Algoritmos e Programação: Estruturas Lineares

Tipos Especiais de Listas

Pilha

Fila

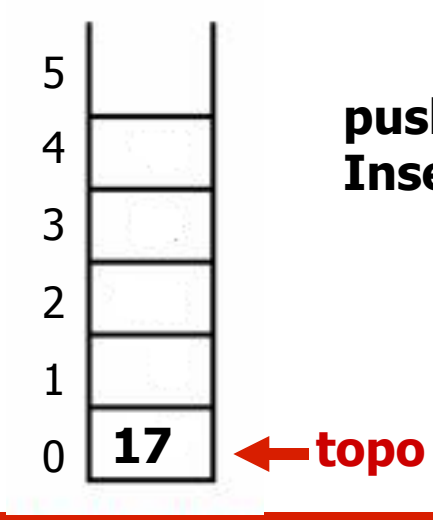
Tipos especiais de listas

- O armazenamento seqüencial (estático) é útil quando as estruturas **sofrem poucas remoções** ou inserções ou ainda quando **inserções e remoções não acarretam grande movimentação de nós**
  - Não é bem o caso da implementação de lista que permite remover e inserir um elemento em qualquer posição da lista...
  - Por isso, o armazenamento seqüencial é mais usado para implementar os tipos especiais de listas:
    - Filas (Queue em inglês),
    - Pilhas (Stack em inglês) e
    - Deques (Deque em inglês)
- São mais ferramentas do programador do que estruturas de armazenamento.

Pilha
(Stack)

Pilhas (stack)

- Os elementos são inseridos e removidos sempre em uma extremidade (a final) chamada de **topo da pilha**.
- Contém um ponteiro (variável) que marca o topo da pilha.
- Implementa norma: **LIFO** (last-in, first-out)
  - último a entrar, primeiro a sair.
- Pilha Vazia: topo=-1

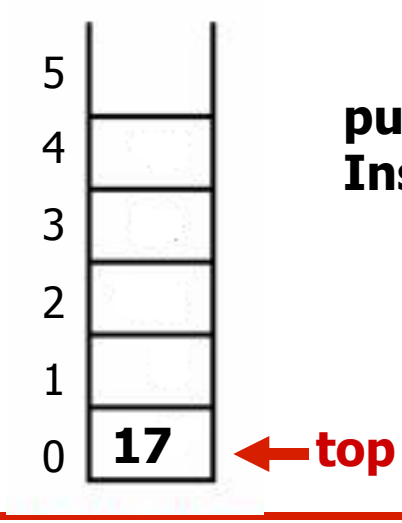


push (17)
Inserção de 17 no topo da pilha

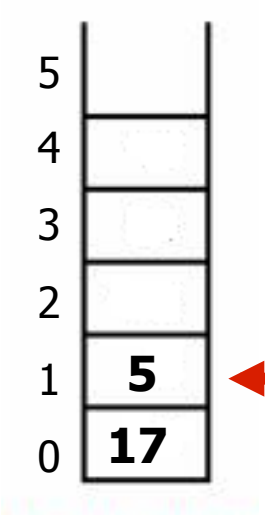
Pilhas (Stack)

- Operações sobre Pilhas:
  - Verificar se a pilha está vazia
  - Verificar se a pilha está cheia
  - Inserir um elemento no topo da pilha
  - Remover um elemento do topo da pilha
  - Inspecionar o elemento do topo da pilha

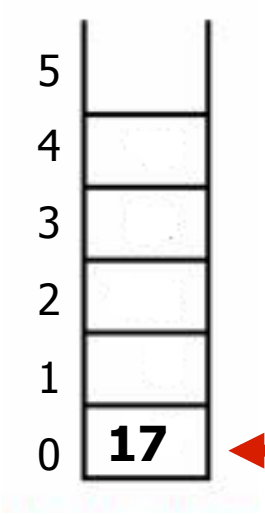
Pilhas



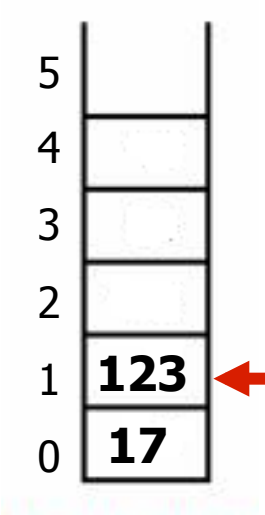
push (17)
Inserção de 17 no topo da pilha



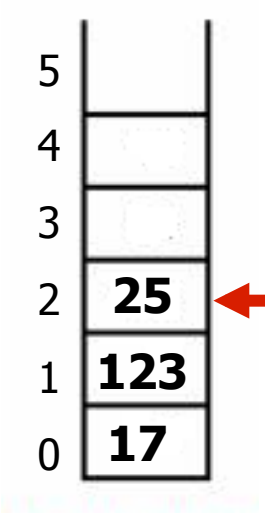
push(5)
Inserção de 5 no topo da pilha



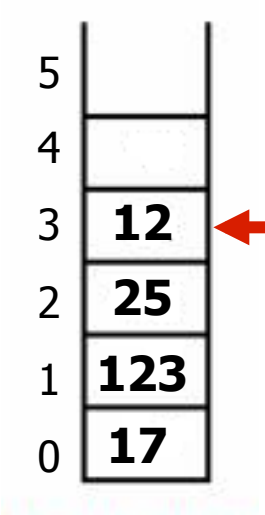
pop()
Remoção (sempre no topo da pilha)



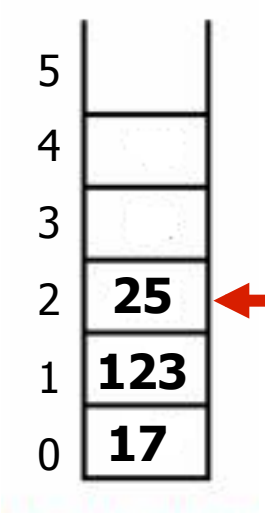
push(123)
Inserção de 123 no topo da pilha



push(25)
Inserção de 25 no topo da pilha



push(12)
Inserção de 12 no topo da pilha



pop()
Remoção (sempre no topo da pilha)

- Estouro de pilhas:
  - **Estouro negativo (underflow)**: pilha vazia sofre operação de extração
  - **Estouro positivo (overflow)**: quando a inserção de um elemento excede a capacidade total da pilha.

Fila
(Queue)

Filas (Queue)

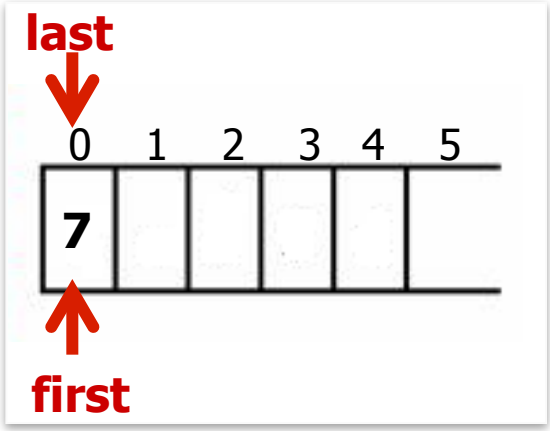
- Elementos são inseridos no fim da fila e retirados do início da fila.
- Geralmente, a implementação, contém 2 ponteiros (variáveis):
  - Um ponteiro para o início da fila (**first**)
  - Um ponteiro para o fim da fila (**last**)
- **FIFO** (first-int, first-out)
  - primeiro a entrar, primeiro a sair
- Fila vazia:
  - **last=first=-1**;

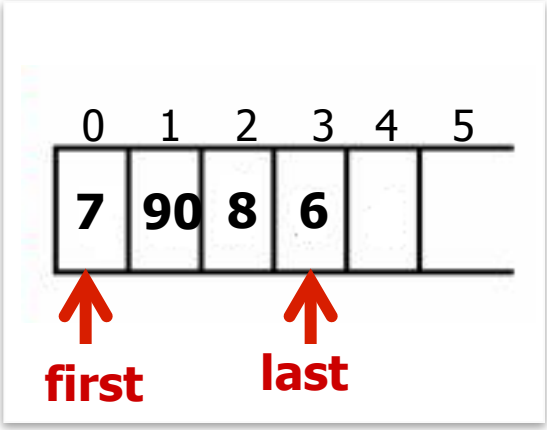
Underflow: fila vazia sofre operação de extração
Overflow: quando a inserção de um elemento excede a capacidade total da fila.

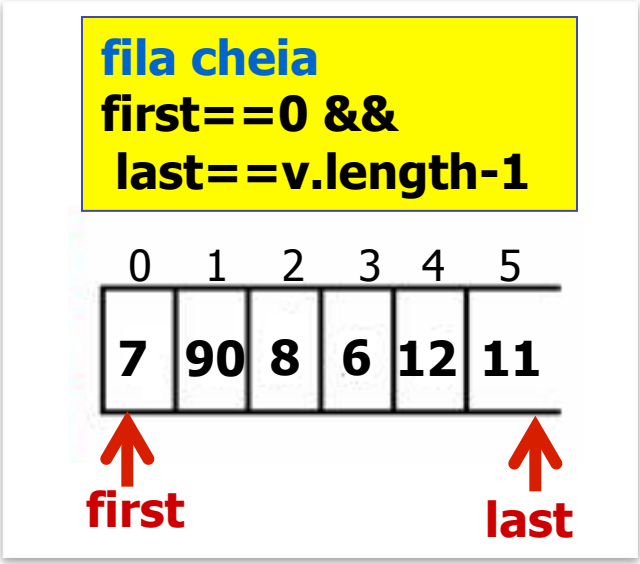
Operações sobre Filas (Queue)

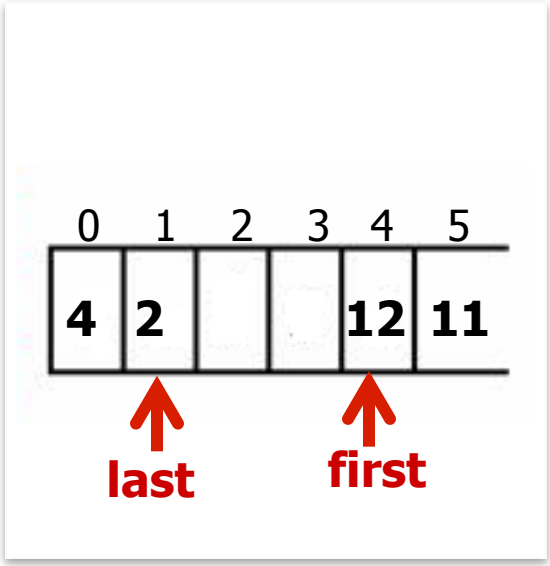
- Verificar se a fila está vazia
- Inserir um elemento no final da fila
- Remover e retornar o elemento do início da fila
- Consultar o elemento do início da fila
- Obter o tamanho da fila

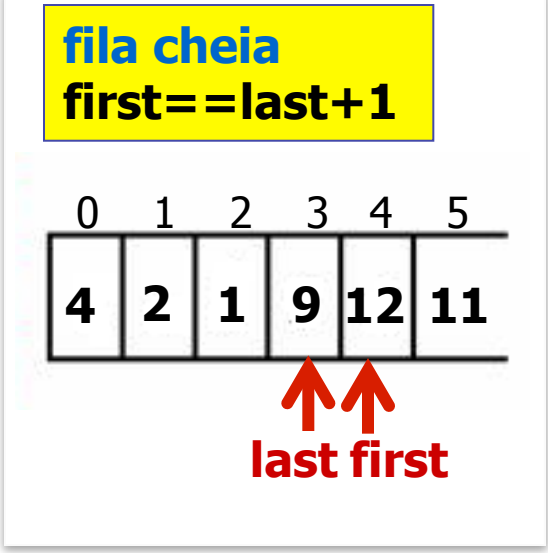
Operações da Fila



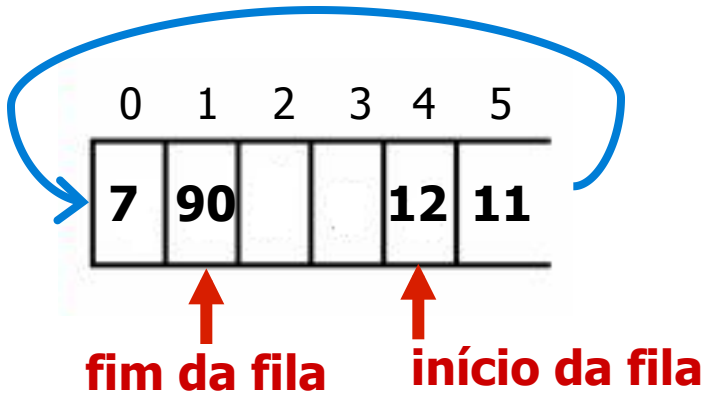








Fila com implementação circular



Interface de Pilha (Stack)

Interface Stack<E>

```
public interface Stack<E> {

        /** Return the number of elements in the stack. */
        public int numElements();

        /** Return whether the stack is empty. */
        public boolean isEmpty();

        /** Return whether the stack is full. */
        public boolean isFull();

        /** Inspect the element at the top of the stack.*/
        public E top() throws UnderflowException;

        /**Insert an element at the top of the stack.*/
        public void push (E element) throws OverflowException;

        /** Remove the top element from the stack.*/
        public E pop() throws UnderflowException;
}
```

Classe de Pilha (StaticStack)

Classe StaticStack

```
public class StaticStack<E> implements Stack<E> {

   // Índice do elemento no topo da pilha
   protected int top;

   // Array que armazena as referências para os elementos
   protected E elements[];

   // Constroi uma pilha com um tamanho máximo
   public StaticStack(int maxSize) {
        elements = (E[])new Object[maxSize];
        top = -1;
   }
```

Interface de Fila (Queue)

Interface Queue

```
public interface Queue<E> {
  /** Returns the number of elements in the queue.*/
  public int numElements();
  /** Returns whether the queue is empty. */
  public boolean isEmpty();
  /** Returns whether the queue is full. */
  public boolean isFull();
  /** Inspects the element at the front of the queue.*/
  public E front() throws UnderflowException;
  /** Inspects the element at the back of the queue.*/
  public E back() throws UnderflowException;
  /** Inserts an element at the rear of the queue. */
  public void enqueue (E element) throws OverflowException;
  /** Removes the element at the front of the queue.*/
  public E dequeue() throws UnderflowException;
}
```

Classe de Fila (StaticQueue)

Classe StaticQueue

```
public class StaticQueue<E> implements Queue<E>{

        //Index to the first element
        protected int first;

        //Index to the last element
        protected int last;

        // Generic array used to store the elements
        protected E elements[];

        public StaticQueue(int maxSize) {
                elements = (E[]) new Object[maxSize];
                first = last = -1;
        }
```

Testando a Pilha

```
public class StaticStackTest {
        public static void main(String[] args) {
                StaticStack<Integer> s = new StaticStack<Integer>(10);
                try{
                        s.push(1);
                        s.push(2);
                        s.push(3);
                        s.push(4);
                        s.push(5);
                } catch(OverflowException e) {
                        System.out.println(e);
               }

                try{
                        while (!s.isEmpty()) {
                                System.out.println(s.pop());
                        }
                }
                catch(UnderflowException e){
                        System.out.println(e);
                }
        }
}
```

Testando a Fila

```
public class StaticQueueTest {
        public static void main(String[] args) {
                StaticQueue<Integer> q = new StaticQueue<Integer>(5);
                try{
                        q.enqueue(1);
                        q.enqueue(2);
                        q.enqueue(3);
                        q.enqueue(4);

                        q.dequeue();
                        q.dequeue();

                        q.enqueue(5);
                        q.enqueue(6);
                        q.enqueue(7);
                } catch(OverflowException e) {
                        System.out.println(e);
                }
                catch(UnderflowException e) {
                        System.out.println(e);
                }
        }
}
```

Exercícios

- Complete a implementação das classes StaticStack e StaticQueue
  - Inclua em cada uma destas classes o método toString(), que deve retornar uma String que representa a estrutura de dados.
    - Para a pilha, o toString deve retornar uma String que contém os elementos começando do topo, **um abaixo do outro**
    - Para a fila, o toString deve retornar uma String que contém os elementos começando no first e terminando no last, **um ao lado do outro**