

# Laboratório: LISTAS LINEARES, PILHAS E FILAS

A stylized illustration of a person with red hair and a red shirt sitting cross-legged on a large, glowing blue laptop. The person is holding a smaller laptop. The background is a vibrant orange and yellow gradient. Floating around the person and the laptop are various code snippets and symbols, including '&lt;div&gt;', 'scr=...', '{ }', '&lt;/&gt;', '==', '!', and '[]'. The overall aesthetic is futuristic and tech-oriented.

**LISTA DE FIGURAS**

Figura 2.1 – Estrutura de uma lista linear do tipo PILHA.....	7
Figura 2.2 – Esquema de tabuleiro para ensinar trajeto para um robô. ....	9
Figura 2.3 – Algoritmo da operação INIT para pilha encadeada. ....	11
Figura 2.4 – Estado da pilha executando a operação INIT. ....	11
Figura 2.5 – Algoritmo da operação IsEmpty pilha encadeada. ....	12
Figura 2.6 – Algoritmo da operação PUSH pilha encadeada. ....	12
Figura 2.7 – Esquema de alocação de novo nó na operação PUSH pilha encadeada. ....	13
Figura 2.8 – Algoritmo da operação TOP pilha encadeada.....	14
Figura 2.9 – Algoritmo da operação POP pilha encadeada. ....	14
Figura 2.10 – Esquema de retirada do nó que está no topo com a operação POP() pilha encadeada. ....	15
Figura 2.11 – Tela de saída da execução do programa que utiliza Pilha_INT. ....	18
Figura 2.12 – Esquema do <i>loop</i> para inserção de dados pelo PUSH() na pilha encadeada. ....	19
Figura 2.13 – Esquema do <i>loop</i> para remoção de elemento pelo POP() na pilha encadeada. ....	20
Figura 2.14 – Esquema da Lista Linear Especial: Fila .....	22
Figura 2.14 – Exemplo de um esquema de Fila Encadeada .....	22
Figura 2.15 – Algoritmo da operação INIT para Fila Encadeada.....	24
Figura 2.16 – Estado da fila executando a operação INIT. ....	24
Figura 2.17 – Algoritmo da operação IsEmpty para Fila Encadeada. ....	25
Figura 2.18 – Algoritmo da operação ENQUEUE para Fila Encadeada.....	25
Figura 2.19 – Esquema de alocação de novo nó na operação ENQUEUE fila encadeada. ....	26
Figura 2.20 – Esquema de alocação de novo nó na operação ENQUEUE fila encadeada. ....	27
Figura 2.21 – Algoritmo da operação DEQUEUE para Fila Encadeada.....	28
Figura 2.22 – Esquema da operação DEQUEUE para Fila Encadeada.....	28

**LISTA DE QUADROS**

Quadro 2.1 – Exemplo de sequência de operações sobre o tipo de dado PILHA e suas ações. ....	8
Quadro 2.2 – Sequência de operações usando PILHA para armazenar pontos de referência. ....	9

EXEMPLO

**LISTA DE CÓDIGOS-FONTE**

Código-fonte 2.1 – Implementação de Pilha_INT com uso das operações em JAVA .....	17
Código-fonte 2.2 – Programa que converte valor da base 10 para binário escrito em JAVA. ....	21
Código-fonte 2.3 – Implementação de Fila_INT com uso das operações em JAVA .	30
Código-fonte 2.4 – Programa que simula a fila de pacientes em consultório, escrito em JAVA. ....	33

EMSE

## SUMÁRIO

2 LABORATÓRIO: LISTAS LINEARES, PILHAS E FILAS .....	6
2.1 Listas Lineares Especiais: Pilhas Encadeadas .....	6
2.1.1 Implementando o tipo de dado pilha .....	10
2.1.2 Implementação em JAVA da Pilha de Inteiros .....	15
2.1.3 Exemplo de uso do tipo de dado pilha .....	20
2.2 Listas lineares especiais: filas encadeadas.....	21
2.2.1 Implementando o tipo de dado Fila .....	23
2.2.2 Implementação de FILA em JAVA .....	29
2.2.3 Exemplo de Uso do FILA.....	30
EXERCÍCIOS DE FIXAÇÃO.....	34
REFERÊNCIAS.....	36

## 2 LABORATÓRIO: LISTAS LINEARES, PILHAS E FILAS

Já foi definido que uma lista linear é um conjunto de elementos ordenados. A ordem que deve ser seguida depende da aplicação.

Por exemplo:

- Lista de presença de alunos em uma aula tem como critério de ordenação a ordem alfabética dos nomes dos alunos.
- Lista de clientes a serem atendidos em um banco deve ser ordenado pela ordem de chegada dos clientes.

O estudo dos aspectos conceituais sobre listas lineares feito anteriormente, apenas introduziu a base para que possamos agora criar listas lineares a serem usadas em aplicações reais. Então, vamos começar por listas lineares especiais pilha e fila, por serem de implementação mais simples. A estrutura fila, apesar de nos ser mais familiar, é um pouco mais trabalhosa de ser implementada, por isso, vamos iniciar nossos estudos com a implementação e uso de pilha.

### 2.1 Listas Lineares Especiais: Pilhas Encadeadas

Apenas para lembrar a definição, uma pilha (*stack*) é uma lista linear, e nela, as operações de inserção e de remoção são efetuadas em apenas uma extremidade, denominada **topo da pilha**. Estruturas deste tipo são conhecidas como LIFO (*last in first out* – último a entrar primeiro a sair). A figura: Estrutura de uma lista linear do tipo PILHA apresenta uma representação de uma pilha.

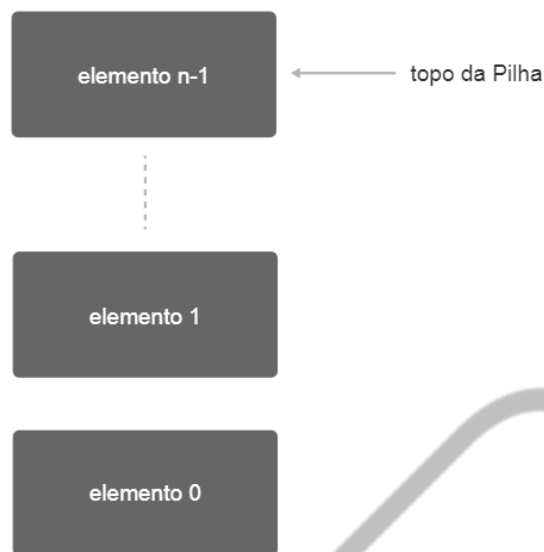


Figura 2.1 – Estrutura de uma lista linear do tipo PILHA.  
Fonte: Elaborado pelo autor (2018).

Aprendemos que quando definimos um tipo de dado, estamos definindo, além dos valores que podem ser armazenados por esse tipo, as operações que esses valores poderão sofrer. Desta forma, em relação aos valores que podem ser armazenados no tipo de dado PILHA, depende do tipo de dado que se deseja armazenar como elemento da pilha, o que depende da aplicação para qual estamos usando o tipo de dado pilha. Já quanto às operações do tipo de dado PILHA são definidas as seguintes operações:

#### **Básicas:**

- **PUSH(P,v):** armazena (empurra) na extremidade conhecida como topo da pilha P o elemento com valor v, aumentando o tamanho da pilha.
- **POP(P,v):** retira do topo da pilha P o elemento e armazena o valor do elemento em v, diminuindo a pilha.
- **TOP(P,v):** lê o elemento que está no topo da pilha P e armazena o valor do elemento em v.

#### **Auxiliares:**

- **INIT(P):** inicia a pilha deixando-a vazia.

- **IsEmpty (P):** (traduzindo para o português: “está vazia?”) verifica se a pilha está vazia, retornando verdade se a pilha estiver vazia e falso, caso contrário.
- **IsFull(P):** (traduzindo para o português: “está cheia?”) verifica se a pilha está cheia, retornando verdade se estiver a pilha cheia e falso, caso contrário.

No quadro a seguir, são apresentados exemplos de realização de operações sobre uma pilha P onde cada elemento armazena um valor inteiro.

Operações	Estado da Pilha P depois da Operação	v
INIT (P)	{}	
PUSH (P, 2)	{2}	
PUSH (P, 4)	{2, 4}	
TOP(P, v)	{2, 4}	v = 4
POP (P, v)	{2}	v = 4
PUSH (P, 5)	{2, 5}	
POP (P, v)	{2}	v = 5
POP (P, v)	{}	v = 2

Quadro 2.1 – Exemplo de sequência de operações sobre o tipo de dado PILHA e suas ações.

Fonte: Elaborado pelo autor (2018).

Descrevendo, de forma resumida, a aplicação da execução das operações apresentadas no quadro:

- A operação INIT(P) faz com que a estrutura de dados pilha seja iniciada como vazia.
- A execução da operação PUSH(P,2) insere na pilha P que estava vazia o valor 2, que passa a ser o elemento que está no topo da pilha.
- A execução da operação PUSH(P,4) insere na pilha P que estava com elemento 2 o valor 4 que passa a ser o elemento que está no topo da pilha.
- A operação TOP(P, v) faz a leitura do valor do elemento que está no topo da pilha P, porém a pilha não é alterada.
- A operação POP(P, v) faz retirada do elemento que está no topo da pilha P, neste caso o valor 4, a pilha diminui de tamanho.

Conforme a seção em que descrevemos pilha, uma lista linear PILHA é usada quando precisamos inverter a ordem de entrada dos elementos.



Então vamos a um exemplo prático. Suponha o seguinte tabuleiro com coordenadas [linha;coluna] conforme figura: Esquema de tabuleiro para ensinar trajeto para um robô. O caminho destacado deve ser o trajeto a ser programado em um robô para que ele caminhe no tabuleiro para chegar a um determinado ponto do tabuleiro e depois deve voltar ao ponto de onde partiu refazendo exatamente o mesmo trajeto. Assim:

- O caminho de ida deve ser: [1;1], [1;2],[2;2], [3;2],[3;3] e [3;4]
- O caminho de volta será: [3;4] , [3;3], [3;2], [2;2], [1;2] e [1;1]

	1	2	3	4
1				
2				
3				
4				

Figura 2.2 – Esquema de tabuleiro para ensinar trajeto para um robô.  
Fonte: Elaborado pelo autor (2018).

Para essa aplicação teríamos que usar uma pilha, uma vez que o caminho de volta deve ser exatamente o inverso da ida. Assim, devemos executar as operações descritas no quadro a seguir no programa a ser instalado no robô.

Operação	Pilha	Posição
INIT(pilha)	{ }	
PUSH(pilha, [1;1] )	{[1;1]}	
PUSH(pilha, [1;2] )	{[1;1], [1;2]}	
PUSH(pilha, [2;2] )	{[1;1], [1;2],[2;2]}	
PUSH(pilha, [2;3] )	{[1;1], [1;2],[2;2],[3;2]}	
PUSH(pilha, [3;3] )	{[1;1], [1;2],[2;2],[3;2],[3;3]}	
PUSH(pilha, [3;4] )	{[1;1], [1;2],[2;2],[3;2],[3;3],[3;4]}	
POP (pilha, posicao)	{[1;1], [1;2],[2;2],[3;2],[3;3]}	[3;4]
POP (pilha, posicao)	{[1;1], [1;2],[2;2],[3;2]}	[3;3]
POP (pilha, posicao)	{[1;1], [1;2],[2;2]}	[3;2]
POP (pilha, posicao)	{[1;1], [1;2]}	[2;2]
POP (pilha, posicao)	{[1;1]}	[1;2]
POP (pilha, posicao)	{ }	[1;1]

Quadro 2.2 – Sequência de operações usando PILHA para armazenar pontos de referência.

Fonte: FIAP (2018).

Agora que já conhecemos o que é pilha e suas operações, vamos implementá-la em JAVA.

### 2.1.1 Implementando o tipo de dado pilha

Para construirmos o tipo de dado pilha encadeada, deve-se definir o nó que armazena a informação a ser organizada como pilha e que agora precisa ter incluída a indicação do próximo elemento da pilha.

Apresentando primeiro na forma de um tipo abstrato de dado (TAD), ou seja, em algoritmo, a definição do nó seria:

---

Registro NO
Inicio
dado: do tipo_dos_elementos
prox: ponteiro para registro NO
Fim

---

Para que fazer a implementação de uma pilha é preciso não apenas definir o nó que armazena cada elemento, mas também a indicação da extremidade que representa o topo da pilha. Assim, já apresentando implementação do nó e do topo da pilha encadeada, da classe Pilha\_INT temos o seguinte trecho de programa JAVA:

```
public class Pilha_INT {  
    private static class NO{  
        public int dado;  
        public NO prox;  
    }  
  
    private static NO topo;  
    ...  
}
```

Nessa declaração, o atributo dado do NO é do tipo inteiro (int), mas nos exemplos e exercícios em que usaremos Pilha, vamos ter que adequar para o atributo dado poder armazenar, por exemplo, o nome de um aluno (string), um valor de preço (número real) ou mesmo um registro com atributos de um cliente.

Definido o nó, agora vamos implementar cada operação lembrando que nosso objetivo é sempre definir como as operações serão implementadas de forma independente da linguagem de programação.

## INIT

Lembrando que a pilha deve estar vazia no momento de sua criação. A função INIT deve deixar a indicação que o topo da pilha não aponta para um nó, essa indicação depende da linguagem de programação utilizada. Em linguagem algorítmica, será definida uma constante denominada “NULO”, que indica que o topo está apontando para nenhum nó. Desta forma, o algoritmo que representa a operação INIT é apresentado na figura: Algoritmo da operação INIT para pilha encadeada.

```
modulo INIT(topo)
início
topo = NULO;
fim
```

Figura 2.3 – Algoritmo da operação INIT para pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

Nesse momento, é importante já apresentar como é a representação esquemática do efeito da execução desse método que implementa a operação INIT.



Figura 2.4 – Estado da pilha executando a operação INIT.  
Fonte: Elaborado pelo autor (2018).

## IsEmpty

Verifica se a pilha está vazia. Retornando verdade se estiver vazia, caso contrário, retorna falso. A figura a seguir apresenta o algoritmo dessa operação.

```
modulo IsEmpty (topo)
  início
    se (topo == NULO) então
      retorna(verdade)
    senão
      retorna(falso)
  fim
```

Figura 2.5 – Algoritmo da operação IsEmpty pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

## PUSH

Esta operação deve primeiro alocar mais um nó e em seguida “encaixar” esse novo nó como o nó que fica no topo da pilha. A figura a seguir apresenta o algoritmo da operação PUSH.

```
modulo PUSH(topo, elem)
  início
    novo: ponteiro para nó
    novo: ALOCA (NO)
    novo..prox = topo
    novo.dado = elem
    topo = novo
  fim
```

Figura 2.6 – Algoritmo da operação PUSH pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

A figura a seguir apresenta o esquema da execução dos comandos executados pela operação PUSH, supondo como estado inicial da pilha é que esteja vazia, e com a inserção de um novo nó.

Suponha a operação PUSH (topo, 3).

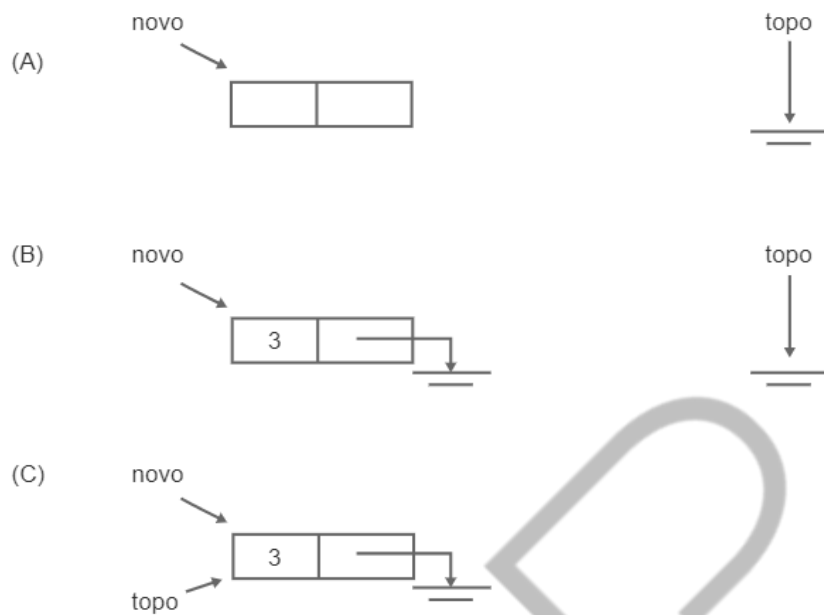


Figura 2.7 – Esquema de alocação de novo nó na operação PUSH pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

- (A) Supondo que a pilha está vazia, depois do ALOCA() o nó está alocado e é apontado por novo.
- (B) Como o novo nó passa a ser o topo da pilha este novo nó deve indicar que seu sucessor é o local que era o topo da pilha antes da inserção do novo nó (apontado por novo). Assim, o campo prox passa a apontar para o mesmo local que o ponteiro topo aponta, como a pilha estava vazia o campo prox recebe NULO. Em seguida o campo dado do novo nó recebe o valor 3 passado como parâmetro para o módulo PUSH.
- (C) Posicionando o topo para apontar a pilha em sua nova configuração, lembre-se que na pilha o topo sempre deve apontar para o último elemento que foi inserido.

## TOP

Esta operação retorna o valor do dado do nó que está no topo da pilha caso a pilha não estiver vazia. O algoritmo a seguir apresenta o módulo TOP.

```
modulo TOP(topo, elem)
if (IsEmpty(topo) == FALSO)
  início
  elem = topo.dado
  retorna(elem)
  fim
fim
```

Figura 2.8 – Algoritmo da operação TOP pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

## POP

Esta operação deve considerar que só poderá retirar um nó se a pilha não estiver vazia. No algoritmo a seguir, o módulo POP verifica se a pilha não está vazia utilizando a operação IsEmpty() que retorna verdade se a pilha estiver vazia.

```
modulo POP(topo, elem)
início
if (IsEmpty(topo) == FALSO)
  início
  elem = topo.dado
  topo = topo.prox
  fim
fim
```

Figura 2.9 – Algoritmo da operação POP pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

Supondo a pilha que foi gerada na explicação da operação PUSH (que terminou com apenas um elemento com dado 3) e, em seguida, executando a operação POP(topo, elem) para exemplificar. A figura a seguir apresenta o esquema do nó e das alterações de ponteiros realizadas na operação POP.

- (A) Inicialmente a pilha está com apenas 1 nó e topo aponta para este.
- (B) Com a atribuição elem = topo.dado o valor 3 é atribuído parâmetro elem e, em seguida, topo = topo.prox fazendo com que o ponteiro topo passe a apontar para o mesmo local onde o campo prox aponta, ou seja, para NULO.
- (C) Com a liberação da área do nó com dado 3 a configuração final da pilha é que topo aponta para NULO.

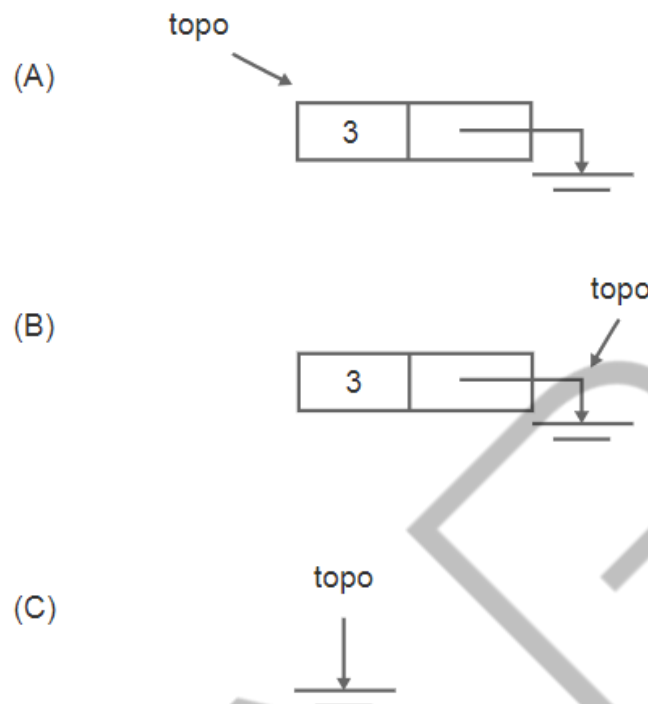


Figura 2.10 – Esquema de retirada do nó que está no topo com a operação POP() pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

### 2.1.2 Implementação em JAVA da Pilha de Inteiros

A partir da especificação do Tipo Abstrato de Dado (TAD) da pilha agora é hora de implementarmos. Usando JAVA para execução nossa TAD Pilha\_INT (como chamaremos a partir deste momento), o tipo de dado pilha torna-se então uma classe chamada de Pilha\_INT, nesta temos a classe NO, o atributo topo e métodos para todas as operações necessárias para manipular a pilha.

Observe código a seguir, que apresenta essa implementação, e em seguida, vamos fazer comentários sobre as adaptações necessárias para a construção do tipo concreto de dado na linguagem JAVA.

```
import java.util.*;

public class Pilha_INT {

    //definição do NO
    private static class NO{
        public int dado;
        public NO prox;
    }
}
```

```
}

//definição do retornos dos métodos POP e TOP
private static class Retorno{
    public int item;
    public boolean ok;
}

//definição do ponteiro topo da pilha
private static NO topo;

public void INIT() {
    topo = null;
}

public boolean IsEmpty() {
    return topo == null;
}

public void PUSH(int item) {
    NO novo = new NO();
    novo.dado = item;
    novo.prox = topo;
    topo = novo;
}

public Retorno POP() {
    Retorno saida = new Retorno();
    if(!IsEmpty()) {
        saida.item = topo.dado;
        topo = topo.prox;
        saida.ok = true;
    }
    else
        saida.ok = false;
    return saida;
}

public Retorno TOP() {
    Retorno saida = new Retorno();
    if(!IsEmpty()) {
        saida.item = topo.dado;
        saida.ok = true;
    }
    else
        saida.ok = false;
    return saida;
}

//Função main que exemplifica a utilização das operações sobre pilha
public static void main(String[] args) {

    Pilha_INT s = new Pilha_INT();
    Scanner entrada = new Scanner(System.in);
```



```
//Declaração de variáveis necessárias para usar métodos da Pilha_INT
    int item ;
    int opcao;
    Retorno res = new Retorno();

//inicia a pilha fazendo topo = null
    s.INIT();

//invoca metodo TOP para obter dado do nó do topo da pilha
    res = s.TOP();
    if (res.ok)
        System.out.println("Execução do TOP: " + res.item);
    else
        System.out.println("Execução do TOP: pilha VAZIA");

//repetição para inserir elementos na pilha
    do {
        System.out.println("Digite valor inteiro para dado ");
        item= entrada.nextInt();
        s.PUSH(item);
        System.out.println("Digite 0 para encerrar empilhamento de dados ");
        opcao=entrada.nextInt();
    } while (opcao != 0);

//invoca metodo TOP para obter dado do nó do topo da pilha
    res = s.TOP();
    if (res.ok)
        System.out.println("Execução do TOP: " + res.item);

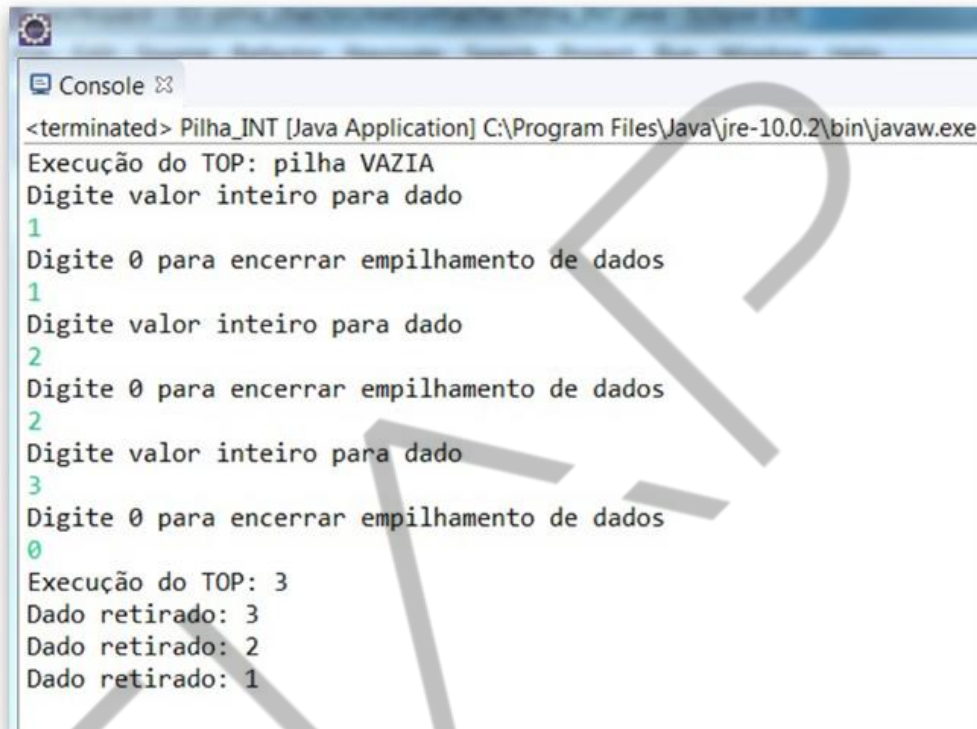
//repetição para retirar e apresentar todos elementos da pilha
    do {
        res = s.POP();
        if (res.ok)
            System.out.println("Dado retirado: " + res.item);
    } while (res.ok);
    entrada.close();
}
```

Código-fonte 2.1 – Implementação de Pilha\_INT com uso das operações em JAVA  
Fonte: Elaborado pelo autor (2018)

Na main() é possível notar que foi necessário declarar algumas variáveis que são necessárias para utilização dos métodos. A única que merece comentário é a res, como os métodos TOP() e POP() podem ser invocados quando a pilha está vazia, e se isso acontecer, não há dado para ser retornado, torna-se necessário que não apenas o valor do dado retorne, mas também se a operação efetuada foi um sucesso ou não. Assim, se a pilha não está vazia, o dado pode ser retornado e está ok (*true* - é verdade que a operação foi um sucesso). Portanto, o tipo de retorno dos métodos

TOP() e POP() foi alterado para o tipo Retorno, pois assim o dado e a situação ok ou não são retornados juntos.

A tela de saída da execução do programa é apresentada na figura: Tela de saída da execução do programa que utiliza Pilha\_INT e será usada para explicação do que ocorre com a execução do programa.



```
<terminated> Pilha_INT [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe
Execução do TOP: pilha VAZIA
Digite valor inteiro para dado
1
Digite 0 para encerrar empilhamento de dados
1
Digite valor inteiro para dado
2
Digite 0 para encerrar empilhamento de dados
2
Digite valor inteiro para dado
3
Digite 0 para encerrar empilhamento de dados
0
Execução do TOP: 3
Dado retirado: 3
Dado retirado: 2
Dado retirado: 1
```

Figura 2.11 – Tela de saída da execução do programa que utiliza Pilha\_INT.  
Fonte: Elaborado pelo autor (2018)

A pilha inicia sendo colocada no estado “vazia” usando o método **INIT()**.

Logo após, foi feita a invocação do método **TOP(res)** e como a pilha está vazia é retornado o valor de ok é *false*, uma vez que não há nó inserido na pilha é apresentada na tela de saída a mensagem de “Execução do TOP: pilha VAZIA”.

No programa, os dados são inseridos na pilha fazendo uso de uma repetição que lê do teclado o valor da variável item, e depois invoca o método **PUSH(item)** para inserir novo nó tendo o valor de item como campo dado. Supondo a execução apresentada na tela de saída, seriam inseridos, nessa ordem, os valores de item 1, 2 e 3. A sequência de operações realizadas é apresentada na figura: Esquema do *loop* para inserção de dados pelo PUSH() na pilha encadeada, assim como na real execução do programa a pilha se encontra vazia antes de iniciar a construção *do..while*. Lembre-se que os comandos que são efetuados dentro do método PUSH já

foram descritos e esquematizados na figura: Esquema de alocação de novo nó na operação PUSH pilha encadeada.

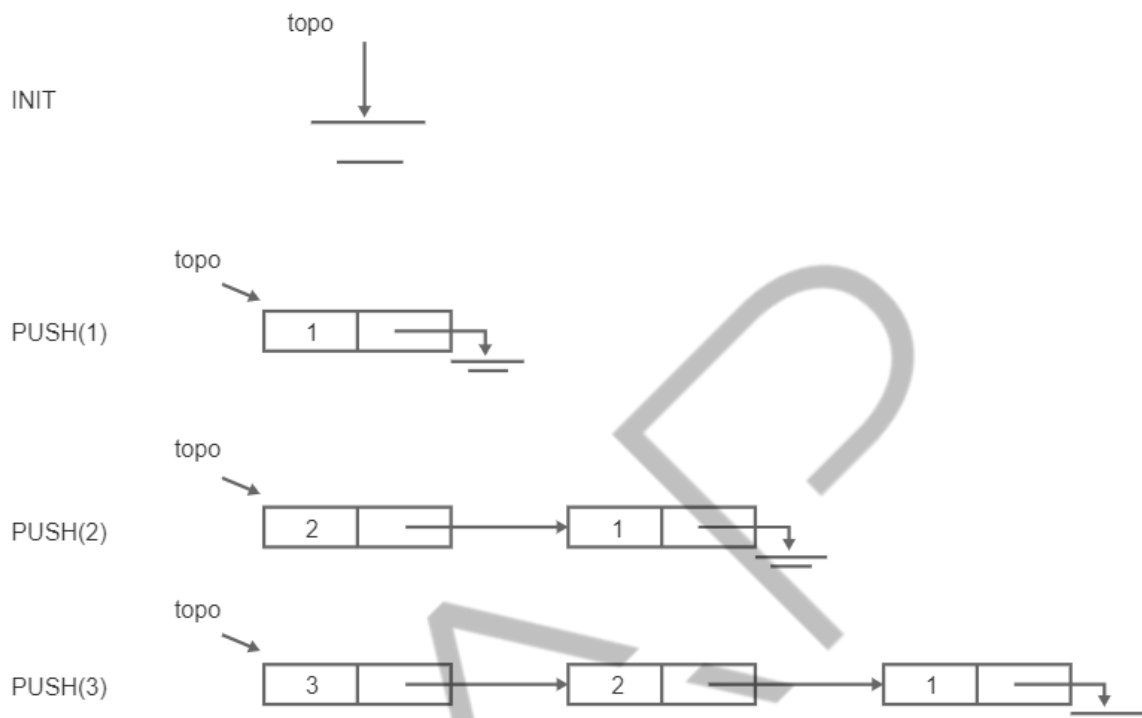


Figura 2.12 – Esquema do *loop* para inserção de dados pelo PUSH() na pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

Continuando a execução do programa, mais uma vez foi feita a invocação do método **TOP(res)**, só que desta vez a pilha não está vazia. Desta forma, o atributo `ok` de `res` recebe do método o valor *true* e assim o valor do dado que está no topo da pilha é apresentado na tela de saída (`res.item`).

Finalmente, o programa termina com a retirada de todos os nós da pilha e apresentação dos dados contidos em cada nó. Para realizar essa etapa mais uma vez foi utilizado `res`, que recebe do método **POP(res)** os valores para os atributos `ok` (*true* no caso da pilha não vazia e *false* caso contrário) e `item` que terá apenas valor válido (valor do dado que está no topo da pilha) no caso de `ok = true`. Sendo assim, a repetição *do... while* fará com que POP seja executado até o momento em que a pilha estiver vazia. Observe na figura Tela de saída da execução do programa que utiliza `Pilha_INT` que os valores são retirados na ordem inversa em que foram inseridos, conforme o correto funcionamento de uma pilha (LIFO - último a entrar/ primeiro a sair). O esquema da pilha em cada execução da operação POP é apresentado na figura a seguir.

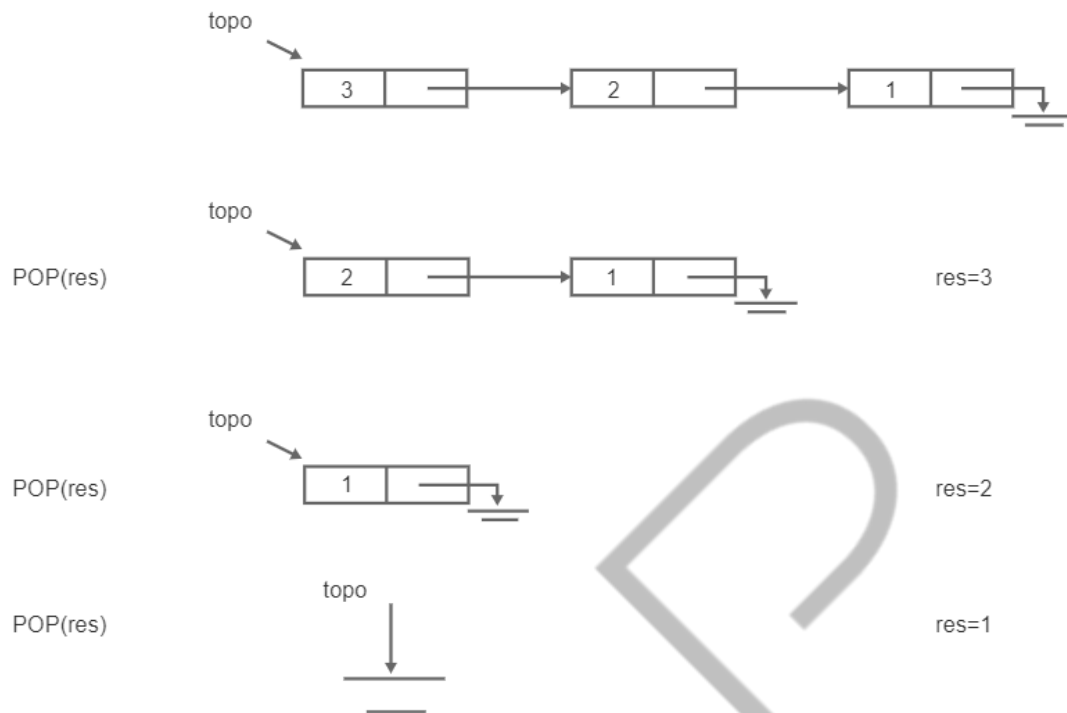


Figura 2.13 – Esquema do *loop* para remoção de elemento pelo POP() na pilha encadeada.  
Fonte: Elaborado pelo autor (2018).

### 2.1.3 Exemplo de uso do tipo de dado pilha

Agora que elaboramos as operações que manipulam pilha temos que utilizá-las em uma possível aplicação, a fim de mostrar como as funções são utilizadas. Suponha o seguinte problema: a conversão de um número representado em decimal em binário. Apenas para recordar como essa conversão é feita suponha o valor 9 em decimal:

$$13 / 2 = 6 \text{ resto } 1$$

$$6 / 2 = 3 \text{ resto } 0$$

$$3 / 2 = 1 \text{ resto } 1$$

$$1 / 2 = 0 \text{ resto } 1$$

O valor na base 2 é obtido escrevendo o resto na ordem inversa em que foram obtidos, assim, 13 na base 10 em binário é 1101 que representa  $(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)$ .

Para construir o programa é importante entender como a pilha nos auxiliará.

Lembre-se, pilha é usada para inverter a ordem em que os elementos foram inseridos. Desta forma, cada resto obtido é inserido na pilha e quando a divisão obtiver resultado da parte inteira 0 basta desempilhar todos os elementos da pilha.

O programa para realizar a conversão de decimal para binário é apresentado a seguir. No código abaixo foi omitida toda a especificação da classe Pilha\_INT, pois é rigorosamente a mesma implementação apresentada no programa anterior.

```
public static void main(String[] args) {  
  
    Pilha_INT s = new Pilha_INT();  
    Retorno res = new Retorno();  
    Scanner entrada = new Scanner(System.in);  
    int resto, num;  
  
    //inicia a pilha fazendo topo = null  
    s.INIT();  
  
    System.out.print("Digite valor número na base 10: ");  
    num = entrada.nextInt();  
  
    // fazendo divisões sucessivas e empilhando os valores do  
    // resto até que num seja zero  
    while (num > 0) {  
        resto = num % 2;  
        s.PUSH(resto);  
        num = num/2;  
    }  
  
    /* escrevendo o resto na ordem inversa que foram obtidos  
    usando a propriedade LIFO da pilha*/  
    System.out.println("Numero em binario: ");  
    do {  
        res = s.POP();  
        if (res.ok)  
            System.out.print(" " + res.item);  
    } while (res.ok);  
    entrada.close();  
}
```

Código-fonte 2.2 – Programa que converte valor da base 10 para binário escrito em JAVA.  
Fonte: Elaborado pelo autor (2018)

## 2.2 Listas lineares especiais: filas encadeadas

Uma fila (*queue*) é um conjunto ordenado de elementos onde a remoção de elementos se faz em um extremo (chamado início da fila) e a inserção em outro extremo (final da fila). Em outras palavras, uma estrutura de dados que segue a disciplina FIFO – *First In First Out* (primeiro a entrar primeiro a sair).

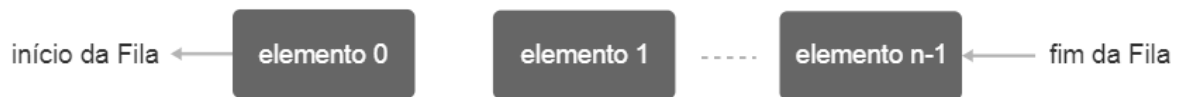


Figura 2.14 – Esquema da Lista Linear Especial: Fila  
Fonte: Elaborado pelo autor (2018).

Como vamos estudar apenas a implementação dinâmica, ou seja, os elementos da fila serão alocados dinamicamente, deve-se inicialmente definir o nó que será o elemento da fila. A declaração do nó é exatamente a mesma da pilha e da lista já estudadas, ou seja, cada nó é composto de 2 campos: dado (do tipo da informação que se deseja armazenar na fila) e ponteiro para próximo.

A figura: Exemplo de um esquema de Fila Encadeada apresenta um exemplo de esquema de uma fila encadeada com apenas 2 nós. Note que para manter as extremidades de início e final da fila é preciso agora ter 2 ponteiros auxiliares: ini (início da fila) e fim (final da fila). Assim, pela figura também podemos inferir que o nó com dado v1 foi o primeiro elemento que entrou na fila, uma vez que o ponteiro ini aponta para ele. Além disso, podemos ver que pelo ponteiro prox do nó com dado v1 que o nó com dado v2 foi o segundo inserido. Como a fila tem apenas 2 elementos, o nó com dado v2 é também o último elemento a entrar na fila por isso o ponteiro final está apontando para esse nó.

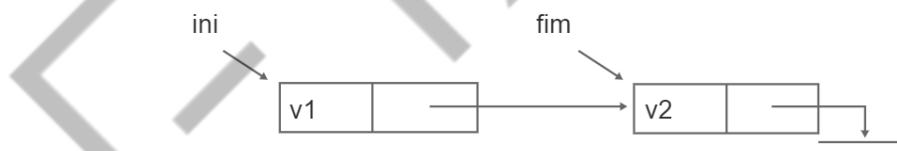


Figura 2.15 – Exemplo de um esquema de Fila Encadeada  
Fonte: Elaborado pelo autor (2018).

Com a descrição feita até aqui, podemos concluir que uma fila precisa dos seguintes componentes:

- O armazenamento de elementos.
- Ponteiro para início da fila (ini).
- Ponteiro para final da fila (fim).

Portanto, definindo como TAD esses componentes, temos:

---

Registro NO

Início

dado: do tipo\_dos\_elementos

prox: ponteiro para registro NO

Fim

ini,fim : ponteiro para registro NO

---

Agora vamos para fazer especificação de tipo de dado FILA de forma mais forma:

- Os valores que serão armazenados na fila depende do tipo de dado que a aplicação precisa armazenar.
- As operações necessárias para a manipulação da fila são:
  - **ENQUEUE (fila,v)**: insere um elemento em apenas uma extremidade da fila, conhecida como final da fila.
  - **DEQUEUE(fila,v)**: remove um elemento em apenas uma extremidade da fila, conhecida como início da fila.
  - **FIRST(fila,v)**: lê o elemento que está no início da fila e armazena em v.
  - **INIT(fila)**: inicia a fila deixando-a vazia.
  - **IsEmpty (fila)**: verifica se a fila está vazia, retornando verdade se estiver vazia e falso, caso contrário.

Agora, mãos à obra para elaborar as operações sobre fila encadeada. Sempre primeiro apresentando o tipo abstrato (TAD) e depois a implementação em linguagem JAVA.

## 2.2.1 Implementando o tipo de dado Fila

### INIT

Lembrando que a fila deve estar vazia no momento de sua criação. A função INIT deve deixar a indicação que os ponteiros de início e fim da fila não apontam para

um nó. Para tanto, em linguagem algorítmica, os dois ponteiros devem receber (ou apontar) para NULO.

Assim, iniciar uma fila deixando-a na condição de fila vazia implica em fazer  $ini = fim = NULO$ .

Desta forma, o algoritmo que representa a operação INIT será como apresentado como na figura a seguir.

```
modulo INIT (ini, fim)
  Início
    ini = NULO;
    fim = NULO;
  Fim
```

Figura 2.16 – Algoritmo da operação INIT para Fila Encadeada  
Fonte: Elaborado pelo autor (2018).

A representação do efeito da execução da função INIT é apresentada na figura.



Figura 2.17 – Estado da fila executando a operação INIT.  
Fonte: Elaborado pelo autor (2018).

### **IsEmpty**

Verifica se a fila está vazia, retornando verdade se estiver vazia, caso contrário retorna falso. Em algoritmo da operação, IsEmpty é como apresentado na figura Algoritmo da operação IsEmpty para Fila Encadeada.



```
modulo IsEmpty (ini, fim)

Início
se (ini == NULO) e (fim == NULO) então
    retorna(verdade)
senão
    retorna(falso)
Fim
```

Figura 2.18 – Algoritmo da operação IsEmpty para Fila Encadeada.  
Fonte: Elaborado pelo autor (2018).

Observe que a fila é considerada vazia se estiver no estado em que é iniciada pela função INIT.

### ENQUEUE (Insere na Fila)

Esta operação deve iniciar alocando um novo nó para ser inserido na fila, depois armazenar um valor no campo dado do novo nó e finalmente, “encaixar” o novo nó na posição correta e alterar os ponteiros necessários. Para o “encaixe”, devemos primeiro entender que quando inserimos um elemento na fila esse sempre se torna o último, sendo assim, não tem elemento que o sucede e, portanto, o ponteiro prox desse nó deve apontar para NULO.

Além disso, é preciso perceber que existem duas situações distintas que devem ser analisadas para a inserção de um novo elemento:

- Se a fila estiver vazia, ambos os ponteiros de início e final da fila devem apontar para o mesmo nó.
- Caso contrário, apenas deve-se movimentar o ponteiro que aponta para o nó que está no final da fila.

```
modulo ENQUEUE(ini, fim, v)

Início
novo: ponteiro para nó
novo = ALOCA (no)
novo.prox = NULO
novo.dado = v
se (IsEmpty(ini, fim) == verdade)
    ini = novo
senão
    fim.prox = novo
fim = novo
Fim
```

Figura 2.19 – Algoritmo da operação ENQUEUE para Fila Encadeada.  
Fonte: Elaborado pelo autor (2018).

A figura: Esquema de alocação de novo nó na operação ENQUEUE fila encadeada apresenta esquemas gráficos das etapas realizadas operação ENQUEUE. Supondo que o estado inicial da fila esteja que está vazia e, em seguida é efetuada a chamada da função ENQUEUE (ini, fim, 8).

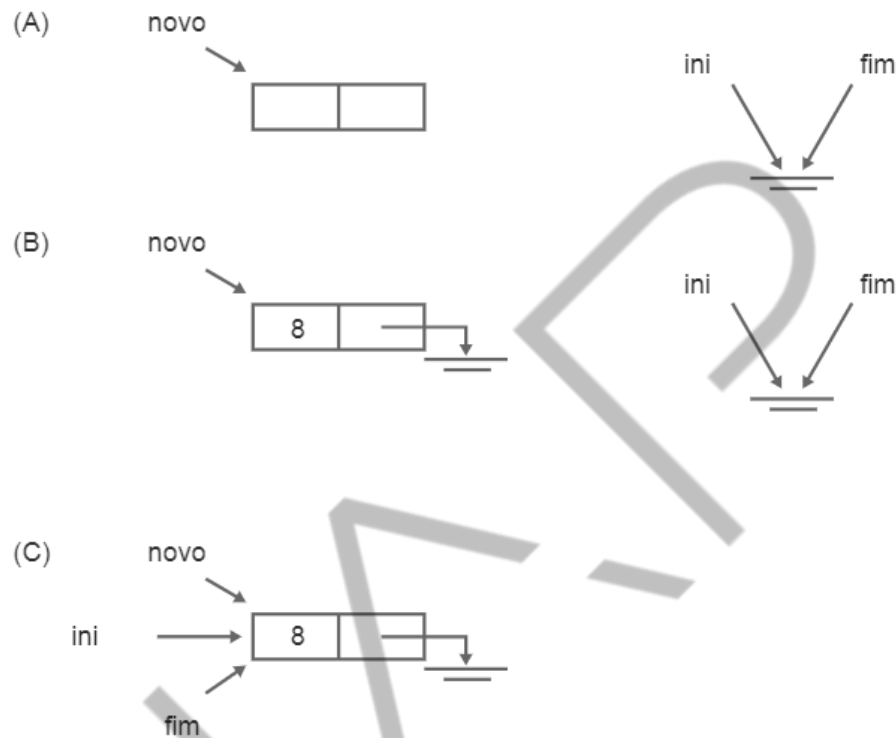


Figura 2.20 – Esquema de alocação de novo nó na operação ENQUEUE fila encadeada.  
Fonte: Elaborado pelo autor (2018).

A descrição do passo a passo da inserção do novo nó:

- (A) A indicação de que a fila está vazia ocorre pelo estado em que se encontram os ponteiros ini e fim. Depois de alocar o nó este é apontado por novo.
- (B) O campo dado recebe 8 e o campo prox do novo nó alocado recebe o ponteiro NULO (uma vez que sempre o nó inserido será o último da fila, ou seja, sem sucessor).
- (C) Posiciona o ponteiro ini (uma vez que a fila está vazia) e o ponteiro fim para apontar o novo nó alocado.

Para entender bem todas as etapas realizadas na inserção de um novo nó, vamos supor que a operação ENQUEUE(ini, fim, 5) seja executada em seguida. Assim, a figura a seguir ilustra as operações realizadas.

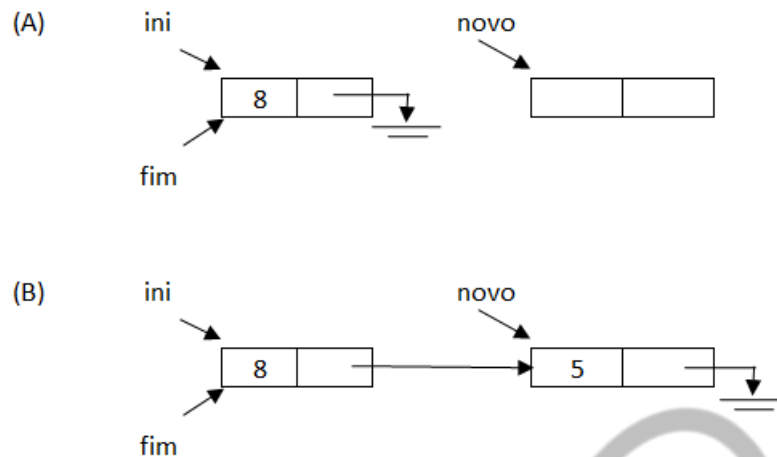


Figura 2.21 – Esquema de alocação de novo nó na operação ENQUEUE fila encadeada.  
Fonte: Elaborado pelo autor (2018).

- (A) Iniciando com a alocação de um novo nó e a fila com estado deixado pelo ENQUEUE anterior.
- (B) No novo nó campo dado recebe 5 e prox sempre recebe NULO. Verificando que a fila não está mais vazia, o nó que estava no final da fila passa ter seu campo prox apontando para o mesmo local que o ponteiro novo aponta (ou seja, para o novo nó alocado) e depois fim passa apontar para o novo nó.

### DEQUEUE (Retira Elemento da Fila)

Esta operação retira um elemento do início da fila se esta não estiver vazia. Para tanto, outras condições devem ser verificadas. Quais? Vamos analisar agora.

- Verifica se fila está vazia, se não estiver vazia.
- Se fila for composta por apenas 1 elemento.
- Os ponteiros ini e fim devem ser alterados para NULO quando o elemento for retirado.
- Caso tenha mais do que 1 elemento.
- O ponteiro ini é o único a ser alterado e deve apontar para o nó que se torna o primeiro da fila, ou seja, o nó sucessor do que está sendo retirado.

Em linguagem algorítmica é apresentada na Figura 2.22 seguir.

```

modulo DEQUEUE(ini, fim, v)

Início
se (IsEmpty(ini, final) == verdade) então
    retorna (falso)
senão
    Início
    v=ini.dado
    ini= ini.prox
    se(ini == NULO)
        fim = NULO
    retorna (verdade)
Fim
Fim

```

Figura 2.22 – Algoritmo da operação DEQUEUE para Fila Encadeada  
Fonte: Elaborado pelo autor (2018).

A figura Esquema da operação DEQUEUE para Fila Encadeada apresenta como a operação DEQUEUE atua sobre uma fila que inicialmente tem 3 elementos e que tem o nó apontado por ini retirado da fila alterando o parâmetro v recebendo o valor 8 e o módulo termina retornando verdade.

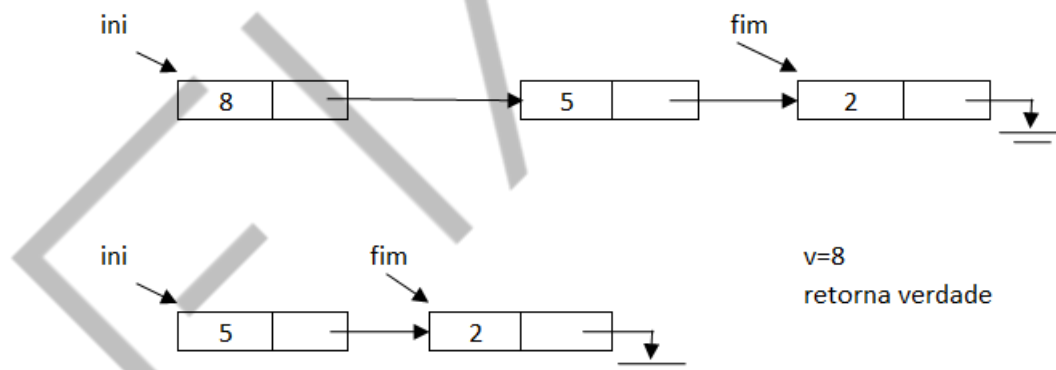


Figura 2.23 – Esquema da operação DEQUEUE para Fila Encadeada  
Fonte: Elaborado pelo autor (2018).

## FIRST

Essa operação lê o valor do dado do elemento que está no início da fila e retorna pelo parâmetro v. Essa função é equivalente da função TOP de uma pilha, ou seja, apenas retorna o valor do dado do elemento que está no início da fila sem alterar a fila.

Como exercício implemente essa função. Lembre-se de verificar se a fila não está vazia.

### 2.2.2 Implementação de FILA em JAVA

A implementação do tipo de dado FILA segue praticamente todas as considerações já descritas na implementação de pilha. Assim, poucas observações serão necessárias.

O código a seguir apresenta a implementação do tipo concreto de dado Fila\_INT com seus atributos e os métodos que implementam todas as operações do tipo abstrato de dado FILA.

```
public class Fila_INT {  
    //definição nó da FILA  
    private static class NO{  
        public int dado;  
        public NO prox;  
    }  
  
    //definição dos ponteiros ini e fim  
    private static NO ini;  
    private static NO fim;  
  
    private static class Retorno{  
        public int item;  
        public boolean ok;  
    }  
  
    public void INIT() {  
        ini = null;  
        fim = null;  
    }  
  
    public boolean IsEmpty() {  
        return (ini == null && fim == null);  
    }  
  
    public void ENQUEUE(int item) {  
        NO novo = new NO();  
        novo.dado = item;  
        novo.prox = null;  
        if (IsEmpty())  
            ini = novo;  
        else  
            fim.prox = novo;  
        fim = novo;  
    }  
  
    public Retorno DEQUEUE() {  
        Retorno saida = new Retorno();
```

```
        if(!isEmpty()) {
            saida.item = ini.dado;
            ini = ini.prox;
            if (ini == null) fim = null;
            saida.ok = true;
        }
        else
            saida.ok = false;
        return saida;
    }

    public static void main(String[] args) {

        //Instanciando o objeto fila
        Fila_INT fila = new Fila_INT();

        Retorno resultado = new Retorno();
        Scanner entrada = new Scanner(System.in);
        int item, opcao;

        fila.INIT();

        //repetição para inserir elementos na FILA
        do {
            System.out.print("Digite dado inteiro: ");
            item= entrada.nextInt();
            fila.ENQUEUE(item);
            System.out.print("Digite 0 para encerrar inserção de dados ");
            opcao=entrada.nextInt();
        }while (opcao != 0);

        // repetira elemento da FILA até que esta fique vazia
        do {
            resultado = fila.DEQUEUE();
            if (resultado.ok)
                System.out.println("Dado retirado: " + resultado.item);
        } while (resultado.ok);

        entrada.close();
    }
}
```

Código-fonte 2.3 – Implementação de Fila\_INT com uso das operações em JAVA  
Fonte: elaborado pelo autor (2018)

### 2.2.3 Exemplo de Uso do FILA

A aplicação mais óbvia do tipo de dado fila é sua utilização em situações cotidianas.

O programa a seguir simula o atendimento de pacientes em um consultório que não trabalha com agenda de horário colocando os pacientes em uma fila por ordem de chegada. A simulação ocorre através de um menu onde cada ação possível pode ser escolhida.

Cada paciente para entrar na fila deve fornecer o seu nome. A fila terá como campo dado o nome (String) de cada paciente que entra no consultório e que será atendido.

```
import java.util.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Fila_Paciente {

    private static class NO{
        public String dado;
        public NO prox;
    }

    private static NO ini;
    private static NO fim;

    private static class RetornoP{
        public String item;
        public boolean ok;
    }

    public void INIT() {
        ini = null;
        fim = null;
    }

    public boolean IsEmpty() {
        return (ini == null && fim == null);
    }

    public void ENQUEUE(String item) {
        NO novo = new NO();
        novo.dado = item;
        novo.prox = null;
        if (IsEmpty())
            ini = novo;
        else
            fim.prox = novo;
            fim = novo;
    }

    public RetornoP DEQUEUE() {
        RetornoP saida = new RetornoP();
```

```
        if(!isEmpty()) {
            saida.item = ini.dado;
            ini = ini.prox;
            if (ini == null) fim = null;
            saida.ok = true;
        }
        else
            saida.ok = false;
        return saida;
    }

    public static void main(String[] args) throws IOException {

        // Instacia a fila de pacientes
        Fila_Paciente fila = new Fila_Paciente();
        RetornoP resultado = new RetornoP();
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        Scanner entrada = new Scanner(System.in);

        int opcao;

        // Fila inicia vazia
        fila.INIT();

        // Menu para simular entrada e saída de pacientes em uma
        // fila em um consultório médico
        do {
            System.out.println("1 - Insere paciente na fila de espera ");
            System.out.println("2 - Chama cliente para atendimento ");
            System.out.println("3 - Sair - apenas se não houver mais cliente na fila
");

            opcao=entrada.nextInt();
            switch (opcao) {
                case 1:
                    System.out.println("Digite nome do paciente ");
                    String item= in.readLine();
                    fila.ENQUEUE(item);
                    break;
                case 2:
                    resultado = fila.DEQUEUE();
                    if (resultado.ok)
                        System.out.println("Cliente Chamado: "+
resultado.item);
                    break;
                case 3:
                    if (!fila.isEmpty()) {
                        opcao = 4;
                        System.out.println("Não pode encerrar, pois há
clientes na fila ");
                    }
                    break;
                default:
                    System.out.println("Opção inválida ");
            }
        }
    }
}
```



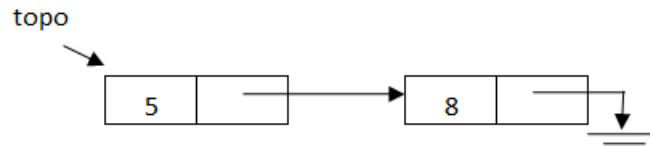
```
    }  
    }while (opcao !=3);  
    entrada.close();  
}  
}
```

Código-fonte 2.4 – Programa que simula a fila de pacientes em consultório, escrito em JAVA.  
Fonte: Elaborado pelo autor (2018)

EMENDAS

## EXERCÍCIOS DE FIXAÇÃO

Sabendo que a pilha se encontra como na figura a seguir:



- (a) Faça o esquema gráfico da execução do POP(res)
- (b) Faça o esquema gráfico da execução do PUSH(2)

Com base na especificação da TAD pilha, faça o esquema gráfico do estado da pilha com a execução de cada operação e também apresente o que será apresentado na tela de saída pelo comando escreva().

```
INIT()
PUSH(topo, 3);
PUSH(topo, 6);
POP(topo, x);
PUSH(topo, 2);
POP(topo, y);
s = x + y;
POP(topo, x);
m = s * x;
escreva(s, m, x, y);
```

Crie uma pilha para armazenar dados do tipo caractere criando uma classe Pilha\_CHAR.

Utilizando a classe PILHA\_CHAR do exercício anterior, elabore um programa que use a pilha de caracteres para verificar se uma palavra.

Exemplos de palíndromos: asa, arara, 1001, etc.

1) Implemente o método da operação FIRST da classe Fila\_INT.

2) Implemente um programa que simule a inserção e remoção de processos na fila de uso do processador, para tanto o programa deve ter um menu com as seguintes opções:

- Submete processo: lê do teclado a identificação do processo (pid - valor inteiro) e insere na fila de processos;
- Processa: retira da fila 1 processo (apresente o pid) e depois lê do teclado se este foi concluído
  - Se sim escreve na tela de saída mensagem de conclusão do processo (pid).
  - Se não processo deve retornar ao final da fila.
  - Encerrar programa, permitido apenas se a fila estiver vazia.

## REFERÊNCIAS

ASCÊNCIO, A.F.G; ARAUJO, G.S. **Estruturas de Dados**: Algoritmos, Análise de Complexidade e Implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010.

DEITEL, P; J.; Deitel, H.M. **Java: como programar**. 8 ed. São Paulo: Pearson Prentice Hall, 2010.

PEREIRA, S.L. **Estruturas de Dados Fundamentais**: Conceitos e Aplicações. São Paulo: Érica, 1996.

TENEMBAUM, A.M et al. **Estruturas de Dados usando C**. São Paulo: Makron Books, 1995.