28. [Sebesta, 2000] Escreva um tipo de dado abstrato para pilhas cujos elementos armazenem nomes de 10 caracteres. Os elementos da pilha devem ser alocados dinamicamente na *heap*. As operações da pilha são empilhar, desempilhar e esvaziar.

```
/**
 * Classe responsável pela representação do nodo
 */
public class Node
{
  /**
   * Elemento do nodo
   */
  private String element;

  /**
   * Próximo nodo
   */
  private Node next;

  /**
   * Constructor
   *
   * @param element elemento do nodo
   */
  public Node(final String element)
  {
    this(element, null);
  }

  /**
   * Constructor
   *
   * @param element elemento do nodo
   * @param next próximo nodo
   */
  public Node(final String element, final Node next)
  {
    if (element.length() > 10)
    {
      this.element = element.substring(0, 10);
    }
    else
    {
      this.element = element;
    }

    setNext(next);
  }

  /**
   * Retornar o elemento do nodo
   *
   * @return elemento do nodo
   */
  public String getElement()
  {
    return element;
  }
```

```
  /**
   * Retornar o próximo nodo
   *
   * @return próximo nodo
   */
  public Node getNext()
  {
    return next;
  }

  /**
   * Configurar o próximo nodo
   *
   * @param next próximo nodo
   */
  public void setNext(final Node next)
  {
    this.next = next;
  }
}

/**
 * Classe responsável pela especificação da pilha
 */
public class Stack
{
  /**
   * Primeiro nodo da pilha
   */
  private Node header;

  /**
   * Constructor
   */
  public Stack()
  {
    header = null;
  }

  /**
   * Remover todos os elementos da pilha
   */
  public void clear()
  {
    while (!empty())
    {
      pop();
    }
  }

  /**
   * Verificar se a pilha está vazia
   *
   * @return true se a pilha está vazia, false em caso contrário
   */
  public boolean empty()
  {
    return header == null;
  }
```

```
/**
 * Recuperar o elemento no topo da pilha,
 * ou retornar null se a pilha estiver vazia
 *
 * @return elemento no topo da pilha, ou null se a pilha estiver vazia
 */
public String top()
{
  if (!empty())
  {
    return header.getElement();
  }

  return null;
}

/**
 * Recuperar e remover o elemento no topo da pilha,
 * ou retornar null se a pilha estiver vazia
 *
 * @return elemento no topo da pilha, ou null se a pilha estiver vazia
 */
public String pop()
{
  if (!empty())
  {
    var node = header;

    header = header.getNext();

    node.setNext(null);

    return node.getElement();
  }

  return null;
}

/**
 * Inserir o elemento especificado no topo da pilha
 *
 * @param element elemento especificado
 */
public void push(final String element)
{
  if (empty())
  {
    header = new Node(element);
  }
  else
  {
    header = new Node(element, header);
  }
}
```

```
  /**
   * Retornar o número de elementos na pilha
   *
   * @return número de elementos na pilha
   */
  public int size()
  {
    var node = header;

    var count = 0;

    while (node != null)
    {
      node = node.getNext();

      count = count + 1;
    }

    return count;
  }
}



/**
 * Classe responsável pela execução da aplicação
 */
public class Application
{
  /**
   * Construtor padrão
   */
  private Application()
  {

  }

  /**
   * Método principal da linguagem de programação Java
   *
   * @param args argumentos da linha de comando (não utilizado)
   */
  public static void main(String[] args)
  {
    System.out.println("Stack Test");

    final var stack = new Stack();

    System.out.println("empty: " + stack.empty());

    System.out.println("push : Fulano");
    stack.push("Fulano");

    System.out.println("empty: " + stack.empty());

    System.out.println("pop  : " + stack.pop());

    System.out.println("push : Ciclano");
    stack.push("Ciclano");
```

```
    System.out.println("top : " + stack.top());

    System.out.println("push : Beltrano");
    stack.push("Beltrano");

    System.out.println("push : Tiburcio");
    stack.push("Tiburcio");

    System.out.println("size : " + stack.size());

    System.out.println("pop  : " + stack.pop());

    System.out.println("pop  : " + stack.pop());

    System.out.println("pop  : " + stack.pop());

    System.out.println("empty: " + stack.empty());
  }
}
```