UFABC	Universidade Federal do ABC Disciplina: Programação Orientada a Objetos 2Q-2019

Atividade 3: Construtores, Sobrecarga de métodos, Atributos / métodos de classe

Obs.: Para entrega: a pasta src com os arquivos .java de cada um dos projetos. Renomeie cada pasta de modo a identificar o exercício correspondente. Submeta uma pasta compactada na atividade do Tidia.

Exercício 1

Escreva uma classe **ConversaoDeTemperatura** que contenha métodos estáticos para calcular a conversão entre diferentes escalas de temperatura. Considere as fórmulas de conversão abaixo:

- Celsius (C) para Fahrenheit (F): F = (9.0/5.0)*C + 32
- Celsius (C) para Kelvin (K): K = C + 273.15
- Fahrenheit (F) para Celsius (C): C = (5.0/9.0)*(F - 32)
- Kelvin (K) para Celsius (C): C = K- 273.15
- Fahrenheit (F) para Kelvin (K): K = (5.0/9.0)*(F + 459.67)
- Kelvin (K) para Fahrenheit (F): F = (9.0/5.0)*K – 459.67

Crie uma classe **Teste**, faça chamadas dos métodos e apresente os resultados. Pode-se solicitar ao usuário o valor da temperatura para conversão e a opção de conversão desejada.

Exercício 2

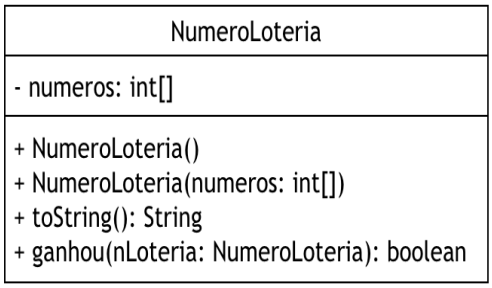
Crie uma aplicação para simular o atendimento de fila única com sistema de senha:

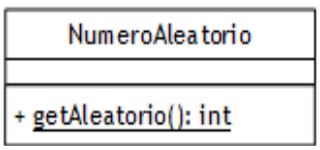
- Crie uma classe para o caixa de atendimento (**CaixaAtendimento**)
- Cada caixa possui um identificador e uma senha (ambos do tipo inteiro)
- Crie o construtor desta classe que deve receber o identificador por parâmetro
- Os caixas devem executar a seguinte operação:
  - Chamar o próximo cliente da fila, usando a informação da senha atual (método `chamaProximoFila()`). Este método deve apenas imprimir uma mensagem indicando a próxima senha a ser atendida e o número (identificador) do caixa ao qual o usuário deverá se dirigir.

Crie a classe **Principal** e faça a instanciação de 5 caixas de atendimento, passando um inteiro de 1 a 5 que será o identificador de cada um deles. Em seguida, para simular a chamada de clientes da fila, chame o método `chamaProximoFila()` aleatoriamente em cada objeto caixa e verifique se a chamada da senha está sendo feita corretamente.

Exercício 3

Vamos criar um sistema de loteria. Para isso, haverá duas classes no pacote **loteria**:





A classe **NumeroAleatorio** possui um método **getAleatorio()** que retorna um número inteiro gerado aleatoriamente. Utilize o método **Math.random()** para gerar o número aleatório.

A classe **NumeroLoteria** possui um vetor de inteiros e os seguintes métodos:

- **NumeroLoteria()**: construtor padrão/*default* que inicializa o vetor com 3 números aleatórios (use a classe **NumeroAleatorio**); Não há a necessidade de verificar se os números gerados são diferentes.
- **NumeroLoteria(int[] numeros)**: construtor com parâmetro que usa os números do argumento ao invés dos números aleatórios para inicializar o vetor.
- **toString()**: retorna uma String contendo os números do vetor
- **ganhou(NumeroLoteria nLoteria)**: compara o número passado como argumento com o número da instância. Compare os números na ordem que foram passados, não há a necessidade de ordenar o vetor.

Implemente o programa **Principal** no pacote *default*. Esse programa lê um número N (quantidade de sorteios) e uma sequência de 3 números digitada pelo usuário.
O programa imprime os N sorteios e compara a sequência do usuário com os números sorteados. Ao final, imprime se o usuário ganhou ou não (ou seja, se acertou ao menos um dos sorteios).

Exercício 4

Neste exercício, vamos criar uma classe para gerenciar números racionais. Para isso, iremos criar um pacote, chamado **racional**, que possui a classe **NumeroRacional**. A classe **NumeroRacional** representa um número racional (com numerador e denominador). Os números devem ser armazenados em sua forma simplificada. Por exemplo, o número 4/8 deve ser armazenado como 1/2. Esta classe contém:

- construtor que recebe como argumento dois números representando seu numerador e denominador, respectivamente. Se o denominador for 0, imprima uma mensagem de erro na tela.
- construtor padrão/*default* que inicializa o número com numerador e denominador aleatórios (pode usar a classe **NumeroAleatorio** do Exercício 3, por exemplo).
- método **simplificar** (privado) que encontra o máximo divisor comum e simplifica a representação do número. Se o numerador for zero, faça o denominador valer 1.
- método **soma** que recebe como argumento um número racional e faz a soma com a instância receptora.
- método **toString** que retorna uma string representando o número racional da seguinte forma: "numerador/denominador".

Lembre-se de sempre manter o número em sua forma simplificada. Implemente também uma classe **Principal** no pacote *default* para criar instâncias da classe **NumeroRacional** e testar seus métodos.

Exercício 5

Dando continuidade ao exercício anterior, agora vamos criar a classe **MatrizRacional** no pacote racional, que representa uma matriz N por M, em que cada elemento é um número racional. Esta classe contém:

- construtor padrão/*default* que cria uma matriz de tamanho 3x3 com números racionais inicializados com valores aleatórios.
- construtor que recebe como argumento números N e M e cria uma matriz N por M com números inicializados com valores zero.
- método **soma** que recebe como argumento uma matriz racional de mesma dimensão e soma seus elementos aos elementos do objeto receptor (importante: apresentar uma mensagem de erro esse método for chamado com matriz de tamanho inválido)
- método **imprimir** que imprime a matriz na tela

O programa principal deve ser criado fora do pacote racional. Neste programa, matrizes de números racionais devem ser instanciadas. Teste os métodos implementados fazendo soma entre as matrizes e imprimindo as matrizes e o resultado na tela.