EXERCÍCIOS

1. Carregar um vetor [100] real de valores de saldos. Calcular e exibir:
   * Média dos saldos positivos entre 100 à 1000
   * Média geral dos saldos
   * Soma dos saldos negativos

R:

Import javax.swing.JOptionPane;

public class ex1{

public static void main(String args[]){

double [] vet = new double [100];

double media1=0;

double div1 = 0;

double mediag=0;

double soma = 0;

for (int i=0; i<100; i++){

vet[i] = Double.parseDouble(JOptionPane.showInputDialog(“insira um valor:”));

if (vet[i]>100 && vet[i]<1000){

div1 ++;

media1+= vet[i];

}

if (vet[i]<0){

soma+=vet[i];

}

mediag+=vet[i];

}

mediag = mediag/100;

if (div1>0){

media1 = media1/div1;

}

soma = soma\*(-1);

JOptionPane.showMessageDialog(null,”Media1 = ” + media1 + “ Media geral = ” + mediag + “ soma dos negativos = ” + soma);

}

}

1. Carregar um vetor [5] inteiro. Enviar cada elemento para uma função e esta irá retornar o seu fatorial que será armazenado em um outro vetor. Exibir os dados dos vetores

R:

Import javax.swing.JOptionPane;

public class ex1{

public static void main(String args[]){

int [] vetor1 = new int [5];

int [] vetor2 = new int [5];

for (int i=0; i<5; i++){

vetor1[i] = Integer.parseInt("Insira um valor inteiro:");

vetor2[i] = func\_fat(vetor1[i]);

JOptionPane.showMessageDialog(null, "" + vetor2[i]);

}

}

public static int func\_fat (int x){

int valor;

if(x=1){

return x;

}

else{

valor = x \* func(x-1);

return valor;

}

}

}

1. Carregar um vetor [100] inteiros positivos ou negativos. Classificar este vetor em ordem crescente e apresentar os valores.

R:

Import javax.swing.JOptionPane;

public class ex1{

public static void main(String args[]){

int [] vetor = new int [100];

int x;

for (int i=0; i<100; i++){

vetor[i] = Integer.parseInt(JOptionPane.showMessageDialog("Insira um valor inteiro"));

}

for(int i=0; i<100; i++){

for (int j=i+1; j<100; j++){

if (vetor[j]<vetor[i]){

x = vetor[i];

vetor[i] = vetor[j];

vetor[j] = x;

}

}

}

for (int i=0; i<100; i++){

JOptionPane.showMessageDialog(null,"" + vetor[i]);

}

}

}

1. Carregar uma matriz [4 x 4 inteiro]. Apresentar:
   * Soma dos valores no intervalo de 1 a 100
   * Quantidade de números ímpares entre 30 a 50
   * Quantidade de números divisíveis por 8
   * Quantidade de números ímpares divisíveis por 3
   * Fatorial do maior número informado na matriz

R:

import javax.swing.JOptionPane;

public class ex1{

public static void main(String args[]){

int [] matriz = new int [4][4];

int qt1=0;

int qt2=0;

int qt3=0;

int fat;

int maior = 0;

for (int i=0; i<4; i++){

for (int j=0; j<4; j++){

matriz [i][j] = Integer.parseInt("Insira um valor inteiro:");

if (matriz[i][j]>30 && matriz[i][j]<50){

if(matriz[i][j]%2!=0){

qt1++;

}

}

if (matriz[i][j]%8=0){

qt2++;

}

if(matriz[i][j]%3=0){

qt3++;

}

if(matriz[i][j]>maior){

maior = matriz[i][j];

}

}

}

fat = func\_fat(maior);

for(int i=0; i<4; i++){

for(int j=0; j<4; j++){

JOptionPane.showMessageDialog(null, "" + qt1 + " " + qt2 + " " + qt3 + " " + fat);

}

}

}

public static int func\_fat (int x){

int valor;

if(x=1){

return x;

}

else{

valor = x \* func(x-1);

return valor;

}

}

}

1. Criar e carregar uma matriz [4 x 4] inteiro, onde os valores da diagonal principal serão carregados pela aplicação conforme o gráfico e os demais dados serão digitados pelo usuário

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  |  |
|  | 3 |  |  |
|  |  | 9 |  |
|  |  |  | 27 |

R:

import javax.swing.JOptionPane;

public class ex1{

public static void main(String args[]){

int [] mat = new int [4][4];

int e=0;

for (int i=0; i<4; i++){

for (int j=0; j<4; j++){

if(i=j){

mat[i][j] = Math.pow(3, e);

e++

}

else{

mat[i][j] = Integer.parseInt(JOptionPane.showInputDialog("Insira um numero:"));

}

JOptionPane.showMessageDialog(null, "" + mat[i][j]);

}

}

}

}

6) Simule o método a seguir, demonstrando cada passo em uma representação gráfica de lista:

# public void metodoA (int pos, int valor)

{

if (dados.length == tamanho)

System.out.println("ERRO!”);

else { if (pos <= 0 || pos > tamanho+1)

System.out.println("Posição Inválida!");

else { if (pos = = 1)

adicionaInicio(valor);

# else

if (pos = = tamanho)

adicionaFinal(valor);

# else{

for (int i = tamanho; i >= pos; i--) dados[i]=dados[i-1];

dados[pos-1]=valor; tamanho++;

}

}

}

}

R:

# public void metodoA (int pos, int valor)

{

if (dados.length == tamanho) => se a lista estiver cheia, ele retorna erro!

System.out.println("ERRO!”);

else { if (pos <= 0 || pos > tamanho+1) => se a posição exceeder a lista, ele retorna erro!

System.out.println("Posição Inválida!");

else { if (pos = = 1) => se a posição for 1 ele adiciona o valor ao inicio

adicionaInicio(valor);

# else

if (pos = = tamanho) => se a posição for o tamanho ele adiciona ao final da lista

adicionaFinal(valor);

# else{ => se for algum valor do meio, ele empurra os demais para encaixar o valor

for (int i = tamanho; i >= pos; i--) dados[i]=dados[i-1];

dados[pos-1]=valor; tamanho++;

} => o tamanho da lista aumenta

}

}

}

1. Implemente na classe Lista um método que remove o elemento de uma posição específica recebida por parâmetro.

R:

public void Remover(int x){

if (x>=dados.length){

System.out.println("Posição invalida!");

}

else{

dados[x] = 0;

tamanho --;

}

}

1. Implemente uma Lista de caracteres em alocação estática com todas as operações indicadas a seguir:

-verificar se a lista está vazia, retornando true se estiver vazia e false se não estiver;

-verificar se a lista está cheia, retornando true se estiver cheia e false se não estiver;

-adicionar caractere no início da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar caractere no final da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar caractere em determinada posição da lista, caso a operação não possa ser realizada, mostre mensagem avisando o motivo;

-remover caractere do início da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover caractere do final da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover caractere de determinada posição da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-percorrer a lista concatenando os elementos em uma String que será devolvida no final.

R:

public class Lista{

private char [] dados;

private int tamanho;

public Lista(){

dados = new char [100];

tamanho = 0;

}

public boolean Verifvazia(){

if (tamanho == 0){

return true;

}

else{

return false;

}

}

public boolean Verifcheia(){

if (dados.length == tamanho){

return true;

}

else{

return false;

}

}

public void AdicionarInicio(Lista x){

boolean verif = Verifcheia();

int y;

int z;

if (verif == true){

System.out.println("Lista está cheia");

}

else if (!Verifvazia() && dados[0]!=null){

dados[tamanho+1] = null;

y = dados[0];

for(int i=0; i<=tamanho+1;i++){

z = dados[i+1];

dados[i+1] = y;

y = z;

}

dados[0] = x;

tamanho++;

}

else{

dados[0] = x;

tamanho++;

}

}

public void AdicionarFinal(Lista x){

if(!Verifcheia()){

dados[tamanho] = x;

tamanho++;

}

else{

System.out.println("Lista está cheia");

}

}

public void Adiciona(int y; Lista x){

if(!Verifcheia()){

if(dados[y]!=null){

dados[y] = x;

tamanho ++;

}

else{

System.out.println("Posição possui um valor!");

}

}

else{

System.out.println("Lista está cheia!");

}

}

public Lista RemoveFinal(){

Lista x;

if (!Verifvazia()){

x = dados[tamanho-1];

dados[tamanho-1] = null;

tamanho--;

return x;

}

else{

System.out.println("Lista está vazia!");

}

}

public Lista RemoveInicio(){

Lista x;

if (!Verifvazia()){

x = dados[0];

dados[0] = null;

tamanho --;

return x;

}

else{

System.out.Println("Lista está vazia");

}

}

public Lista Remove(int i){

Lista x;

if (!Verifvazia()){

if(i>tamanho-1){

System.out.println("Valor fora do limite de tamanho");

}

else if(dados[i]==null){

System.out.println("Posição ja vazia!");

}

else{

x = dados[i];

dados[i] = null;

tamanho --;

return x;

}

}

else{

System.out.println("Lista está vazia!");

}

}

public String concatenar(){

String result = null;

for(int i=0; i<tamanho; i++){

result += dados[i];

}

return result;

}

}

9) Implemente uma Lista de Strings em alocação estática com todas as operações indicadas a seguir:

-verificar se a lista está vazia, retornando true se estiver vazia e false se não estiver;

-verificar se a lista está cheia, retornando true se estiver cheia e false se não estiver;

-adicionar uma String no início da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar uma String no final da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar uma String em determinada posição da lista, caso a operação não possa ser realizada, mostre mensagem avisando o motivo;

-remover a String do início da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover a String do final da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover a String de determinada posição da lista, retornando o elemento que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-percorrer a lista concatenando os elementos em uma String que será devolvida no final.

R:

public class Lista{

private String [] dados;

private int tamanho;

public Lista(){

dados = new String [100];

tamanho = 0;

}

public boolean Verifvazia(){

if (tamanho == 0){

return true;

}

else{

return false;

}

}

public boolean Verifcheia(){

if (dados.length == tamanho){

return true;

}

else{

return false;

}

}

public void AdicionarInicio(Lista x){

boolean verif = Verifcheia();

int y;

int z;

if (verif == true){

System.out.println("Lista está cheia");

}

else if (!Verifvazia() && dados[0]!=null){

dados[tamanho+1] = null;

y = dados[0];

for(int i=0; i<=tamanho+1;i++){

z = dados[i+1];

dados[i+1] = y;

y = z;

}

dados[0] = x;

tamanho++;

}

else{

dados[0] = x;

tamanho++;

}

}

public void AdicionarFinal(Lista x){

if(!Verifcheia()){

dados[tamanho] = x;

tamanho++;

}

else{

System.out.println("Lista está cheia");

}

}

public void Adiciona(int y; Lista x){

if(!Verifcheia()){

if(dados[y]!=null){

dados[y] = x;

tamanho ++;

}

else{

System.out.println("Posição possui um valor!");

}

}

else{

System.out.println("Lista está cheia!");

}

}

public Lista RemoveFinal(){

Lista x;

if (!Verifvazia()){

x = dados[tamanho-1];

dados[tamanho-1] = null;

tamanho--;

return x;

}

else{

System.out.println("Lista está vazia!");

}

}

public Lista RemoveInicio(){

Lista x;

if (!Verifvazia()){

x = dados[0];

dados[0] = null;

tamanho --;

return x;

}

else{

System.out.Println("Lista está vazia");

}

}

public Lista Remove(int i){

Lista x;

if (!Verifvazia()){

if(i>tamanho-1){

System.out.println("Valor fora do limite de tamanho");

}

else if(dados[i]==null){

System.out.println("Posição ja vazia!");

}

else{

x = dados[i];

dados[i] = null;

tamanho --;

return x;

}

}

else{

System.out.println("Lista está vazia!");

}

}

public String concatenar(){

String result = null;

for(int i=0; i<tamanho; i++){

result += dados[i];

}

return result;

}

}

10) Implemente uma Lista de temperaturas em alocação estática com todas as operações indicadas a

seguir:

-verificar se a lista está vazia, retornando true se estiver vazia e false se não estiver;

- verificar se a lista está cheia, retornando true se estiver cheia e false se não estiver;

-adicionar uma temperatura no início da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar uma temperatura no final da lista, caso a operação não possa ser realizada, mostre mensagem avisando;

-adicionar uma temperatura em determinada posição da lista, caso a operação não possa ser realizada, mostre mensagem avisando o motivo;

-remover a temperatura do início da lista, retornando o valor que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover a temperatura do final da lista, retornando o valor que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-remover a temperatura de determinada posição da lista, retornando o valor que foi removido, caso a operação não possa ser realizada, mostre mensagem avisando;

-percorrer a lista concatenando os elementos em uma String que será devolvida no final

R:

public class Lista{

private double [] dados;

private int tamanho;

public Lista(){

dados = new double [100];

tamanho = 0;

}

public boolean Verifvazia(){

if (tamanho == 0){

return true;

}

else{

return false;

}

}

public boolean Verifcheia(){

if (dados.length == tamanho){

return true;

}

else{

return false;

}

}

public void AdicionarInicio(Lista x){

boolean verif = Verifcheia();

int y;

int z;

if (verif == true){

System.out.println("Lista está cheia");

}

else if (!Verifvazia() && dados[0]!=null){

dados[tamanho+1] = null;

y = dados[0];

for(int i=0; i<=tamanho+1;i++){

z = dados[i+1];

dados[i+1] = y;

y = z;

}

dados[0] = x;

tamanho++;

}

else{

dados[0] = x;

tamanho++;

}

}

public void AdicionarFinal(Lista x){

if(!Verifcheia()){

dados[tamanho] = x;

tamanho++;

}

else{

System.out.println("Lista está cheia");

}

}

public void Adiciona(int y; Lista x){

if(!Verifcheia()){

if(dados[y]!=null){

dados[y] = x;

tamanho ++;

}

else{

System.out.println("Posição possui um valor!");

}

}

else{

System.out.println("Lista está cheia!");

}

}

public Lista RemoveFinal(){

Lista x;

if (!Verifvazia()){

x = dados[tamanho-1];

dados[tamanho-1] = null;

tamanho--;

return x;

}

else{

System.out.println("Lista está vazia!");

}

}

public Lista RemoveInicio(){

Lista x;

if (!Verifvazia()){

x = dados[0];

dados[0] = null;

tamanho --;

return x;

}

else{

System.out.Println("Lista está vazia");

}

}

public Lista Remove(int i){

Lista x;

if (!Verifvazia()){

if(i>tamanho-1){

System.out.println("Valor fora do limite de tamanho");

}

else if(dados[i]==null){

System.out.println("Posição ja vazia!");

}

else{

x = dados[i];

dados[i] = null;

tamanho --;

return x;

}

}

else{

System.out.println("Lista está vazia!");

}

}

public String concatenar(){

String result = null;

for(int i=0; i<tamanho; i++){

result += dados[i];

}

return result;

}

}

1. Elabore um resumo do artigo “A Study on Performance Analysis of Data Structures” disponível no link abaixo:

[https://www.academia.edu/37436288/A\_Study\_on\_Performance\_Analysis\_of\_Data\_Structures?auto=do wnload](https://www.academia.edu/37436288/A_Study_on_Performance_Analysis_of_Data_Structures?auto=download)

1. Descreva os requisitos funcionais, não funcionais e regras de negócio que pretende aplicar no projeto de Estrutura de Dados e faça um cronograma detalhado de tudo que pretende desenvolver no projeto.