European Funds
Knowledge Education Development
Wrocław University of Science and Technology
European Union
European Social Fund

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

Laboratory – List 2

Introduction

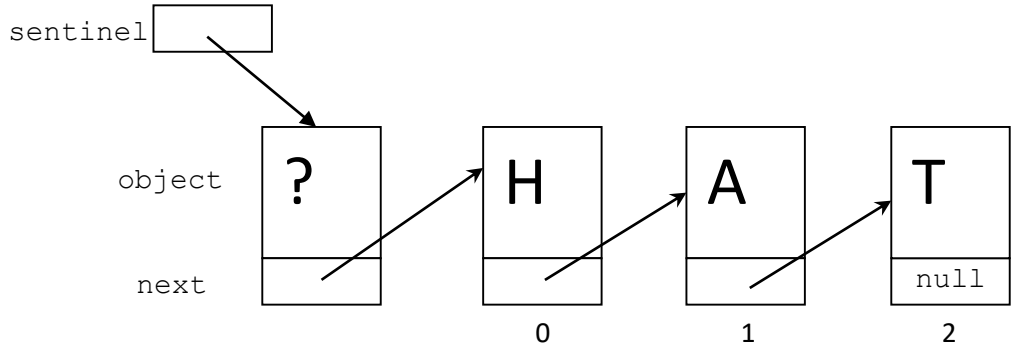
For implementation of the lists it will used below interface:

```
interface IList<E> extends Iterable<E> {
        boolean add(E e); // qdd element to the end of list
        void add(int index, E element) throws NoSuchElementException; // add element on position index
        void clear(); // delete all elements
        boolean contains(E element); // is list containing an element (equals())
        E get(int index) throws NoSuchElementException; //get element from position
        E set(int index, E element) throws NoSuchElementException; // set new value on position
        int indexOf(E element); // where is element (equals())
        boolean isEmpty();
        Iterator<E> iterator();
        ListIterator<E> listIterator() throws UnsupportedOperationException; // for ListIterator
        E remove(int index) throws NoSuchElementException; // remove element from position index
        boolean remove(E e); // remove element
        int size();
}
```

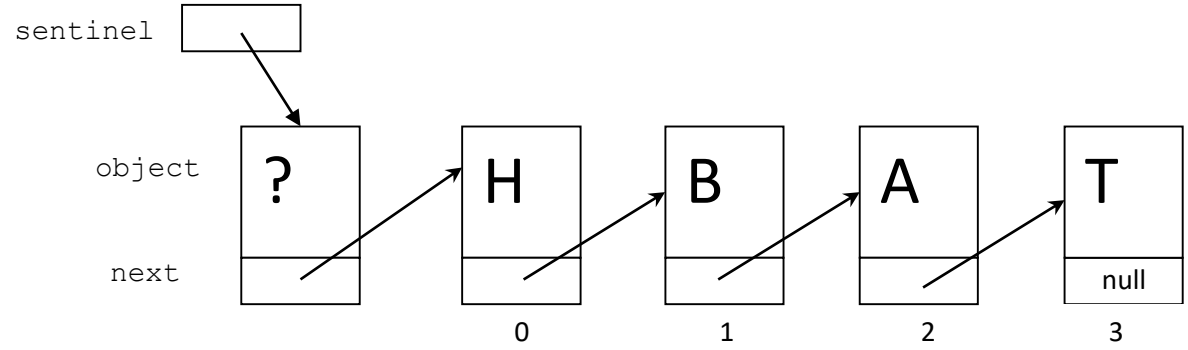
In this task you have to implement one way unordered list with a sentinel. Sentinel is an extra element, which is not visible by the user of the list, but simplifies the implementation. The sentinel will be the first element in the list. There list have to be one way list, so you have to have a handle to the first element (sentinel) – the sentinel field and in every element beside the stored object it have to be a reference to a next element. So the begin of the class would be as follows:

```
class OneWayLinkedList<E> implements IList<E>{

      private class Element{
            public Element(E e) {
                  this.object=e;
            }
            E object;
            Element next=null;
      }

      Element sentinel;
```

The element index in the list starts from 0, but without taking into consideration the sentinel element:



It is means, that after insertion an element "B" on the position 1 we have to insert it between "H" and "A":



The good implementation of the list have to be done as a generic class.

More details of the expected skeleton of object are presented in template files: `IList.java`, `OneWayLinkedList.java`.

Task list

1. Implement `IList<E>` interface without method `listIterator` for a class `OneWayLinkedList<E>`. Use it for a list of object `Link` which will be stored in field `links` of the class `Document`. In object of class `Link` there is only one field – `ref` of the type `String`. Now the loading operation (method `void load(Scanner scan)`) will be done during construction of an object of the class `Document`. The idea of loading document was presented in previous list of tasks. More details about skeleton of the classes you can find in the template files `Document.java`, `Link.java`.
2. Your implementation of the classes `OneWayLinkedList<E>`, `Link`, `Document` will be tested with format specified in the appendix. Prepare 2-3 interesting tests using this format.

For 100 points present solutions for this list till Week 3.
For 80 points present solutions for this list till Week 4.
For 50 points present solutions for this list till Week 5.
After Week 5 the list is closed.

Appendix

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different documents, which there are created as the first operation in the test. Each document can be constructed separately.

If a line is empty or starts from `'#'` sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If a line has a format:
`go <nMax>`
`Main` program will create an array of *nMax* Documents (without creation the documents). The lists are numbered from 0 like an array of documents. Default current position is the number 0.

If a line has a format:
`ch <n>`
`Main` program will choose a position of a number `n`, and all the following functions will operate on this position in the table (most often on the `links` field of the selected object, it is ensured in the tests that such an object will be constructed earlier). There is *0<=n<nMax*.

If a line has a format:
`ld <name>`
`Main` program will call a constructor of a document with parameters `name` and `scan` for current position, in order to load the document according to the same rules as for the previous list. If there is a document in its current position, it will be overwritten. Detected links, in the order read from the document, are to be saved to the `links` list.

If a line has a format:
`add <str>`
`Main` program will call `add(new Link(str))` for the link list of the current document. The result of the addition will be printed on one line.

If a line has a format:
`addi <index> <str>`
`Main` program will call `add(index, new Link(str))` for the link list of the current document. It will print the result or "error" if the method throws an appropriate exception.

If a line has a format:
`get <index>`
`Main` program will call `get(index)` for the link list of the current document and write on the screen field `ref` from returned `Link` object or "`error`" if the method throws an appropriate exception.

If a line has a format:
`clear`
`Main` program will call `clear()` for the link list of the current document.

If a line has a format:
`set <index> <str>`

`Main` program will call `set(index, new Link(str))` for the link list of the current document and write on the screen field `ref` from returned `Link` object or "`error`" if the method throws an appropriate exception.

If a line has a format:
`index <str>`
`Main` program will call `index(str)` the link list of the current document and write on the screen returned value in one line.

If a line has a format:
`show`
`Main` program will write on the screen the result of caling `toString()` for the current document. The first line has to present text "`Document: <name>`", the rest of lines – every link in separated line.

If a line has a format:
`remi <index>`
`Main` program will call `remove(index)` for the link list of the current document and write on the screen field `ref` from returned `Link` object or "`error`" if the method throws an appropriate exception.

If a line has a format:
`rem <str>`
`Main` program will call `remove(str)` for the link list of the current document and write on the screen returned value in one line.

If a line has a format:
`size`
`Main` program will call `size()` for the link list of the current document and write on the screen returned value in one line.

If a line has a format:
`ha`
`Main` program will end the execution, writing as the last line "`END OF EXECUTION`". Every test ends with this line.

For example for input test:

```
go 10
ld doc1
link=a and link=b
write also link=c end finish.
eod
show
add d
show
rem a
remi 2
show
get 0
set 1 w
show
```

```
ch 1
ld doc2
eod
show
remi 0
index x
ha
```

The output have to be:

```
START
!go 10
!ld doc1
!show
Document: doc1
a
b
c
!add d
true
!show
Document: doc1
a
b
c
d
!rem a
true
!remi 2
d
!show
Document: doc1
b
c
!get 0
b
!set 1 w
c
!show
Document: doc1
b
w
!ch 1
!ld doc2
!show
Document: doc2
!remi 0
error
!index x
-1
!ha
END OF EXECUTION
```