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 3 (version 1.0) – for two weeks

Introduction

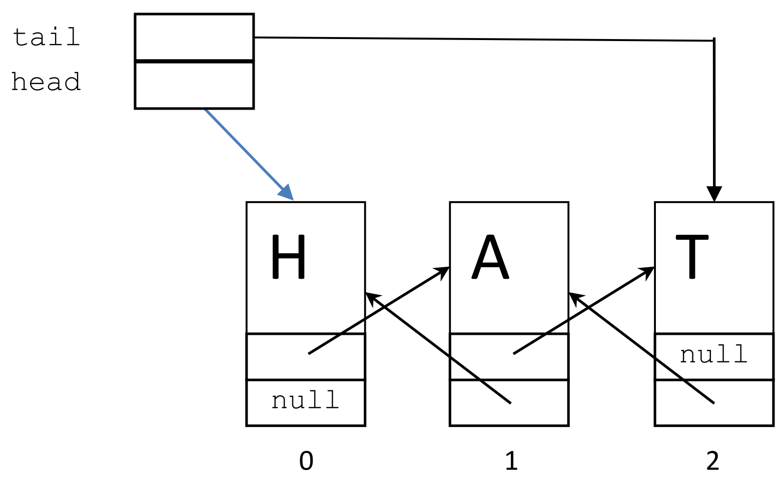
For implementation of the lists it will be used below interface:

```
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();
}
```

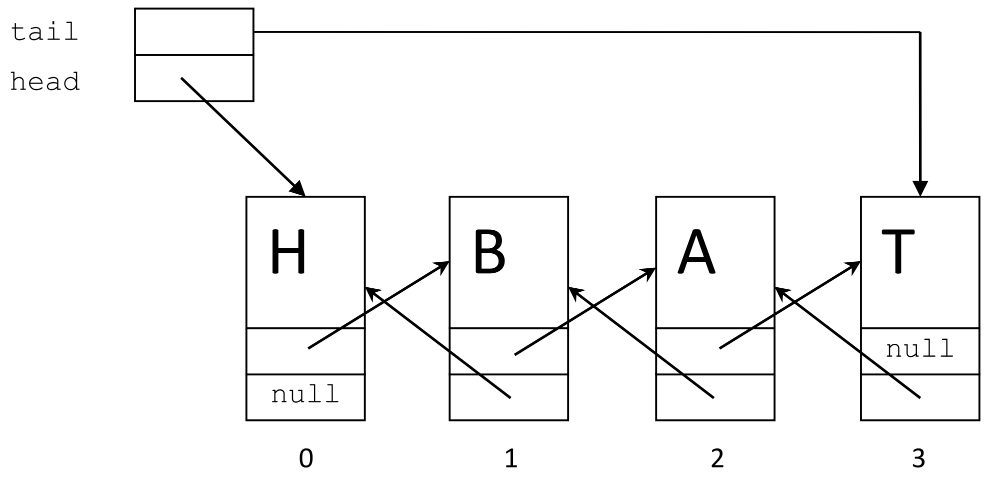
In this task you have to implement two way unordered list with a head and a tail. The reference in a field `head` has to point to the first element (or has to equals `null` for empty list) of the list, the reference in a field `tail` has to point to the last element (or has to equals `null` for empty list) of the list. Because the list is a two-way list, every element has to have two references `next` and `prev` which will point to a next element in the list and to a previous element in the list, respectively. So the begin of the class would be as follows:

```
class TwoWayUnorderedListWithHeadAndTail<E> implements IList<E>{
        private class Element{
                public Element(E e) {
                        this.object=e;
                }
                E object;
                Element next=null;
                Element prev=null;
        }
        Element head;
        Element tail;
…
}
```

The element index in the list starts from 0, and (because the list is unordered one) you can use indexes in many operation on a list link `add`, `remove` etc. Graphically the list can be presented in the following way:



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`, `TwoWayUnorderedListWithHeadAndTail.java`, `Link.java`.

Task list

1. Implement `IList<E>` interface without method `listIterator` for a class `TwoWayUnorderedListWithHeadAndTail<E>`. Don’t use a sentinel for this class. Use the list for a list of object `Link` which will be stored in object of the class `Document`. In object of class `Link` there is only one field – `ref` of the type `String`. 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 in previous list of tasks. In this list I propose to use a field `size`, but you can try to implement the list without it.
   a. Implement also a method: `void add(TwoWayUnorderedListWithHeadAndTail <E> anotherList)` which will add `another` list to the end of **`this`** list. After the operation the another list have to be empty. **The complexity of this method**

   have to be O(1). If the second list is the same list (the same reference!) – do nothing.
   b. Implement also a method:
   `String toStringReverse()` in which you have to use the references to previous elements in the list or `ListIterator` in a way presented in the template.

   If there is a code `throw new UnsupportedOperationException();` in the template you don't need implement the method.
2. Your implementation of the classes `TwoWayUnorderedListWithHeadAndTail<E>`, `Link`, `Document` will be tested with format specified in the appendix.

For 100 points present solutions for this list till Week 5.
For 80 points present solutions for this list till Week 6.
For 50 points present solutions for this list till Week 7.
After Week 7 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>`
your program has to 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>`
your program has to choose a position of a number `n`, and all next functions will operate on this position in the array. There is *0<=n<nMax*.

If a line has a format:
`ld <name>`
your program has to call a constructor of a document with parameters `name` and `scan` for current position. If there is a document on the current position it will be overwritten.

If a line has a format:
`add <str>`
your program has to call `add(new Link(str))` for the current document. The result of the addition has to be written in one line.

If a line has a format:
`addi <index> <str>`
your program has to call `add(index, new Link(str))` for current document. If the index is incorrect, write in one line "error".

If a line has a format:
`get <index>`
your program has to call `get(index)` for current document and write on the screen field `ref` from returned `Link` object. If the index is incorrect, write in one line "error".

If a line has a format:
`clear`
your program has to call `clear()` for current document.

If a line has a format:
`set <index> <str>`
your program has to call `set(index, new Link(str))` for current document and write on the screen field `ref` from returned `Link` object. If the index is incorrect, write in one line "`error`".

If a line has a format:
`index <str>`
your program has to call `index(str)` for current document and write on the screen returned value in one line.

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

If a line has a format:
`remi <index>`
your program has to call `remove(index)` for current document and write on the screen field `ref` from returned `Link` object. If the index is incorrect, write in one line "`error`".

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

If a line has a format:
`size`
your program has to write on screen the result of `size()` for current document and write on the screen returned value in one line.

If a line has a format:
`addl <listNo>`
your program has to call `add(secondList)` for current document and the second list it has to be a document of the number `listNo` from the array of documents.

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

The example from previous task is also proper for this list of tasks. A simple test for adding two list is as follow:

```
go 10
ld zero
text link=a and link=b
```

```
eod
show
reverse
ch 1
ld first
correct link=xxx or link=yyy
eod
show
reverse
addl 0
show
reverse
ch 0
show
reverse
ha
```

The output have to be:

```
START
!go 10
!ld zero
!show
Document: zero
a
b
!reverse
Document: zero
b
a
!ch 1
!ld first
!show
Document: first
xxx
yyy
!reverse
Document: first
yyy
xxx
!addl 0
!show
Document: first
xxx
yyy
a
b
!reverse
Document: first
b
a
yyy
xxx
!ch 0
!show
Document: zero
!reverse
Document: zero
!ha
END OF EXECUTION
```