Tipos Especiais de Listas

Pilha
Fila
Deque

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Tipos especiais de listas

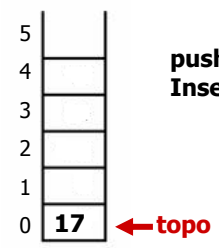
- O armazenamento seqüencial (estático) é útil quando as estruturas **sofrem poucas remoções** ou inserções ou ainda quando **inserções e remoções não acarretam grande movimentação de nós**
  - Não é bem o caso da implementação de lista que permite remover e inserir um elemento em qualquer posição da 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)

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

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



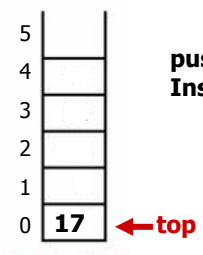
push (17)
Inserção de 17 no topo da pilha

Pilhas (Stack)

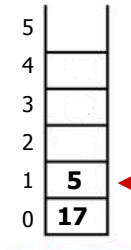
- Operações sobre Pilhas:
  - Verificar se a pilha está vazia
  - Verificar se a pilha está cheia
  - Inserir um elemento no topo da pilha
  - Remover um elemento do topo da pilha
  - Inspecionar o elemento do topo da pilha

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

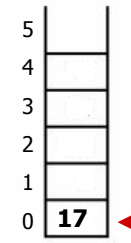
Pilhas



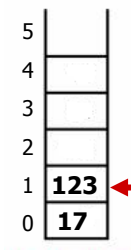
push (17)
Inserção de 17 no topo da pilha



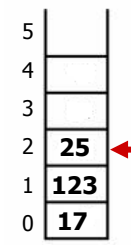
push(5)
Inserção de 5 no topo da pilha



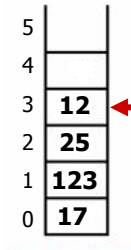
pop()
Remoção (sempre no topo da pilha)



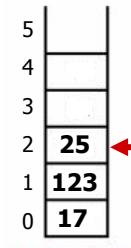
push(123)
Inserção de 123 no topo da pilha



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.

Interface Stack<E>

```
public interface Stack<E> {

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

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

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

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

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


Estouro da Pilha

```
public class EmptyStackException extends RuntimeException {
  public EmptyStackException(String err) {
    super(err);
  }
}
```

```
  public class FullStackException extends RuntimeException {
    public FullStackException(String err) {
      super(err);
    }
  }
```

Classe ArrayStack

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

    // Capacity of the stack array
    protected int capacity;

   // default array capacity
    public static final int CAPACITY = 1000;

    // Generic array used to implement the stack
    protected E S[];

    // index for the top of the stack
    protected int top = -1;

     /**Initializes the stack to use an array of default length.*/
     public ArrayStack() {
        this(CAPACITY); // default capacity
     }
```


Classe ArrayStack (cont.)

```
  /** Initializes the stack to use an array of given length.*/
  public ArrayStack(int cap) {
      capacity = cap;
      S = (E[]) new Object[capacity];
  }

  /**Returns the number of elements in the stack. */
  public int size() {
   return (top + 1);
  }

  /**Testes whether the stack is empty. */
  public boolean isEmpty() {
   return (top < 0);
  }
```

Classe ArrayStack (cont.)

```
/** Inserts an element at the top of the stack. */
 public void push(E element) throws FullStackException {
   if (size() == capacity)
     throw new FullStackException("Stack is full.");
   S[++top] = element;
 }

 /** Inspects the element at the top of the stack. */
 public E top() throws EmptyStackException {
   if (isEmpty())
     throw new EmptyStackException("Stack is empty.");
   return S[top];
 }
```


Classe ArrayStack (cont.)

```
/** Removes the top element from the stack. */

  public E pop() throws EmptyStackException {
    E element;
    if (isEmpty())
      throw new EmptyStackException("Stack is empty.");
    element = S[top];
    S[top--] = null; // dereference S[top] for garbage collection.
    return element;
  }
}
```

Testando a Pilha

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

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


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:
  - Quando **last==first-1**;

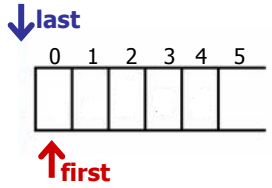
Underflow: fila vazia sofre operação de extração
Overflow: quando a inserção de um elemento excede a capacidade total da fila.

Operações sobre Filas (Queue)

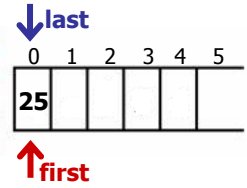
- Verificar se a fila está vazia
- Inserir um elemento no final da fila
- Remover e retornar o elemento do inicio da fila
- Consultar o elemento do inicio da fila
- Obter o tamanho da fila

Filas

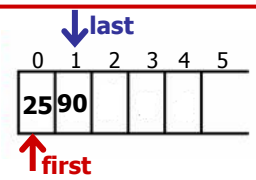
Início: Fila vazia



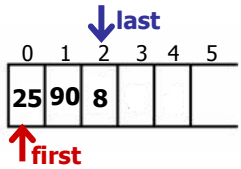
enqueue (25)
Inserção de 25 na Fila



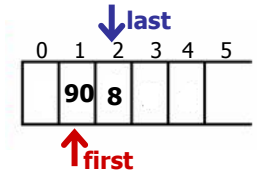
Filas



enqueue(90)
Inserção de 90 na fila

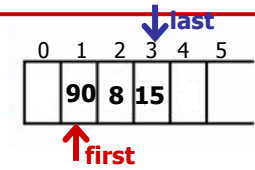


enqueue(8)
Inserção de 8 na fila

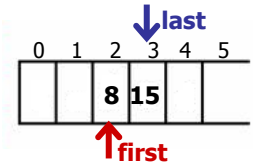


dequeue()
Remoção na fila

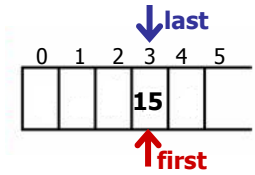
Fila



enqueue(15)
Inserção de 15 na fila

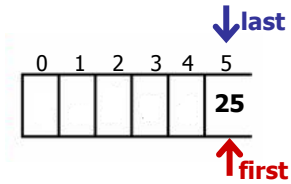


dequeue()
Remoção



dequeue()
Remoção

Fila



dequeue()
Remoção

Fila está cheia???

Interface Queue

```
public interface Queue<E> {

  /** Returns the number of elements in the queue.*/
  public int size();

  /** Returns whether the queue is empty. */
  public boolean isEmpty();

  /** Inspects the element at the front of the queue.*/
  public E front() throws EmptyQueueException;

  /** Inserts an element at the rear of the queue. */
  public void enqueue (E element);

  /** Removes the element at the front of the queue.*/
  public E dequeue() throws EmptyQueueException;
}
```


Classe Queue

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

        // The actual capacity of the queue array
        protected int capacity;

        //Index to the first element
        protected int first = 0;

        //Index to the last element
        protected int last = -1;

        // default array capacity
        public static final int CAPACITY = 1000;

        // Generic array used to implement the queue
        protected E Q[];
```

Classe Queue (cont.)

```
/**Initializes the queue to use an array of default length. */
 public ArrayQueue() {
        this(CAPACITY);
 }

 /** Initializes the queue to use an array of given length. */
 public ArrayQueue(int cap){
        capacity = cap;
        Q = (E[]) new Object[capacity];
 }

 /** Testes whether the queue is empty. */
 public boolean isEmpty() {
        return (last == (first -1));
 }
```


Classe Queue (cont.)

```
/** Testes whether the queue is empty. */
    public boolean isEmpty() {
        return (last == (first -1));
    }

    /** Returns the number of elements in the queue. */
    public int size() {
        return ((last - first) + 1);
    }

    /** Removes and return the element at the front of the queue.*/
    public E dequeue() throws EmptyQueueException {
        if (isEmpty()) {
            throw new EmptyQueueException("Queue is empty.");
        }
        E temp = Q[first];
        first++;
        return temp;
    }
```

Classe Queue (cont.)

```
    /** Inserts an element at the rear of the queue. */
        public void enqueue(E element) throws FullQueueException {
            if (last == Q.length -1) {
                throw new FullQueueException("Queue is full.");
            }
            last++;
            Q[last] = element;
        }

        /** Inspects the element at the front of the queue.*/
        public E front() throws EmptyQueueException {
            if (isEmpty()) {
                throw new EmptyQueueException("Queue is empty.");
            }
            return Q[first];
        }
}
```

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Classes de Exceção

```
public class FullQueueException extends RuntimeException {
        public FullQueueException(String err) {
           super(err);
         }
}
```

```
public class EmptyQueueException extends RuntimeException {
   public EmptyQueueException(String err) {
   super(err);
  }
 }
```

Testando a Fila

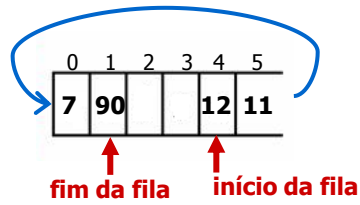
```
public class ArrayQueueTest {
	public static void main(String[] args) {
		ArrayQueue<Integer> q = new ArrayQueue<Integer>(5);
		try{
			q.enqueue(1);
			q.enqueue(2);
			q.enqueue(3);
			q.enqueue(4);
			q.enqueue(5);
			System.out.println(q);
		}catch(FullQueueException e) {
			System.out.println(e);
		}

		try{
			while (!q.isEmpty()){
				System.out.println(q.dequeue());
			}
		}
		catch(EmptyQueueException e) {
			System.out.println(e);
		}
	}
}
```


Fila Circular
(Circular Queue)

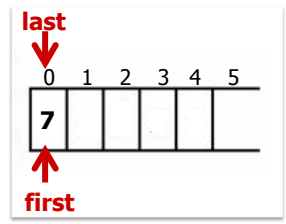
Fila com implementação circular

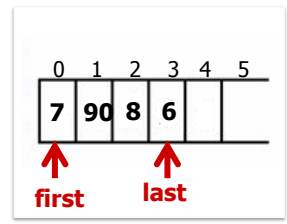
- Na implementação anterior, observamos que a fila pode ser considerada como cheia, mesmo não contendo nenhum elemento.
- Isso se deve a forma como lidamos com os índices que controlam o início e final da fila.
- Uma solução para esse problema é trabalhar com filas circulares.

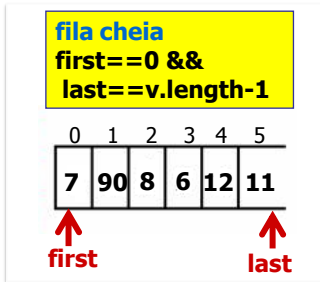


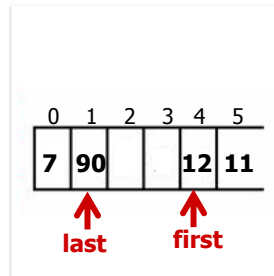
UNISINOS

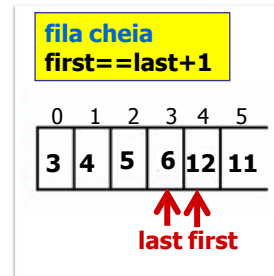
Fila com Implementação Circular











Classe ArrayCircularQueue

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

        // The actual capacity of the queue array
        protected int capacity;

        //Index to the first element
        protected int first = -1;

        //Index to the last element
        protected int last = -1;

        // default array capacity
        public static final int CAPACITY = 1000;

        // Generic array used to implement the queue
        protected E Q[];

        /**Initializes the queue to use an array of default length. */
        public ArrayCircularQueue() {
                this(CAPACITY);
        }
```


Classe ArrayCircularQueue (cont.)

```
/** Initializes the queue to use an array of given length. */
        public ArrayCircularQueue(int cap){
                capacity = cap;
                Q = (E[]) new Object[capacity];
        }

        /** Testes whether the queue is empty. */
        public boolean isEmpty() {
                if (first == -1) {
                        return true;
                }
                else {
                        return false;
                }
        }
```

Classe ArrayCircularQueue (cont.)

```
/** Returns the number of elements in the queue. */
        public int size() {
                if (first > last) {
                        return (Q.length - first) + (last + 1);
                }
                else {
                        return (last - first + 1);
                }
        }

/** Inspects the element at the front of the queue.*/
        public E front() throws EmptyQueueException {
                if (isEmpty())
                        throw new EmptyQueueException("Queue is empty.");
                return Q[first];
        }
```


Classe ArrayCircularQueue (cont.)

```
/** Inserts an element at the rear of the queue. */
        public void enqueue(E element) throws FullQueueException {
                if ((first == 0 && last == Q.length -1) ||
                                (first == last + 1)) {
                        throw new FullQueueException("Queue is full");
                }
                else {
                        if ((last == Q.length -1) || (last == -1)) {
                                last = 0;
                                Q[last] = element;
                                if (first == -1) {
                                        first = 0;
                                }
                        }
                        else {
                                Q[++last] = element;
                        }
                }
        }
```

Classe ArrayCircularQueue (cont.)

```
/** Removes and return the element at the front of the queue.*/
        public E dequeue() throws EmptyQueueException {
                if (isEmpty()) {
                        throw new EmptyQueueException("Queue is Empty.");
                }

                E element = Q[first];

                if (first == last) { //Queue contains only one element
                        first = last = -1;
                }
                else if (first == Q.length -1){
                        first = 0;
                }
                else{
                        first++;
                }
                return element;
        }
}
```


Testando a Fila Circular

```
public class ArrayCircularQueueTest {
        public static void main(String[] args) {
                ArrayCircularQueue<Integer> q = new ArrayCircularQueue<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(FullQueueException e) {
                        System.out.println(e);
                }
                catch(EmptyQueueException e) {
                        System.out.println(e);
                }
        }
}
```

Exercício

- Implemente métodos para exibir o conteúdo da fila circular de 2 maneiras:
  1. No método main, faça o código necessário para que enquanto a fila não for vazia, esvazie a fila, exibindo os elementos retirados.
  2. Crie um método print na classe CircularQueue que exiba o conteúdo da fila na ordem em que foram inseridos

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Deque
(Deque)

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Deque

- O **deque** é um tipo de fila em que as inserções e remoções podem ser realizadas em ambas as extremidades, que chamaremos de frente (front) e final (back).

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Operações sobre Deques

- Verificar se o deque está vazio
- Verificar se o deque está cheio
- Inserir um elemento na inicio do deque
- Inserir um elemento no final do deque
- Remover um elemento do inicio do deque
- Remover um elemento do final do deque
- Retornar (sem remover) o primeiro elemento do deque
- Retornar (sem remover) o último elemento do deque

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Interface Deque

```
public interface Deque<E> {
 /** Returns the number of elements in the deque.*/
  public int size();

  /**  Returns whether the deque is empty.*/
  public boolean isEmpty();

  /** Returns the first element; an exception is thrown if deque is empty.*/
  public E getFirst() throws EmptyDequeException;

  /**Returns the last element; an exception is thrown if deque is empty.*/
  public E getLast() throws EmptyDequeException;

  /**Inserts an element to be the first in the deque.*/
  public void addFirst (E element);

  /**Inserts an element to be the last in the deque.*/
  public void addLast (E element);

  /** Removes the first element; an exception is thrown if deque is empty.*/
  public E removeFirst() throws EmptyDequeException;

  /** Removes the last element; an exception is thrown if deque is empty.*/
  public E removeLast() throws EmptyDequeException;
}
```


Classe ArrayDeque

```
public class ArrayDeque<E> extends ArrayQueue<E> implements Deque<E> {

        /**Initializes the deque to use an array of default length. */
        public ArrayDeque(){
                super();
        }

        /** Initializes the deque to use an array of given length. */
        public ArrayDeque(int cap) {
                super(cap);
        }

        /**Inserts an element to be the first in the deque.*/
        public void addFirst(E element) throws FullDequeException{
                if (last == Q.length -1) {
                        throw new FullDequeException("Deque is full.");
                }
                System.arraycopy(Q, first, Q, first + 1, last - first + 1);
                Q[first] = element;
                last++;
        }
```

Classe ArrayDeque (cont.)

```
/**Inserts an element to be the last in the deque.*/
        public void addLast(E element)  throws FullDequeException{
                try{
                        enqueue(element);
                } catch(FullQueueException e){
                        throw new FullDequeException("Deque is full.");
                }
        }

        /** Returns the first element; an exception is thrown if deque is empty.*/
        public E getFirst() throws EmptyDequeException {
                return super.front();
        }

        /**Returns the last element; an exception is thrown if deque is empty.*/
        public E getLast() throws EmptyDequeException {
                if (isEmpty()){
                        throw new EmptyDequeException("Deque is empty.");
                }
                return Q[last];
        }
```


Classe ArrayDeque (cont.)

```
/** Removes the first element; an exception is thrown if deque is empty.*/
        public E removeFirst() throws EmptyDequeException {
                return dequeue();
        }

        /** Removes the last element; an exception is thrown if deque is empty.*/
        public E removeLast() throws EmptyDequeException {
                if (isEmpty()) {
                        throw new EmptyDequeException("Deque is Empty.");
                }
                return Q[last--];
        }

}
```

Testando a classe ArrayDeque

```
public class ArrayDequeTest {
        public static void main(String[] args) {
                ArrayDeque<Integer> d = new ArrayDeque<Integer>();
                try{
                        d.addLast(1);
                        d.addFirst(2);
                        d.addLast(3);
                        d.addFirst(4);

                        System.out.println(d.removeFirst());
                        System.out.println(d.removeLast());
                        System.out.println(d.removeFirst());
                        System.out.println(d.removeLast());
                }
                catch(FullDequeException e) {
                        System.out.println(e);
                }
                catch (EmptyDequeException e){
                        System.out.println(e);
                }
        }
}
```