27. [Sebesta, 2000] Escreva um tipo de dado abstrato para filas cujos elementos armazenem nomes de 10 caracteres. Os elementos da fila devem ser alocados dinamicamente do *heap*. As operações da fila são enfileirar, desenfileirar e esvaziar. Use o Modula-2, a Ada, o C++ ou o Java.

```
import java.io.Serializable;

/**
 * Nodo de uma lista simplesmente encadeada de inteiros
 */
public class Node implements Serializable
{
  /**
   * Identificador de serializacao da classe
   */
  private static final long serialVersionUID = 1L;

  /**
   * Elemento deste nodo
   */
  private final int element;

  /**
   * Proximo elemento deste nodo
   */
  private Node next;

  /**
   * Criar um nodo com um dado elemento e o proximo nodo
   *
   * @param element elemento deste nodo
   * @param next proximo elemento deste nodo
   */
  public Node(final int element, final Node next)
  {
    this.element = element;

    this.next = next;
  }

  /**
   * Retornar o elemento deste nodo
   *
   * @return elemento deste nodo
   */
  public int getElement()
  {
    return element;
  }

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

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

```
/**
 * Classe responsavel pela implementacao da fila,
 * utilizando uma lista simplesmente encadeada de inteiros
 */
public class Queue
{
  /**
   * Referencia para o nodo armazenado no inicio da fila
   */
  private Node header;

  /**
   * Construtor para inicializar a fila
   */
  public Queue()
  {
    header = null;
  }

  /**
   * Verificar se a fila esta vazia
   *
   * @return true se a fila esta vazia, false em caso contrario
   */
  public boolean empty()
  {
    return header == null;
  }

  /**
   * Inserir um novo elemento no fim da fila
   *
   * @param @param element novo elemento
   */
  public void offer(final int element)
  {
    if (empty())
    {
      header = new Node(element, header);
    }
    else
    {
      Node node = header;

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

```
    }

    node.setNext(new Node(element, null));
  }
}

/**
 * Recuperar o elemento do inicio da fila,
 * ou retornar zero caso a fila esteja vazia
 *
 * @return elemento do inicio da fila, ou zero caso a fila esteja vazia
 */
public int peek()
{
  if (!empty())
  {
    return header.getElement();
  }

  return 0;
}

/**
 * Recuperar e remover o elemento do inicio da fila,
 * ou retornar zero caso a fila esteja vazia
 *
 * @return elemento do inicio da fila, ou zero caso a fila esteja vazia
 */
public int poll()
{
  if (!empty())
  {
    final Node node = header;

    header = header.getNext();

    node.setNext(null);

    return node.getElement();
  }

  return 0;
}

/**
 * Retornar a quantidade de elementos na fila
 *
 * @return quantidade de elementos na fila
 */
public int size()
{
  Node node = header;

  int count = 0;

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

```
      count = count + 1;
    }

    return count;
  }

  @Override
  public String toString()
  {
    final StringBuilder out = new StringBuilder();

    out.append("[");

    Node node = header;

    while (node != null)
    {
      out.append(node.getElement());

      if (node.getNext() != null)
      {
        out.append(", ");
      }

      node = node.getNext();
    }

    out.append("]");

    return out.toString();
  }
}
```