Exercícios: Operações sobre arrays

1. Após a execução da linha char[] alfabeto = {’A’, ’B’, ’C’, ’D’, ’E’, ’F’, ’G’, ’H’, ’I’}; em um método qualquer, responda:
   1. Quantas posições tem o array?

R: 9

* 1. Qual índice e valor da posição 4?

R: D e 3 😊

1. Qual será o conteúdo dos arrays declarados na aplicação abaixo ao término da execução do método main?

public class Arrays1

{

public static void main(String[] argumentos)

{

double[] valores = {1,2,3,4,5,6};

double[] primeiraCópia = valores;

double[] segundaCópia = valores;

primeiraCópia[1] = 1;

segundaCópia[2] = valores[0]+primeiraCópia[1];

primeiraCópia[3] = valores[1]+segundaCópia[2];

valores[4] = primeiraCópia[2]+segundaCópia[3];

valores[5] = segundaCópia[3]+primeiraCópia[4];

}

}

double[] valores = { 1, 2, 3, 4, 7, 9 };

double[] primeiraCopia = { 1, 1, 3, 4, 5, 6 };

double[] segundaCopia = { 1, 2, 2, 4, 5, 6 };

1. Qual será o conteúdo dos arrays declarados na aplicação abaixo ao término da execução do método main?

public class ArrayDeFloats

{

public static void main(String[] argumentos)

{

float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};

float[] duplicata = constantes;

resetaArray(duplicata);

}

private static void resetaArray(float[] array)

{

for(int índice=0;índice<array.length;índice++)

array[índice] = 0f;

}

}

Variável constantes começa com:

float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};

duplicate termina assim:

float[] duplicate = { 0f, 0f, 0f, 0f, 0f, 0f };

1. O método abaixo pode ser um método da classe ArraysDeFloats (exercício 3)? Explique

public void mudaTamanho(int novoTamanho)

{

array.length = novoTamanho;

}

Não, porque arrays são imutáveis em tamanho. O tamanho deles podem ser alterado usando métodos como .pop(), que remove 1 índice do array ou .add() que adiciona os valores ao array de destino.

1. Escreva uma aplicação em Java que declare e inicialize um vetor de booleanos (lendo-os do teclado, onde 0 é falso e qualquer outro valor é true), e depois calcule quantos elementos são iguais a true.

import java.util.Scanner;  
import java.util.Vector;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Scanner userInput = new Scanner(System.*in*);  
 Vector<Boolean> vetorBooleano = new Vector<>();  
 String entradaUsuario;  
 int contador = 0;  
  
 for (int i = 0; i <= 10; i++) {  
 entradaUsuario = userInput.nextLine();  
  
 if (entradaUsuario.equals("0")){  
 vetorBooleano.add(false);  
 }  
 else {  
 vetorBooleano.add(true);  
 contador++;  
 }  
 }  
  
 System.*out*.println(vetorBooleano);  
 System.*out*.println("O numero de True eh: " + contador);  
 }  
}

1. Escreva uma classe em Java que encapsule um array de 12 bytes, onde cada elemento do array contém o número de dias no mês correspondente, desconsiderando se o ano é bissexto ou não. Por exemplo, o elemento 0 do array (correspondente a janeiro) deve valer 31. Escreva um método que retorne o valor encapsulado para determinado mês.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner userInput = new Scanner(System.*in*);  
 String entradaUsuario = userInput.nextLine();  
  
 System.*out*.println("O numero de dias no mes escolhido eh " + *diasDoMes*(entradaUsuario));  
 }  
  
 private static int diasDoMes(String userInput) {  
 int[] ano = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
 return ano[Integer.*parseInt*(userInput)];  
 }  
}

1. Escreva um programa que declare um array bidimensional chamado tabuada de 10 × 10 posições e preencha os elementos do array com os valores da tabuada da soma para aquele elemento, de forma que, por exemplo, o elemento tabuada[7][9] valha 16. Use o tipo de dado mais adequado para este array.

public class Main {  
 public static void main(String[] args) {  
 int[][] tabuada = new int[10][10];  
  
 for (int i = 0; i < 10; i++){  
 for (int j = 0; j < 10; j++){  
 tabuada[i][j] = i + j;  
 }  
 }  
 }  
}

1. Crie na classe ArraysDeFloats (exercício 3) um método eIgual que receba outra instância da classe como argumento e retorne true se os tamanhos e valores do array encapsulado e do passado como argumento forem iguais.

class ArrayDeFloats {  
 public static void main(String[] argumentos) {  
 float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};  
 float[] duplicata = constantes;  
 *resetaArray*(duplicata);  
 }  
  
 private static void resetaArray(float[] array) {  
 for(int índice=0;índice<array.length;índice++)  
 array[índice] = 0f;  
 }  
  
 private static void metodoElgual(ArrayDeFloats arrayDeFloats){  
 return;  
 }  
}

1. Crie na classe ArrayDeFloats (exercício 3) um método soma que some (acumule) a cada um dos elementos do array encapsulado uma constante do tipo float que será passada como argumento.

public class Main {  
 public static void main(String[] argumentos) {  
 float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};  
 float[] duplicata = constantes;  
  
 System.*out*.println("Total de vetor constantes igual a " + *soma*(constantes));  
 *resetaArray*(duplicata);  
 }  
  
 private static void resetaArray(float[] array) {  
 for(int índice=0;índice<array.length;índice++)  
 array[índice] = 0f;  
 }  
  
 private static float soma(float[] array) {  
 float total = 0;  
  
 for (int i = 0; i < array.length; i++){  
 total += array[i];  
 }  
  
 return total;  
 }  
}

1. Crie na classe ArrayDeFloats (exercício 3) o método existe, que recebe um valor do tipo float como argumento e retorna o booleano true se o valor passado como argumento existir no array encapsulado.

public class Main {  
 public static void main(String[] argumentos) {  
 float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};  
 float[] duplicata = constantes;  
  
 System.*out*.println("Total de vetor constantes igual a " + *soma*(constantes));  
 System.*out*.println("O numero escolhido existe? " + *existe*(110f, constantes));  
 *resetaArray*(duplicata);  
 }  
  
 private static void resetaArray(float[] array) {  
 for(int índice=0;índice<array.length;índice++)  
 array[índice] = 0f;  
 }  
  
 private static float soma(float[] array) {  
 float total = 0;  
  
 for (int i = 0; i < array.length; i++){  
 total += array[i];  
 }  
  
 return total;  
 }  
  
 private static boolean existe(float numero, float[] array) {  
 for (int i = 0; i < array.length; i++) {  
 if (numero == array[i]) {  
 return true;  
 }  
 }  
  
 return false;  
 }  
}

1. Escreva uma classe que encapsule uma matriz de tamanho 2 × 2 de valores do tipo float ou double, usando um array de duas dimensões. Nessa classe, escreva um método que calcule o determinante da matriz encapsulada e um método que permita a impressão formatada dos seus valores (duas linhas com dois valores cada).

Dica: Dada matriz M, o o determinante é calculado como (x00 × x11) − (x01 × x10).

public class Main {  
 public static void main(String[] argumentos) {  
 double[][] matrix = {{1.0, 2.0}, {3.0, 4.0}};  
 double valor;  
  
 valor = *determinanteMatrix*(matrix);  
  
 System.*out*.println("Determinante: " + valor);  
 }  
  
 private static double determinanteMatrix(double[][] matrix) {  
 return (matrix[0][0] \* matrix[1][1]) - (matrix[0][1] \* matrix[1][0]);  
 }  
}

1. Crie uma classe que implementa um array de escrita única de valores de um tipo numérico qualquer. Um array de escrita única é um array cujos elementos só possam ser modificados uma única vez. Para a implementação, devemos ter, para cada elemento do array, um valor booleano associado que diz se o elemento pode ser modificado ou não. Quando instâncias dessa classe forem criadas, todos os elementos do array poderão ser modificados, mas assim que um elemento for modificado pela primeira vez o seu valor booleano associado será modificado de forma que da próxima vez que o elemento for modificado nada ocorrerá. Crie também uma aplicação que demonstre o uso desta classe.

public class Main {  
 public static void main(String[] argumentos) {  
 double[] array = {0, 1, 2 };  
 ImmutableArray immutableArray = new ImmutableArray();  
  
 System.*out*.println(immutableArray.modificaArray(array));  
 System.*out*.println(immutableArray.modificaArray(array));  
 }  
}

public class ImmutableArray {  
 double[] array = new double[3];  
 boolean verificacao = false;  
  
  
 public String modificaArray(double[] entrada) {  
 if(!verificacao){  
 array = entrada;  
 verificacao = true;  
  
 return "O array foi modificado";  
 }  
 else{  
 return "Voce nao tem permissao para modificar esse array";  
 }  
 }  
}

1. Usando o exercício anterior como base, crie uma classe que implemente um array cujos elementos só podem ser modificados um certo número limitado de vezes. Quando instâncias dessa classe forem criadas, elas devem receber um valor que diz quantas vezes um dado elemento do array encapsulado pode ser modificado. Dica: O valor deve ser o mesmo para todos os elementos do array, mas cada elemento individual do array poderá ser modificado o número de vezes que for especificado.

public class Main {  
 public static void main(String[] argumentos) {  
 double[] array = {0, 1, 2 };  
 ImmutableArray immutableArray = new ImmutableArray();  
  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 System.*out*.println(immutableArray.modificaArray(array, 5));  
 }  
}

public class ImmutableArray {  
 double[] array = new double[3];  
 boolean verificacao = false;  
 int contador = 0;  
  
 public String modificaArray(double[] entrada, int trocaMaxima) {  
 if(!verificacao){  
 array = entrada;  
 contador++;  
  
 if (contador == trocaMaxima) {  
 verificacao = true;  
 }  
  
 return "O array foi modificado";  
 }  
 else{  
 return "Voce nao tem permissao para modificar esse array";  
 }  
 }  
}

1. Escreva na classe ArrayDeFloats o método reverte, que reverte a ordem dos elementos do array encapsulado, de forma que o primeiro passe a ser o último e vice-versa. Por exemplo, se o array encapsulado for {9, 9, 2, 7, 0, 5}, depois da execução do método ele será {5, 0, 7, 2, 9, 9}. Dica: Existem duas abordagens para a solução desse problema, uma que modifica os valores do array encapsulado e outra que cria uma nova instância; considere implementar as duas.

public class Main {  
 public static void main(String[] argumentos) {  
 float[] constantes = {100f, 10f, 1f, 0.1f, 0.01f, 0.001f};  
 float[] duplicata = constantes;  
  
 *reversoDoArray*(constantes);  
 *resetaArray*(duplicata);  
 }  
  
 private static float[] reversoDoArray(float[] array) {  
 float[] arrayReverso = new float[array.length];  
 int j = array.length;  
  
 for(int i = 0; i < array.length; i++){  
 arrayReverso[j - 1] = array[i];  
 j -= 1;  
 }  
  
 return arrayReverso;  
 }  
  
 private static void resetaArray(float[] array) {  
 for (int índice = 0; índice < array.length; índice++)  
 array[índice] = 0f;  
 }  
  
}

1. Escreva para a classe ArrayDeFloats o método éPalíndromo, que retorna true se o array encapsulado for palíndromo. Um array palíndromo é aquele que pode ser lido do início para o fim e do fim para o início, da mesma forma. Por exemplo, o array (2, −4, 9, 0, 9, −4, 2) é palíndromo. Dica: Existem ao menos duas soluções para o problema, uma que verifica os dados do array, comparando-os, e outra que usa a solução do exercício anterior.

public class Main {  
 public static void main(String[] argumentos) {  
 float[] constantes = {100f, 10f, 1f, 0.1f, 0.01f, 0.001f};  
 float[] palindromo = {100f, 10f, 1f, 0f, 1f, 10f, 100f};  
 float[] duplicata = constantes;  
  
 *ePalindromo*(constantes);  
 *ePalindromo*(palindromo);  
   
 *reversoDoArray*(constantes);  
   
 *resetaArray*(duplicata);  
 }  
  
 private static boolean ePalindromo(float[] array) {  
 float[] arrayReverso = new float[array.length];  
 int j = array.length;  
  
 for(int i = 0; i < array.length; i++){  
 arrayReverso[j - 1] = array[i];  
 j -= 1;  
 }  
  
 return Arrays.*equals*(arrayReverso, array);  
 }  
  
 private static float[] reversoDoArray(float[] array) {  
 float[] arrayReverso = new float[array.length];  
 int j = array.length;  
  
 for(int i = 0; i < array.length; i++){  
 arrayReverso[j - 1] = array[i];  
 j -= 1;  
 }  
  
 return arrayReverso;  
 }  
  
 private static void resetaArray(float[] array) {  
 for (int índice = 0; índice < array.length; índice++)  
 array[índice] = 0f;  
 }  
  
  
}

1. Crie uma classe que represente um jogo da velha, usando uma matriz de duas dimensões para representar as posições do jogo. A matriz deve ser alocada no construtor da classe, ter o tamanho 3 × 3 e ser de um tipo que suporte três estados possíveis: vazio, preenchido com ’O’ e preenchido com ’X’ (use um enumerador). A classe deve poder ser usada para jogos com dois jogadores. Dica: A classe deve ter os seguintes métodos:
   1. jogaO, que aceita dois valores que são as coordenadas onde um ’O’ será jogado, e marca na matriz a posição somente se esta estiver livre.
   2. jogaX, que aceita dois valores que são as coordenadas onde um ’X’ será jogado, e marca na matriz a posição somente se esta estiver livre.
   3. verifica, que verifica a matriz para ver se existe algum ganhador (esse método deve verificar se existem três marcas iguais que não sejam vazias em uma horizontal, vertical ou diagonal da matriz).
   4. toString, que retornará uma string com a representação gráfica do jogo com as posições atuais.

Escreva também um programa que use a classe. Este programa deve executar um laço no qual fica perguntando as posições para os jogadores alternadamente, enquanto não houver vitória, desistência ou acabarem as posições vazias da matriz.

1. Um quadrado mágico é uma matriz quadrada de valores inteiros onde a soma dos valores em cada linha, coluna ou diagonal principal é a mesma, e não existem elementos repetidos. Escreva uma classe QuadradoMagico que tenha o método estático éQuadradoMágico que retorne true caso a matriz, passada como argumento para o método, represente um quadrado mágico.
2. O crivo de Eratóstenes é um algoritmo usado para identificar números primos. O algoritmo (apresentado aqui da maneira mais simples) primeiro declara um array de N posições de valores booleanos, todos iguais a true (considerando que em princípio qualquer número pode ser primo). O algoritmo, em seguida, marca todos os elementos do array cujos índices são múltiplos de 2 e maiores que o próprio 2 como false, indicando que nenhum múltiplo de dois pode ser primo. O algoritmo repete esse último procedimento para todos os valores múltiplos de 3 e maiores que 3, depois para todos os valores múltiplos de 4 e maiores que 4, e assim sucessivamente, até chegar até N/2. Ao final, os índices dos elementos do array que valerem false serão valores não-primos, e os que ainda valerem true depois da execução do algoritmo serão primos. Dica: Para entender melhor o algoritmo, rode uma simulação em papel antes. Esse algoritmo pode ser consideravelmente otimizado, tente fazer isso.
3. O algoritmo de ordenação de bolha (bubblesort) serve para colocar os elementos de uma lista em ordem crescente ou decrescente. O algoritmo é como segue:
   1. Percorra o array da primeira posição até a penúltima, chamando o elemento do array nessa posição de P;
   2. Para cada P percorra o mesmo array da posição seguinte de P até a última posição do array, chamando o elemento nessa posição de Q.
   3. Se P > Q, troque os valores de P e Q.

Uma simulação simples do algoritmo é mostrada abaixo. Consideremos o array {7, 5, 1, 3} que deve ser ordenado de forma crescente. Os passos da simulação seriam:

1. O array vale, inicialmente, {7, 5, 1, 3}.
2. A posição de P no array é 0, e seu valor é 7. A posição de Q no array é 1, e seu valor é 5. Como P > Q, os valores são trocados. Agora o array é {5, 7, 1, 3}.
3. A posição de P no array ainda é 0, e seu valor é 5. A posição de Q no array é 2, e seu valor é 1. Como P > Q, os valores são trocados. Agora o array é {1, 7, 5, 3}.
4. A posição de P no array ainda é 0, e seu valor é 1. A posição de Q no array é 3, e seu valor é 3. Como P não é maior que Q, os valores não são trocados. O array permanece sendo {1, 7, 5, 3}.
5. A posição de P no array agora é 1 (próxima iteração do laço externo), e seu valor é 7. A posição de Q no array é 2, e seu valor é 5. Como P > Q, os valores são trocados. O array agora é {1, 5, 7, 3}.
6. A posição de P no array ainda é 1, e seu valor é 5. A posição de Q no array é 3, e seu valor é 3. Como P > Q, os valores são trocados. Agora o array é {1, 3, 7, 5}.
7. A posição de P no array agora é 2 (próxima iteração do laço externo), e seu valor é 7. A posição de Q no array é 3, e seu valor é 5. Como P > Q, os valores são trocados. Agora o array é {1, 3, 5, 7}.
8. O array vale, finalmente, {1, 3, 5, 7}

Implemente na classe ArrayDeFloats (listagem 10.5 no livro) o método ordenaCrescentePorBolha que implemente o algoritmo descrito. Crie também o método ordenaDecrescentePorBolha que implemente um algoritmo que ordene de forma decrescente: a única diferença do algoritmo que ordena de forma crescente é que P e Q deverão ser trocados se P < Q. De que outra forma podemos resolver este segundo problema?

1. O jogo japonês Gomoku é jogado por duas pessoas em um tabuleiro quadrado de tamanho 19×19. Cada pessoa recebe um conjunto de peças pretas e brancas que devem ser colocadas alternadamente no tabuleiro, na posição que o jogador desejar. Ganha o jogo o primeiro jogador que conseguir colocar cinco de suas peças em uma linha reta horizontal, vertical ou diagonal. Crie uma classe em Java que represente um jogo de Gomoku, usando uma matriz de duas dimensões para representar as posições do jogo. A matriz deve ser alocada no construtor da classe, ter o tamanho 19 × 19 e ser de um tipo que suporte três estados possíveis: vazio, preenchido com peça preta e preenchido com peça branca (use um enumerador para isto!). A classe deve poder ser usada para jogos com dois jogadores. A classe deve ter os seguintes métodos:

* jogaPreta, que aceita dois valores que são as coordenadas onde uma peça preta será jogada, e marca na matriz a posição somente se esta estiver livre.
* jogaBranca, que aceita dois valores que são as coordenadas onde uma peça branca será jogada, e marca na matriz a posição somente se esta estiver livre.
* verifica, que verifica a matriz para ver se existe algum ganhador (este método deve verificar se existem cinco peças iguais que não sejam vazias em uma horizontal, vertical ou diagonal da matriz, depois de cada jogada feita).
* toString, que retornará uma string com a representação gráfica do jogo com as posições atuais.

Escreva também um programa que use a classe. Esse programa deve executar um laço no qual fica perguntando as posições para os jogadores alternadamente, enquanto não houver vitória, desistência ou acabarem as posições vazias da matriz. Dica: O algoritmo do jogo não é tão diferente do jogo da velha (exercício 16), exceto pelo método verifica. Esse método pode, para cada posição do array bidimensional, ver se existem linhas de cinco peças iguais contadas a partir da posição sendo procurada. O único cuidado adicional é garantir que o algoritmo não procurará peças fora do tabuleiro.