**Estrutura de Dados**

1. (1.0) No projeto lista implemente um método de busca (search), de forma que a partir de um valor dado, retorne de o valor existe ou não (retornar true no caso de existente, e false no caso de não existente). public boolean search(int value);

**APP.Java**

import java.util.Scanner;

import javax.swing.JOptionPane;

public class App {

    static List list = new ArrayList();

    public static int inputInt(String message) {

        Scanner scanner = new Scanner(System.in);

        System.out.print(message);

        int value = scanner.nextInt();

        return value;

    }

    public static void outputText(String message) {

        System.out.println(message);

    }

    public static  void insertOp() {

        int num = inputInt("Informe o numero: ");

        list.add(num);

    }

    public static  void showOp() {

        outputText("mostrando... \n"+ list.show());

    }

    public static  void removeOp() {

        int num = inputInt("Informe o numero: ");

        list.remove(num);

    }

    public static  void updateOp() {

        int num = inputInt("Informe o numero: ");

        int pos = inputInt("Informe a posicao: ");

        list.update(pos, num);

    }

    public static void searchOp() {

        int num = inputInt("Informe o numero: ");

        System.out.println(list.search(num));

    }

    public static void main(String[] args) {

        boolean toBeContinue = true;

        while (toBeContinue) {

            int option = inputInt("Informe a opção desejada:"

                    + "\n 0 - Parar"

                    + "\n 1 - Inserir"

                    + "\n 2 - Listar"

                    + "\n 3 - Atualizar"

                    + "\n 4 - Buscar"

                    + "\n 5 - Remover\n>> ");

            if (option == 0) {

                toBeContinue = false;

                outputText("Teminando Aplicação...");

            } else if (option == 1) {

                insertOp();

            } else if (option == 2) {

                showOp();

            } else if (option == 3) {

                updateOp();

            } else if (option == 4) {

                searchOp();

            } else if (option == 5) {

                removeOp();

            } else {

                outputText("Opção inválida...");

            }

        }

    }

}

**ArrayList.Java**

public class ArrayList implements List {

    int[] data = new int[1000];

    int quantity = 0;

    public void add(int value) {

        data[quantity] = value;

        quantity++;

    }

    public boolean search(int value) {

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

            if (this.data[i] == value) {

                return true;

            }

        }

        return false;

    }

    public void remove(int value) {

        int foundIndex = -1;

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

            if (data[i] == value) {

                foundIndex = i;

                break;

            }

        }

        if (foundIndex == -1) {// se o indice continua -1, então não achou

            return;

        }

        //copia por cima do elemento que queremos remover

        int lastIndex = quantity - 1;

        for (int i = foundIndex; i < lastIndex; i++) {

            data[i] = data[i + 1];

        }

        quantity--;

    }

    public String show() {

        String output = "";

        int i = 0;

        while (i < quantity) {

            output = output + " " + data[i];

            i++;

        }

        return output;

    }

    public void update(int position, int value) {

        int index = position - 1;

        if (index < 0 || index >= quantity) {

            return;

        }

        this.data[index] = value;

    }

    public int size() {

        return quantity;

    }

}

**LinkedList.Java**

public class LinkedList implements List {

    Node head;

    @Override

    public void add(int value) {

    }

    @Override

    public boolean search(int value) {

        return false;

    }

    @Override

    public void remove(int value) {

    }

    @Override

    public String show() {

        return null;

    }

    @Override

    public void update(int position, int value) {

    }

    @Override

    public int size() {

        return 0;

    }

}

**List.Java**

public interface List {

    public void add(int value);

    public void remove(int value);

    public String show();

    public void update(int position, int value);

    public int size();

    public boolean search(int value);

}

**Node.Java**

public class Node {

    int content;

    Node next;

}

|  |
| --- |
|  |

1. (0,5) Explique os seguintes termos e operações, descrevendo o cenário de cada um.

As estruturas de dados FIFO e LIFO são duas das mais comumente utilizadas na computação devido à funcionalidade que implementam.  
**1- FIFO** - A pilha implementa o conceito de FILO (First-In, Last-Out) ou “Primeiro a Entrar, Último a Sair”. O último elemento a ser inserido na pilha é o primeiro a ser removido, enquanto o primeiro a ser inserido é o último que sai.

**Cenário:** Imagine que você deve organizar vários livros que encontram-se espalhados para depois distribuí-los ordenadamente. Primeiramente você juntará todos os livros em uma pilha, um sobre o outro.

Feito isso, você passará a distribuí-los um a um. O primeiro livro a ser removido da pilha para ser entregue é aquele que se encontra no topo, ou seja, o último que foi inserido.

A nossa classe Stack possuirá apenas três métodos:

**Inserir(obj)**: Insere o elemento recebido como parâmetro no final da lista.

**RemoverUltimo**(): Retorna o valor do último elemento inserido e o remove da lista. Este método é o mais utilizado nesse contexto, o próximo (LerUltimo) foi criado apenas como auxiliar.

**LerUltimo**(): Retorna o valor do último elemento inserido o do topo da pilha.

**2- FILO -** A fila, ou queue como também é conhecida é uma estrutura de dados que implementa o conceito de FIFO (First-In, First-Out) ou “Primeiro a Entrar, Primeiro a Sair”. Nesta estrutura temos dois métodos principais, um para inserir um item na fila e outro para ler e remover o primeiro elemento. Vale salientar que este segundo método sempre retornará o elemento de índice zero, ou seja, o primeiro.

**Cenário:** Essa situação é semelhante, por exemplo a uma fila de supermercado ou banco, onde o primeiro usuário a entrar na fila é também o primeiro a ser atendido e sair dela. Nesse caso, ilustrações são dispensadas pois trata-se de um conceito ainda mais simples que o anterior.

De forma complementar, a classe que será apresentada aqui terá um método a mais, que permitirá a leitura do primeiro elemento sem que este seja removido da coleção. Em suma, os métodos são os seguintes:

**Inserir(obj):** insere um item no final da fila.

**RemoverPrimeiro**: retorna o valor do primeiro elemento da fila, o de índice zero, e o remove.

**LerPrimeiro**: retorna o valor do primeiro elemento da lista, mas sem que este seja removido.

1. (1,5) Pesquise cenários de uso prático das seguintes estruturas de dados

**1- Pilha** - São estruturas de dados que armazenam os elementos em um formato sequencial, empilhando um item acima do outro (imagine uma pilha de pratos, por exemplo). Estas estruturas permitem “empilhar” os itens que serão armazenados e “desempilhar” estes elementos da pilha quando precisarmos removê-lo. Sempre que um novo elemento é inserido (ou empilhado) damos a ele o nome de “topo”, pois é o primeiro elemento ao qual teremos acesso. A Pilha é uma estrutura de dados muito comum em sistemas computacionais. Dentre as várias soluções possíveis que a pilha permite que podemos citar: Inversão de listas, Armazenar dados e Implementar LIFOs.

**Cenário:** A utilização dessa estrutura é natural onde temos cenários que a última operação empilhada seja a primeira a ser processada, em outras palavras, as mais recentes primeiro. Um exemplo prático é a utilização em logística de produtos que não tem vencimento, isso traz algumas vantagens para essas empresas pois geralmente as empresas que operam com esse modelo de negócio precificam a mercadoria com base na última compra, isso faz com que os produtos mais antigos em estoque que porventura fossem comprados com custo menor tenham margem de lucro maior, pois serão vendidos com preço de venda maior baseado na compra mais recente.

**Código:** É uma implementação simples e efetiva de uma estrutura de dados do tipo pilha.

using System;

public class Stack<T>

{

private StackItem<T> topo;

private bool isPilhaVazia

{

get { return this.Count == 0;

}

}

public int Count;

// Método que empilha

public void Push(T item)

{

if (isPilhaVazia)

this.topo = new StackItem<T>(item);

else

{

var stackItem = new StackItem<T>(topo, item); antigo this.topo = stackItem;

}

this.Count++;

}

// Método que desempilha

public T Pop()

{

if (isPilhaVazia)

throw new IndexOutOfRangeException();

else

{

T item = this.topo.item;

this.topo = this.topo.anterior;

this.Count--;

return item;

}

}

//Classe de suporte as operações da pilha, responsável por vincular os elementos e armazenar valores

private class StackItem<T>

{

public T item;

public StackItem<T> anterior;

public StackItem(T item)

{

// Constructor de uma pilha vazia

this.item = item;

}

public StackItem(StackItem<T> anterior, T item)

{

// Constructor de uma pilha com elementos

this.anterior = anterior;

this.item = item;

}

}

}

**2- Fila -** São estruturas de dados bastante utilizadas na computação, onde o primeiro elemento a ser inserido, será também o primeiro a ser retirado. Desta forma, serão adicionados elementos no fim e removê-los pelo início. A estrutura de dados fila segue um padrão conhecido como **FIFO** (first-in first-out), onde o primeiro a entrar é o primeiro a sair.

**Cenário:** Imagine o exemplo de um fila do banco, onde diversos usuários a compõe em uma manhã de sexta-feira. A senhora que chegou a fila às 7 da manhã se posiciona na fila. Como ela foi a primeira a chegar, estará no início da fila, a partir daí todos os usuários do banco que chegarem terão que se posicionar depois da senhora. Ao iniciar o atendimento, o primeiro usuário (início da fila) será atendido. Ao fazer isso, este sairá da fila e o elemento (usuário) que estava atrás deste passará a ser o início da fila. Este comportamento seguirá até que nenhum outro usuário se posicione ao final da fila e todos os usuários da fila sejam atendidos, tornando-a vazia.

**Código:** É uma implementação de uma fila com 100 elementos.

import javax.swing.\*;

class Fila{

int inicio;

int fim;

int tamanho;

int qtdeElementos;

int f[];

public Fila(){

inicio = fim = -1;

tamanho = 100;

f = new int[tamanho];

qtdeElementos = 0;

}

public boolean estaVazia(){

if (qtdeElementos == 0){

return true;

}

return false;

}

public boolean estaCheia(){

if (qtdeElementos == tamanho - 1){

return true;

}

return false;

}

public void adicionar(int e){

if (! estaCheia()){

if (inicio == -1){

inicio = 0;

}

fim++;

f[fim] = e;

qtdeElementos++;

}

}

public void remover(){

if (! estaVazia() ){

inicio++;

qtdeElementos--;

}

}

public void mostrar(){

String elementos = "";

for (int i = inicio; i<=fim; i++) {

elementos += f[i]+ " - ";

}

JOptionPane.showMessageDialog(null, elementos);

}

}

public class FilaAplicacao {

public static void main(String[] args) {

Fila f = new Fila();

f.adicionar(10);

f.adicionar(12);

f.adicionar(30);

f.mostrar();

f.remover();

f.mostrar();

}

}

**3- Arvore Binária-** São estruturas de busca eficientes e muito úteis em diversas situações. Entendendo seu conceito, você criará soluções otimizadas para problemas de busca e acesso de dados. Sem dúvidas, **árvores**são estruturas de dados que usamos em várias situações, pois são muito úteis para armazenar dados hierárquicos. Dessa forma, podemos usá-las em bancos de dados, pastas de arquivos, sistemas de busca e na interface **DOM**(Document Object Model), que usa de conceitos como parent nodes (nó-pai) e child nodes (nó-filho). O dados são divididos em **nós** que compõem os **níveis** da árvore. O nível zero é a **raiz**, pois é a partir dele que toda a árvore vai se organizar. Chamamos os nós que partem da raiz de **nós-filhos** da raiz (child nodes). Eventualmente, quando novos nós saem dos nós-filhos, eles se tornam **nós-pai** (parent nodes) e assim por diante. Finalmente, temos ainda os **nós-folha**, que são aqueles que não tem nós-filhos.

**Cenário:** Usamos [algoritmos](https://blog.cod3r.com.br/algoritmo-e-logica-de-programacao-o-que-sao-e-para-que-servem/) de busca em árvores binárias quando queremos buscar um determinado valor dentro de uma árvore. Por exemplo, quando usamos propriedades childNodes ou firstChild no DOM, essas propriedades retornam uma nodeList de filhos (childNodes) do elemento passado e o primeiro filho do elemento passado (firstChild), respectivamente.

**Código:** Construção de um árvore com mínimo é implementado de maneira recursiva, enquanto o máximo de maneira iterativa:

public Node min() {

if (isEmpty()) return null;

return min(this.root);

}

private Node min(Node node) {

if (node.left == null) return node;

else return min(node.left);

}

public Node max() {

if (isEmpty()) return null;

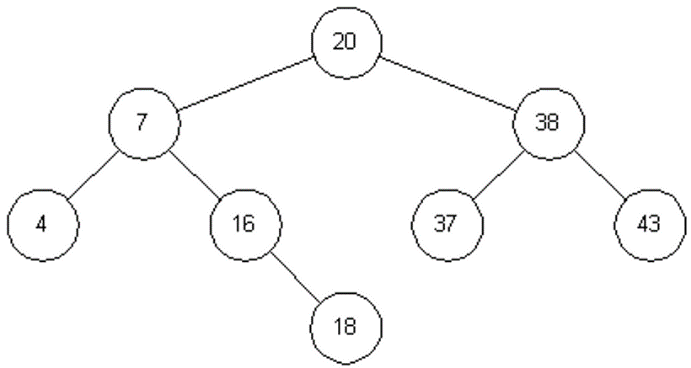
Node node = this.root;

while(node.right != null)

node = node.right;

return node;

}



1. (1 pt)A partir a arvore da figura acima
   1. descreva a saída para exibição em ordem

Na consulta em ordem, cada árvore é mostrada com o ramo da esquerda, a raiz e, posteriormente, o ramo da direita.

Em ordem: 4 7 16 18 20 37 38 43

* 1. descreva a saída para exibição em pós-ordem

Na consulta pós-ordem, cada árvore é mostrada com o ramo da esquerda, o ramo da direita e, posteriormente, a raiz.

Pós-ordem: 4 18 16 7 37 43 38 20

* 1. descreva a saída para exibição em pré-ordem

Na consulta pré-ordem, cada árvore é mostrada com a raiz, o ramo da esquerda e, posteriormente, o ramo da direita.

Pré-ordem: 20 7 4 16 18 38 37 43

* 1. Indique o nível o grau de cada um dos nós

A estrutura mais comum é a **árvore binária**, que tem no máximo dois nós-filhos a partir do nó inicial (chamado de **raiz** ou root). Um nó pode ter pais, irmãos e filhos; nós que têm pelo menos um filho são chamados de nós internos, e nós sem filhos são chamados de externos ou folhas:

Árvore Binária tem o Nível 3

O número 20 é a raiz de nivel zero e possui o nó raiz que é o nó inicial da árvore grau 2

O número 38 é o nível 1 a sub-arvore a direita de grau 2

O número 7 é o nível 1 a sub-arvore a esquerda de grau 2

O número 43 é nível 2 e nó folha de grau 0

O número 16 é nível 2 e nó folha de grau 1

O número 37 é nível 2 e nó folha de grau 0

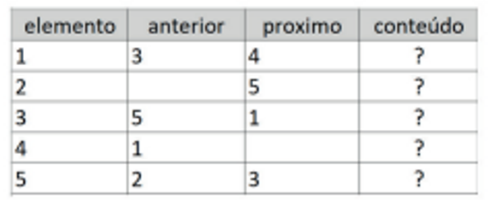
O número 4 é nível 2 e nó folha de grau 0

O número 18 é nível 3 e nó folha de grau 0

1. (0.5 pt) Considere a estrutura de dados do tipo Lista Encadeada, em que os elementos são alocados na memória à medida que são inseridos na lista, e removidos à medida que são dela retirados. Em um dado momento, essa lista possui 5 elementos (E1, E2, E3, E4 e E5), de forma que E1 aponta para E2, E2 aponta para E3, E3 aponta para E4 e E4 aponta para E5. Deseja-se retirar o elemento E3 dessa lista. Com base nas informações apresentadas, descreva a sequencia de passos para remover o elemento

Resposta: E2 passa a apontar para E4; Libera-se E3.

1. (0.5 pt) Abaixo tem-se uma tabela que ilustra o conjunto de nós de uma lista duplamente encadeada, contendo o total de 5 nós.



Ao imprimir a estrutura na ordem correta, o conteúdo apresentado será **I – F – S – P – 2019**, dessa forma, descreva a sequencia que preenchem, corretamente, a coluna “conteúdo”

Resposta: P – I – S – 2019 – F

O elemento 2 não tem elemento Anterior, logo sabemos que ele é o 1º elemento:  
1º -> 2

O próximo elemento ao 2 é o 5, logo:  
1º -> 2  
2º -> 5  
  
O elemento 5 informa que seu anterior é o elemento 2 (que já sabemos) e que seu próximo elemento é o 3:  
1º -> 2  
2º-> 5  
3º -> 3  
  
O elemento 3 informa que seu anterior é o elemento 5 (que já sabemos) e que seu próximo elemento é o 1:  
1º -> 2  
2º -> 5  
3º -> 3  
4º - > 1  
  
O elemento 1 afirma que seu anterior é o elemento 3 (que também já sabemos) e que seu posterior é o elemento 4, assim temos a **ordem correta**:  
1º -> 2  
2º -> 5  
3º -> 3  
4º -> 1  
5 -> 4  
  
O enunciado ainda diz que ao imprimir na **ordem correta** (a que acabamos de estruturar acima), será apresentado o conteúdo **I – F – S – P – 2019**. Logo:  
2 -- **I**  
5 -- **F**  
3 -- **S**  
1 -- **P**  
4 -- **2019**  
  
O comando da questão finalmente pede a correspondência desses valores acima encontrados, para a ordenação da tabela inicialmente apresentada na questão, assim:  
  
1 --> **P**  
2 -->**I**  
3 --> **S**  
4 --> **2019**  
5 --> **F**