20.7 *Lab:* Teletext

Figure 20-12 shows a snapshot from the *Teletext* program.   The program continuously scrolls up a list of headlines. The user can add a headline by typing it in the provided text input field. The line will be added after the blank line that follows "Today's headlines." The user can also delete a headline by entering d. The next headline after the top one will be deleted.   Run *Teletext* by clicking on the Teletext.jar file in J_M\Ch20\Teletext.
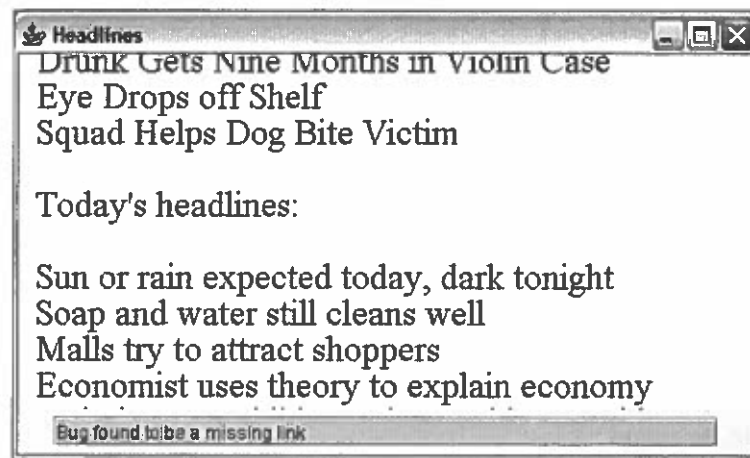

Figure 20-12. A snapshot from the *Teletext* program

Figure 20-13 shows the class diagram for the *Teletext* program. The program keeps the headlines in a doubly-linked circular list. (Circular lists are not used very often; this is a rare occasion where we can benefit from one.) The list is implemented in the TeletextList class.   Your task is to fill in the missing code in that class. TeletextList.java is located in J_M\Ch20\Teletext.
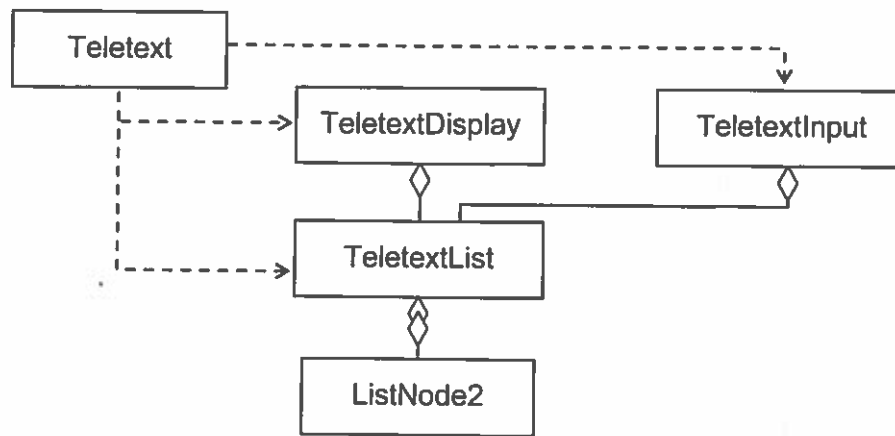
Figure 20-13. Class diagram for the *Teletext* program

20.8 Summary

In a *singly-linked list*, each element is stored in a larger structure, called a *node*. In addition to the element's value, a node contains a reference to the next node. The `next` reference in the last node is set to `null`. If `head` is a reference to the first node in the list, the statement

```
head = new ListNode(value, head);
```

appends a node holding `value` at the beginning of the list.

It is easy to traverse a linked list using a simple `for` loop:

```
for (ListNode node = head; node != null; node = node.getNext())
{
  Object value = node.getValue();
  ...
}
```

It is also not very hard to write a class that implements an "iterable" linked list for which iterators and "for each" loops work.

In a singly-linked list, inserting a node at the beginning takes $O(1)$ time, but appending a node at the end takes $O(n)$ time, where n is the number of nodes in the list. A *linked list with a tail* remedies this situation: it keeps a reference to the last