**Universidade Federal de Campina Grande – UFCG**

**Centro de Engenharia Elétrica e Informática – CEEI**

**Unidade Acadêmica de Sistemas e Computação – UASC**

Disciplina: Programação 2 e Laboratório de Programação 2

**Lista de Exercícios - Java Procedural**

Crie código Java para as questões abaixo. Muitas delas você já deve ter visto e implementado em Programação 1. Chegou a hora de faze-las em Java. :)

Lembre de organizar seu código em pacotes para facilitar o reuso e restringir as visibilidades dos membros de suas classes. Faça a criação de métodos estáticos em uma só classe e chame esses métodos no seu método main.

**1.** Crie um método clicouNoRetangulo(x1, y1, x2, y2, x3, y3) que recebe três pontos, onde x1,y1 e x2,y2 são os pontos utilizados para construir o retângulo (definem canto superior esquerdo e canto inferior direito, mas não necessariamente respectivamente) e x3,y3 é o ponto em que o usuário clicou. A função deve retornar True se o ponto clicado estiver dentro do retângulo (inclusive nas bordas) e False caso contrário.

**2.** Uma empresa aérea pretende mudar o serviço de embarque de seus passageiros em seus vôos e decidiu que os passageiros serão chamados a embarcar de acordo com uma “fila virtual” na qual os passageiros são colocados, de acordo com a ordem de aquisição das passagens. Contudo, a empresa quer priorizar o embarque de pessoas idosas (acima de 60 anos). No momento do embarque, a empresa chama as pessoas idosas antes dos demais. Embora os idosos sejam priorizados, a empresa quer manter os idosos ma mesma ordem de entrada na fila.

Pede-se que você implemente um método que, dada um array das idades dos passageiros (ordenada segundo a fila de embarque), retorne uma outro array contendo apenas os índices dos passageiros idosos. Veja o assert abaixo para compreender a especificação:

fila = [25, 33, 67, 61, 35, 8, 12, 15, 22, 63, 75, 30, 34]

assert indices\_de\_idosos(fila) == [2, 3, 9, 10]

**3.** Escreva um método que receba um array de inteiros e verifica se existe a seqüência "1,..., 2, ..., 3". Observe que os três elementos não precisam ser vizinhos mas devem aparecer nesta ordem. Caso afirmativo, a função deve retornar o índice do número "1" na primeira ocorrência da sequencia. Caso não haja a sequência no array, a função deve retornar -1. Veja os asserts a seguir:

assert tem123plus([3,2,1,2,3]) == 2

assert tem123plus([4,1,1,1,4,2,3]) == 1

assert tem123plus([1,2,2,3]) == 0

assert tem123plus([1,2,2,4]) == -1

4. A principal unidade de comprimento é o metro. Entretanto, existem situações em que essa unidade deixa de ser prática. Por exemplo, se queremos medir a distância entre João Pessoa e Campina Grande, a unidade metro é muito pequena. Por outro lado, se queremos medir o comprimento de uma formiga, a unidade metro é muito grande. Nesses casos, para facilitar, utilizamos os múltiplos e submúltiplos das unidades de medidas. Os múltiplos e submúltiplos do metro, no Sistema Internacional de Medidas (SI), são apresentados na tabela a seguir:

|  |  |  |
| --- | --- | --- |
| **Fator** | **Símbolo** | **Nome** |
| 1000 | km | quilômetro |
| 100 | hm | hectômetro |
| 10 | dam | decâmetro |
| 1 | m | metro |
| 0.1 | dm | decímetro |
| 0.01 | cm | centímetro |
| 0.001 | mm | milímetro |

Um cientista precisa fazer várias medições e soma-las duas a duas. Para ajudá-lo, você precisa desenvolver um programa que realize essa soma para o cientista. O seu programa deve usar os símbolos da tabela acima para realizar a soma de um conjunto de pares de valores medidos pelo cientista. O resultado da soma deve ser impresso na unidade de medida básica (o metro).

**Entrada**

Cada linha da entrada corresponde a um caso de teste diferente. Cada caso de teste é composto por quatro informações: um inteiro X e sua unidade de medida e mais um inteiro Y e sua unidade de medida, todos separados por um espaço em branco.

**Saída**

Para cada caso de teste da entrada o seu programa deve imprimir um número real Z que indique a soma X + Y juntamente com sua unidade de medida. O resultado de cada soma deve possuir duas casas decimais e estar na unidade de medida básica.O programa deve parar quando ambos os valores a somar forem iguais a zero.

|  |  |
| --- | --- |
| **Entrada** | **Saida** |
| 20 cm 1 m 2 km 80 cm 0 cm 0 m | 1.20 m 2000.80 m |

**5.** Escreva um método que multiplique uma matriz A (m x n) por uma matriz B (n x p), representadas como arrays de arrays de inteiros. A função não deve alterar as matrizes recebidas.

mA = [[0, 1, 2], [1, 1, 3]]

mB = [[2, 1], [2, 2], [0,3]]

assert mult\_matrizes(mA, mB) == [[2, 8], [4, 12]]

**Dica:** Para usar matrizes em Java utilize outro par de colchetes na declaração e inicialização de arrays. Por exemplo, considerando a matriz 3x2 da entrada acima, teriamos:

int[][] mB = new int[3][2];

//int[][] mB = {{2,1},{2,2},{0,3}}; <-- Definir diretamente os valores.

matriz[2][1] = 3;

**6.** Escreva o método **contaPalavras(k, palavras)** que receba como parâmetro um valor inteiro k e uma string com uma série de palavras separadas por ":". A função deve retornar a quantidade de palavras com comprimento maior ou igual a k. Considere a existência de pelo menos uma palavra na série de palavras informada e k >= 0. As palavras não são acentuadas nem têm caracteres especiais.

assert conta\_palavras(5, "zero:um:dois:tres:quatro:cinco") == 2

Dica: Procure sobre como usar o método **split** da String. Por exemplo:

String[] palavras = stringOriginal.split(“stringSeparadora”)

**7.** Escreva um programa que identifica os alunos que resolveram um número de exercícios acima da média da turma. Inicialmente, escreva um método acimaDe(N, L) que recebe um valor numérico N e um array de números L e que retorna um array de índices de L, correspondendo aos elementos de L maiores que N.

No final, o programa deve ler da primeira linha da entrada uma sequência de números de exercícios resolvidos pelos alunos. Da segunda linha, os nomes dos alunos. Na saída imprime apenas os que estão acima da média da turma. Você não deve usar sum(), max(), min() ou sort(). Você DEVE usar a função acimaDe(N, L).

|  |  |
| --- | --- |
| **Entrada** | **Saida** |
| 10 12 20 16 15 19 22 a b c d e f g | c 20 f 19 g 22 |

**8**. [TOPPL1-20122] Uma sequência de DNA é uma série de pelo menos quatro letras representando a estrutura primária de uma molécula ou cadeia de DNA. As letras possíveis são A, C, G e T, representando os quatro nucleotídeos existentes. Um dos objetivos de um projeto de mapeamento genético é encontrar ocorrências de uma pequena cadeia de DNA dentro de uma outra cadeia maior. Escreva um programa que indica as posições em que uma cadeia menor ocorre dentro de uma cadeia maior. Para facilitar, utilize um **String[]**, em que cada String é um nucleotídeo da cadeia. Se preferir, pode usar **char[]**.

**Entrada**

A entrada é composta de duas linhas: (1) a primeira linha de entrada contém a primeira cadeia de DNA de tamanho N cujos nucleotídeos são separados por espaços; (2) a segunda linha da entrada contém a segunda cadeia de DNA de tamanho M (onde M >= N).

**Saída**

Seu programa deve imprimir em linhas separadas as posições em que a primeira cadeia de DNA ocorre dentro da segunda cadeia. Veja um exemplo de execução abaixo e observe que a posição inicial da cadeia é 1:

|  |  |
| --- | --- |
| **Entrada** | **Saida** |
| C A G T A T C A G T T C C A G T | 3 9 |