Tipos Especiais de Listas

Pilha

Fila

Implementação com Alocação Dinâmica de Memória

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

Pilha (Stack)

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

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);
  }
}
```

Classe NodeStack

```
public class NodeStack<E> implements Stack<E> {
  protected Node<E> top;// reference to the head node
  protected int size;// number of elements in the stack

  public NodeStack() {// constructs an empty stack
    top = null;
    size = 0;
  }
  public int size() { return size; }

  public boolean isEmpty() {
    if (top == null) return true;
    return false;
  }
```


Classe NodeStack (cont.)

```
  public void push(E elem) {
     Node<E> v = new Node<E>(elem, top);// create and link-in a new node
     top = v;
     size++;
   }

   public E top() throws EmptyStackException {
     if (isEmpty()) throw new EmptyStackException("Stack is empty.");
     return top.getElement();
   }

   public E pop() throws EmptyStackException {
     if (isEmpty()) throw new EmptyStackException("Stack is empty.");
     E temp = top.getElement();
     top = top.getNext();// link-out the former top node
     size--;
     return temp;
   }
}
```

Testando a Pilha

```
public class ArrayStackTest {
        public static void main(String[] args) {
            NodeStack<Integer> s = new NodeStack<Integer>();

            s.push(1);
            s.push(2);
            s.push(3);
            s.push(4);
            s.push(5);

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


Exercício

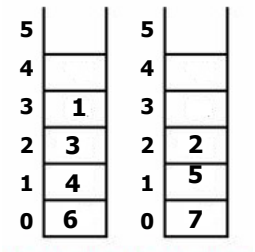
Implemente na classe NodeStack o método toString()

Resposta do Exercício

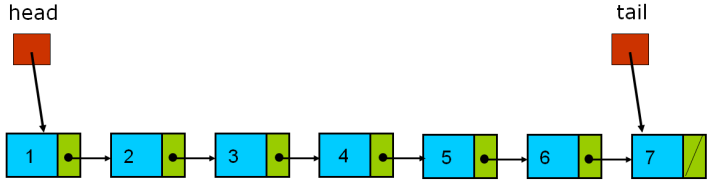
```
public String toString() {
                String s;
                Node<E> cur = null;
                s = "[";
                int n = size();
                if (n > 0) {
                        cur = top;
                        s += cur.getElement();
                }
                if (n > 1)
                        for (int i = 1; i <= n-1; i++) {
                                cur = cur.getNext();
                                s += ", " + cur.getElement();
                        }
                s += "]";
                return s;
        }
```


Exercício Extra de Desafio

- Dada 2 pilhas de inteiros ordenadas do topo para a base. Por exemplo:



- No método main de uma classe qualquer, faça um programa que crie uma fila com alocação dinâmica, tendo os elementos das 2 pilhas ordenados na nova fila. Os elementos podem ser removidos das pilhas.



Fila
(Queue)

UNISINOS
UNIVERSIDADE DO VALE DO RIO DOS SINOS

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 NodeQueue

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

  protected Node<E> head, tail;  // the head and tail nodes
  protected int size;  // Keeps track of number of elements in queue

/** Creates an empty queue. */
   public NodeQueue() {
            head = null;
            tail = null;
            size = 0;
   }

   public int size() {       // Return the current queue size
            return size;
   }

   public boolean isEmpty() { //Return true if the queue is empty
      if ( (head==null) && (tail==null) )
         return true;
     return false;
   }
```


Classe NodeQueue (cont.)

```
public void enqueue(E elem) {
            Node<E> node = new Node<E>();
            node.setElement(elem);
            node.setNext(null); // node will be new tail node
            if (size == 0)
                        head = node; // special case of a previously empty queue
            else
                        tail.setNext(node); // add node at the tail of the list
            tail = node; // update the reference to the tail node
            size++;
}

public E front() throws EmptyQueueException {
      if (size == 0)
                        throw new EmptyQueueException("Queue is empty.");
            return head.getElement();
}
```

Classe NodeQueue (cont.)

```
public E dequeue() throws EmptyQueueException {
        if (size == 0)
                throw new EmptyQueueException("Queue is empty.");
        E tmp = head.getElement();
        head = head.getNext();
        size--;
        if (size == 0)
                tail = null; // the queue is now empty
        return tmp;
    }
```


Classes de Exceção

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

Testando a Fila

```
public class ArrayQueueTest {
        public static void main(String[] args) {
                NodeQueue<Integer> q = new NodeQueue<Integer>();
                        q.enqueue(1);
                        q.enqueue(2);
                        q.enqueue(3);
                        q.enqueue(4);
                        q.enqueue(5);
                        System.out.println(q.toString());

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


Exercício

Implemente na classe NodeQueue o método toString()

Resposta do Exercício

```
public String toString() {
                String s = "";
                s += "[";
                if (!isEmpty()) {
                        Node<E> p = head;
                        do {
                                s += p.getElement() ;
                                if (p != tail)
                                        s += ", ";
                                p = p.getNext();
                        } while (p != null);
                }
                s += "]";
                return s;
        }
```


Exercícios

- Escreva um método na classe NodeQueue (Fila com alocação dinâmica simplesmente encadeada) que inverta o conteúdo da fila. Este método deve retornar um novo objeto fila com o conteúdo do objeto fila corrente invertido. Use uma pilha para auxiliar. A fila corrente não pode ter seus dados removidos.
- No método main de uma classe qualquer, usando qualquer uma das implementações de pilha fornecidas em aula (simplesmente encadeada), transfira os elementos da pilha s1 para a pilha s2 de modo que os elementos de s2 estejam na mesma ordem que em s1. Use uma pilha adicional.