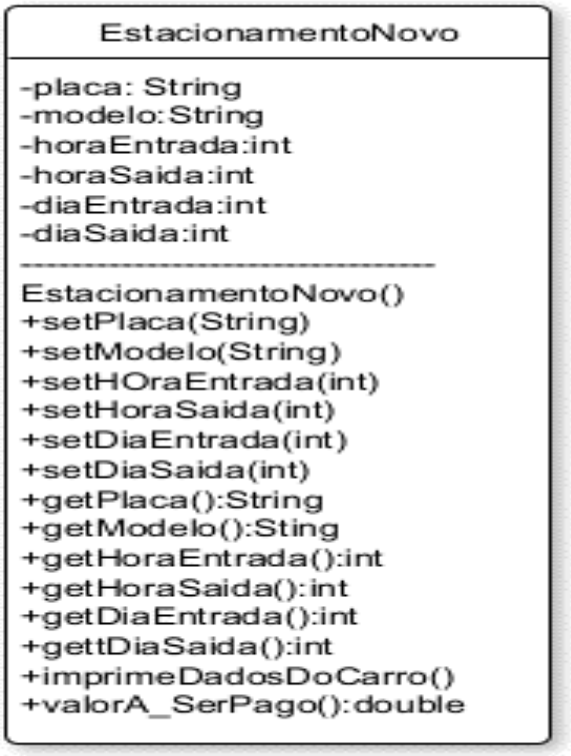
Classe Consumidora: Deve ser construída uma classe consumidora com o método public static main que instancie os objetos e demonstre o funcionamento de todos os seus métodos.

9) Implemente as classes mostradas no exercício. O Estacionamento só funciona no período diurno, ou seja, não há pernoite de carros. O valor a ser pago é de R$ 5,00 reais a cada 60 minutos.



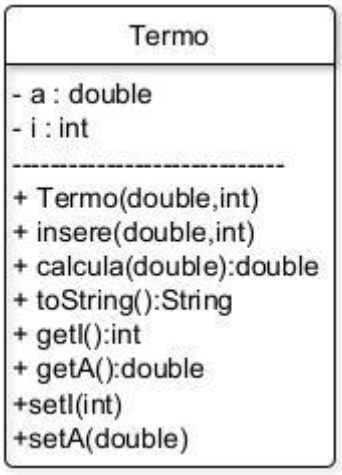
Deve ser construído uma classe consumidora com o método **public static main** que a partir da classe Estacionamento instancie um objeto e mostre o funcionamento de todos os seus métodos;

10) Implemente as classes mostradas no exercício. O Estacionamento só funciona 24 horas (Exceto na virada do mês). O valor a ser pago é de R$ 5,00 reais a hora e a R$ 50 reais a diária. Caso a quantidade de horas em um dia ultrapasse as 10 horas, o mesmo deve cobrar a diária.



11) Considere um polinômio de grau n:

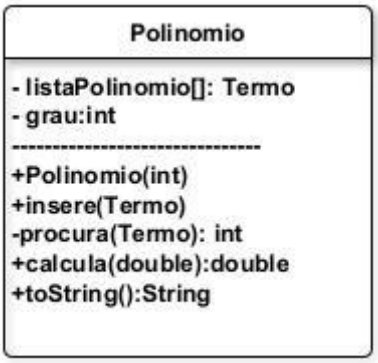
$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0 x^0$$



Escreva uma classe Termo que represente um termo deste polinômio com os seguintes métodos:

- **Construtor:** Recebe dois parâmetros: a_i e i, e cria um objeto em memória na forma $a_i . x^i$
- **Insere:** Recebe um objeto da classe Termo e substitui os valores $a_i . x^i$ do termo corrente por aqueles do termo recebido como parâmetro.
- **Calcula:** Recebe um valor de x como parâmetro e retorna o valor do termo calculado.
- **toString():** Retorna os dados do Termo.
- **getI:** retorna o valor de i
- **getA:** retorna o valor de a.
- **setI:** coloca o valor em i.
- **setA:** coloca o valor em A
- **A classe consumidora** deve instanciara a classe **Termo** e testar todos os métodos.

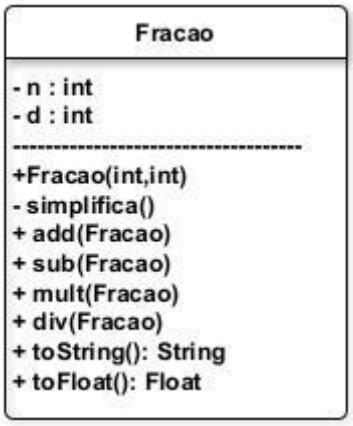
12)Escreva uma **classe Polinômio** que representa polinômio completo na forma de uma sequência de objetos da classe Termo, com os seguintes métodos:



- **Construtor:** Recebe o grau do Termo e cria um polinômio em memória na forma: P(x) = $a_i . x^i$
- **Insere:** Recebe um objeto da classe Termo e adiciona o termo $a_i . x^i$ ao polinômio recebido como parâmetro. O polinômio pode ter um terno $a_q . x^q$ cujo valor de q seja igual a i, neste caso a função deve unificar ambos em um único termo.
- **Procura:** Recebe um objeto do tipo termo e retorna a posição que o mesmo está na lista. Caso não esteja na lista deve ser retornado o número -1.
- **Calcula:** Recebe um valor de x como parâmetro e retorna o valor de P(x).
- **toString():** Retorna os dados do Polinômio.

  Acrescente os métodos que achar necessários nas classes solicitadas.
  Construa uma classe consumidora que seja capaz de testar o funcionamento da classe polinômio.

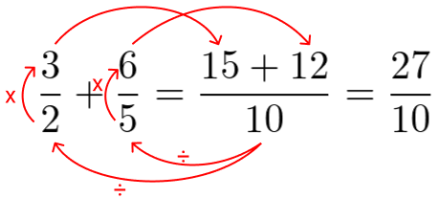
13) Implementar uma classe de frações ordinárias. Sua classe deve se chamar Fração e um objeto dessa classe deve ter dois atributos numerador e denominador, ambos inteiros e os seguintes métodos públicos e privados:



construtor: recebe dois números inteiros, sendo eles o numerador e o denominador da fração.

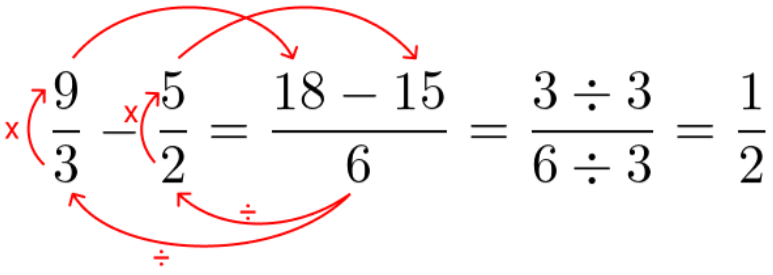
Simplifica: recebe uma fração e simplifica ela fazendo com que o numerador e o denominador sejam primos entre si.

Soma: soma da fração do objeto com a fração recebida como parâmetro.



$$\frac{3}{2} + \frac{6}{5} = \frac{15+12}{10} = \frac{27}{10}$$

Subtração: subtrai da fração do objeto com a fração recebida como parâmetro



$$\frac{9}{3} - \frac{5}{2} = \frac{18-15}{6} = \frac{3 \div 3}{6 \div 3} = \frac{1}{2}$$

Produto: multiplica da fração do objeto com a fração recebida como parâmetro

$$\frac{5}{3} \times \frac{3}{5} \times \frac{15}{3} = \frac{5 \times 3 \times 15}{3 \times 5 \times 3} = \frac{225}{45} = 5$$

Divisão: divisão da fração do objeto com a fração recebida como parâmetro

$$\frac{\frac{3}{5}}{\frac{7}{3}} = \frac{3}{5} \times \frac{3}{7} = \frac{3 \times 3}{5 \times 7} = \frac{9}{35}$$

“A divisão entre duas frações segue um procedimento simples que consiste em repetir a fração que é o dividendo e multiplicar pela fração inversa do divisor.”

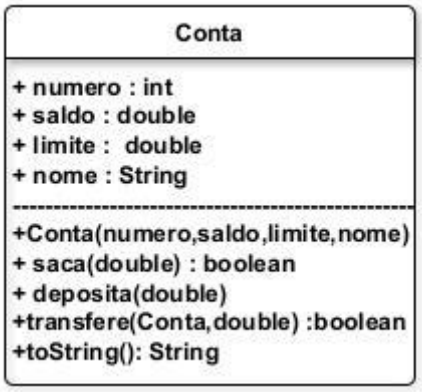
toFloat: conversão para float.

toString: conversão para string.;

Todos os métodos que retornam um objeto Fracao devem fazê-lo de forma que a fração representada esteja na forma simplificada ou seja, numerador e denominador devem ser primos entre si.

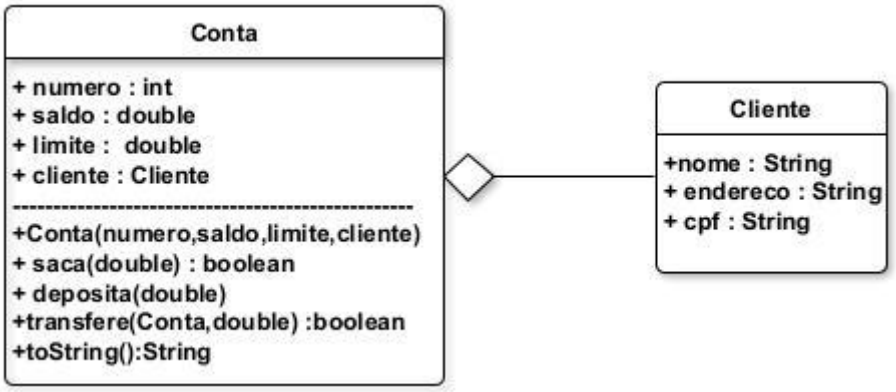
Obs: Um conjunto de números inteiros é chamado de mutuamente primo se não existir um inteiro maior do que 1 que divida todos os elementos. Assim chamamos de números primos entre si um conjunto de dois ou mais números naturais cujo único divisor comum a todos eles seja o número 1.

14) Implemente as classes mostradas no exercício.



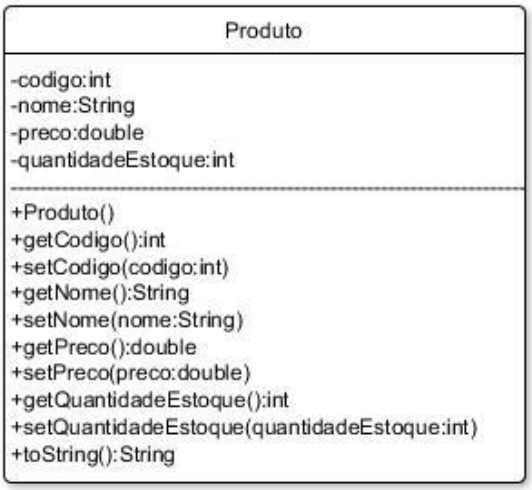
- Os atributos número, saldo, limite e nome são do tipo público;
- O construtor recebe obrigatoriamente os campos número, saldo, limite e nome.
- Todos os métodos são públicos.
- O método saca recebe um valor a ser sacado e retorna true se a operação ocorreu com sucesso e false caso contrário.
- O método deposita recebe o valor a ser depositado.
- O método transfere recebe uma conta e o valor a ser transferido para ela.
- O método toString retorna uma String com todos os dados da conta.
- **A classe consumidora** deve instanciara a classe **Conta** e testar todos os métodos.

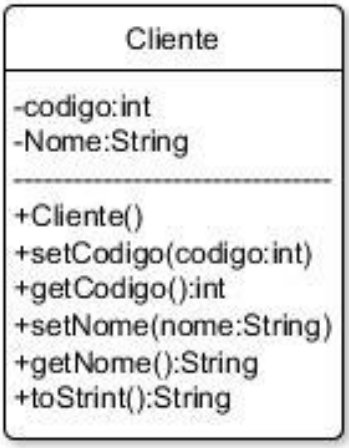
15) Implemente as classes mostradas no exercício.

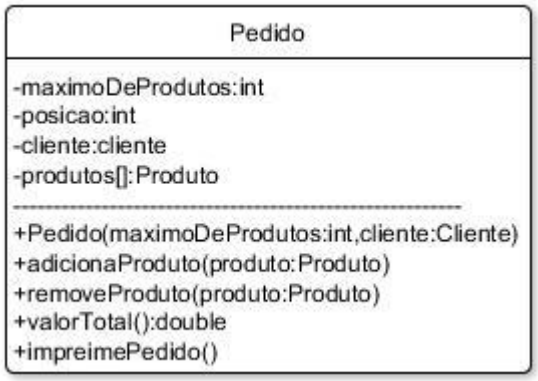


- Os atributos número, saldo, limite e cliente são do tipo público;
- O construtor recebe obrigatoriamente os campos número, saldo, limite e cliente.
- Todos os métodos são públicos.
- O método saca recebe um valor a ser sacado e retorna true se a operação ocorreu com sucesso e false caso contrário.
- O método deposita recebe o valor a ser depositado.
- O método transfere recebe uma conta e o valor a ser transferido para a mesma.
- O método toString retorna uma String com todos os dados da conta.
- **A classe consumidora** deve instanciara a classe **Conta** e testar todos os métodos.

16) Implemente as classes mostradas no exercício.







Implemente as classes mostradas no exercício. O supermercado vende diferentes tipos de produtos. Cada produto tem um código, nome, preço e quantidade em estoque. Um pedido é composto por um cliente e um vetor com todos os produtos adicionado pelo usuário.

Deve ser construído uma classe consumidora com o método **public static main** que a partir da classe Pedido instancie um objeto e mostre o funcionamento de todos os seus métodos;

17)Enunciado de Exercício: Exploração do Conceito de Herança em Java

Este exercício ilustra como uma classe base pode fornecer atributos e comportamentos comuns para classes derivadas, que podem ser especializadas com novos atributos e métodos sobrescritos.

Você foi contratado por uma empresa de gestão de zoológicos para desenvolver um sistema de inventário que organize informações sobre os animais. O sistema deve ser simples, mas suficiente para demonstrar o uso de herança em Java.

Requisitos: todos os atributos serão privados e todos os métodos serão públicos.

1. Classe base (Animal):
    - Crie uma classe Animal com os seguintes atributos:
        - nome (String): nome do animal.
        - idade (int): idade do animal.
        - som (String): som que o animal emite.
    - Inclua os seguintes métodos:
        - Construtor para inicializar os atributos.
        - Métodos get e set para os atributos.
        - Método emitirSom(): exibe no console a mensagem <nome> faz <som>!.
2. Classes derivadas:
    - Crie pelo menos duas classes que herdem de Animal e representem tipos específicos de animais:
        - Cachorro:
            - Atributo adicional: raca (String).
            - Sobrescreva o método emitirSom() para exibir a mensagem: <nome>, o <raca>, faz au au!.
        - Gato:
            - Atributo adicional: pelagem (String).
            - Sobrescreva o método emitirSom() para exibir a mensagem: <nome>, com pelagem <pelagem>, faz miau!.
3. Classe de teste (TesteZoologico):
    - Crie uma classe com o método main para instanciar objetos das classes Animal, Cachorro e Gato.
    - Teste:

- Criação de instâncias.
- Chamada dos métodos para acessar e modificar os atributos.
- Chamada do método emitirSom() para cada objeto.

Exemplo de execução:

- Bob faz som genérico!
- Max, o Labrador, faz au au!
- Mia, com pelagem cinza, faz miau!

18) Enunciado de Exercício: Sistema de Gestão de Funcionários com Herança em Java

Este exercício explora herança avançada em Java, incluindo o uso de classes abstratas, sobrescrita de métodos, e polimorfismo, demonstrando como diferentes tipos de objetos podem compartilhar uma estrutura comum, mas se comportar de maneiras específicas.

Você foi contratado para desenvolver um sistema que gerencie diferentes tipos de funcionários em uma empresa. O sistema deve ser capaz de armazenar informações básicas de cada funcionário, calcular seus salários e identificar suas funções, utilizando o conceito de **herança**.

Requisitos:

1. **Classe base (Funcionario)**:
   Crie uma classe abstrata Funcionario com os seguintes atributos e métodos:
   - **Atributos**:
     - nome (String): nome do funcionário.
     - cpf (String): CPF do funcionário.
     - salarioBase (double): salário base do funcionário.
   - **Métodos**:
     - Construtor para inicializar os atributos.
     - Métodos get e set para todos os atributos.
     - Método abstrato calcularSalario(): retorna o salário total do funcionário (a ser implementado nas subclasses).
     - Método exibirInformacoes(): exibe no console os detalhes do funcionário, incluindo o salário calculado.
2. **Classes derivadas**:
   Crie pelo menos três classes que herdem de Funcionario e representem tipos específicos de funcionários:
   - **Gerente**:
     - Atributos adicionais:
       - bonus (double): valor do bônus recebido pelo gerente.
     - Implementação do método calcularSalario(): retorna salarioBase + bonus.
   - **Vendedor**:
     - Atributos adicionais:
       - comissao (double): percentual de comissão.
       - vendas (double): valor total das vendas realizadas.

        - Implementação do método calcularSalario(): retorna salarioBase + (vendas * comissao).
    - **Estagiario**:
        - Atributos adicionais:
            - horasTrabalhadas (int): número de horas trabalhadas.
            - valorHora (double): valor pago por hora.
        - Implementação do método calcularSalario(): retorna horasTrabalhadas * valorHora.

3. **Classe de teste (TesteEmpresa)**:
    - Crie uma classe com o método main para instanciar objetos das classes Gerente, Vendedor e Estagiario.
    - Teste:
        - Criação de instâncias com diferentes valores.
        - Exibição de informações completas de cada funcionário (incluindo salário total).

Exemplo de Execução:

Gerente:

Nome: Ana Silva

CPF: 123.456.789-00

Salário Base: 8000.0

Bônus: 2000.0

Salário Total: 10000.0

Vendedor:

Nome: João Santos

CPF: 987.654.321-00

Salário Base: 3000.0

Vendas: 50000.0

Comissão: 0.1

Salário Total: 8000.0

Estagiário:

Nome: Maria Oliveira

CPF: 321.654.987-00

Horas Trabalhadas: 100

Valor por Hora: 20.0

Salário Total: 2000.0