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 4

Introduction

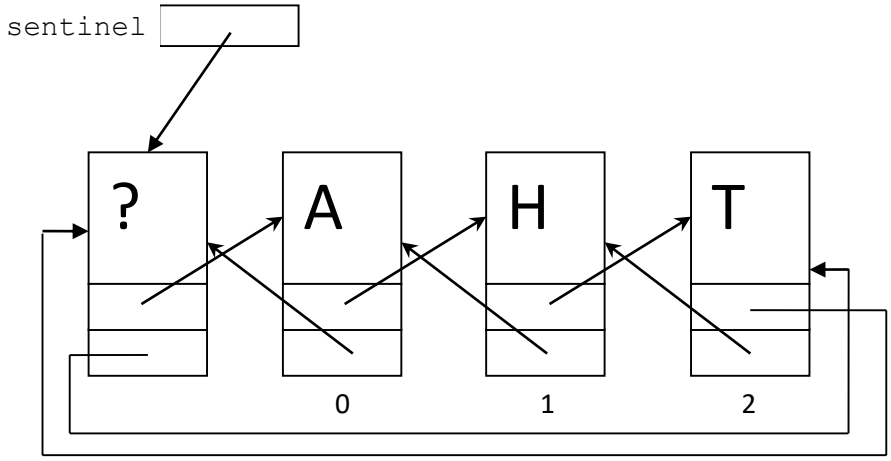
For implementation of the lists we will use below interface:

```
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();
}
```

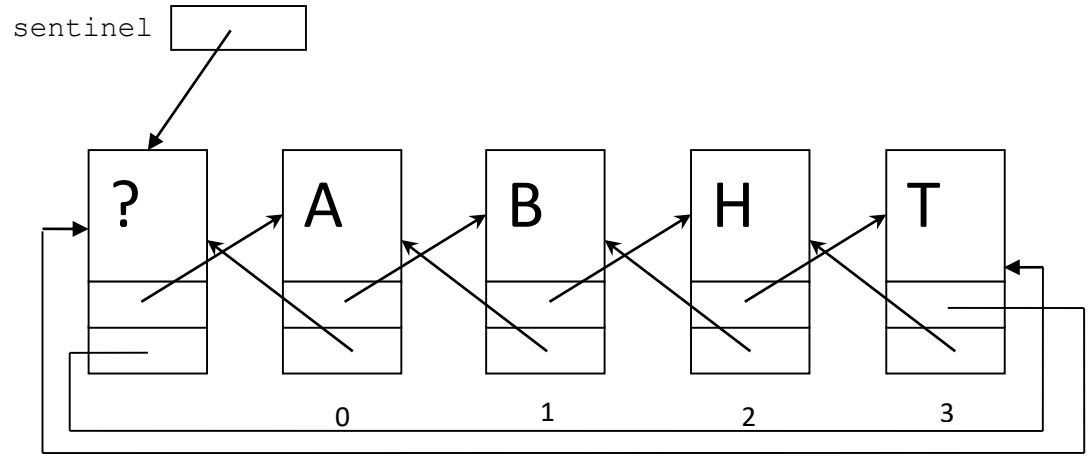
In this task we have to implement two way cycled ordered list with a sentinel. Sentinel is an extra element, which is not visible by the user of the list, but simplifies the implementation. The sentinel will be the first element in the list. Because the list is a two-way list, every element has to have two references `next` and `prev` which will point to next element in the list and to previous element in the list, respectively. So the begin of the class would be as follows:

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

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

        Element sentinel;
...
}
```

We index the element in the list starting from 0, but without taking into consideration the sentinel element. Graphically the list can be presented in the following way:



We cannot give to the user of the list possibility to break the order of elements. So the only operation to insert a new element is the method `add()` without index. It is means, that after insertion an element "B" you have to insert it after "A" and before "H":



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`, `TwoWayCycledOrderedListWithSentinel.java`, `Link.java`.

Task list

1. Modify the class `Link`, which will now store two information: reference string (field `ref`) and the weight of that link (field `weight`). Now after the name of link (without any space) there can be a **positive** integer number between parenthesis like "link=abc(12)". Still it can exists links without such a number between parenthesis like "Link=QeW" – in such a case we assume that the weight is equal one. If we want to compare two links, only fields `ref` will be compared. To do that you have to implement `Comparable` interface for a `Link` class and use `compareTo()` methods in many functions.
1. Modify the way to construct the string for a `Document` in `toString()` function. From now on under first line it have to be 10 links with weights in parenthesis separated by one space (no space after last link in a line). Of course in last line only the rest of links is presented. Implement also auxiliary methods `isCorrectId(String id)`

and `createLink(String link)` which meaning was explained in previous lists of tasks. They are used in the `Main` class.

2. Implement `IList<E>` interface with method `listIterator` for a class `TwoWayCycledOrderedListWithSentinel<T>`. This is ordered two-way cycled list with a sentinel. Because the list is ordered an iterator cannot modify the order of the list. From the same reason let assume, that methods **`void`** `add(`**`int`** `index, E element), E set(`**`int`** `index, E element),` will throw and exception `UnsupportedOperationException`. In the `add(E e)` we assume that the element will be added on proper position depending of the link name on the highest possible position.
    a. Implement also a method:
    `void add(TwoWayCycledOrderedListWithSentinel <T> anotherList)` which will add `another` list to **`this`** list. After the operation the another list have to be empty, all element will be in the first list, but of course ordered. Elements from another list with this same name have to be placed after elements from **`this`** list **. The complexity of this method have to be O(n)**. If the second list is the same list – do nothing.
    b. Implement also a method:
    `void removeAll(E element)` which will remove all elements equal to input element in the time O(n).
3. Your implementation of the classes `TwoWayCycledOrderedListWithSentinel<E>, Link, Document` will be tested with format specified in the appendix. If there is a code `throw new UnsupportedOperationException();` in the template - you don't need implement the method.

For 100 points present solutions for this list till Week 6.
For 80 points present solutions for this list till Week 7.
For 50 points present solutions for this list till Week 8.
After Week 8 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. The `name` have to be a correct identificator. If it is not true, write in one line "`incorrect ID`".
In the constructor the `name` have to stored in lower case letters.

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. The `str` can be in the same format like a link in this task: only identificator or identificator with a weight in parenthesis.

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:
`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 – like explained in tasks list.

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 – like explained in tasks list, of course 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 returned `Link` object in the format `<name>(<weight>)`. If the index is incorrect, write in one line "error".

If a line has a format:
`rem <idStr>`
your program has to call `remove(idStr)` 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:
`remall <idStr>`
your program has to call `removeall(new Link(idStr))` for current document.

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=ggg and link=cc link=a link=e link=GGG(5) link=eeee(15)
link=gGg(4)
adsad link=abc link=cos(30) link=cos(1) link=wrong(-1) link=wrong(asdf)
link=wrong(1.23) link=ok(123) link=kkk
```

```
eod
show
reverse
ch 1
ld first
correct link=cos(15) link=zzz link=cos(12) link=eee(1) link=eeee(14)
this is link=wrong(12 o yeah.
eod
show
reverse
addl 0
show
reverse
ch 0
show
reverse
ch 2
ld 3ddfg
ld ABC
and LiNk=Abc(4)
eod
show
ch 1
show
remall cos
show
ha
```

The output have to be:

```
START
!go 10
!ld zero
!show
Document: zero
a(1) abc(1) cc(1) cos(30) cos(1) e(1) eeee(15) ggg(1) ggg(5) ggg(4)
kkk(1) ok(123)
!reverse
Document: zero
ok(123) kkk(1) ggg(4) ggg(5) ggg(1) eeee(15) e(1) cos(1) cos(30) cc(1)
abc(1) a(1)
!ch 1
!ld first
!show
Document: first
cos(15) cos(12) eee(1) eeee(14) zzz(1)
!reverse
Document: first
zzz(1) eeee(14) eee(1) cos(12) cos(15)
!addl 0
!show
Document: first
a(1) abc(1) cc(1) cos(15) cos(12) cos(30) cos(1) e(1) eee(1) eeee(14)
eeee(15) ggg(1) ggg(5) ggg(4) kkk(1) ok(123) zzz(1)
!reverse
Document: first
zzz(1) ok(123) kkk(1) ggg(4) ggg(5) ggg(1) eeee(15) eeee(14) eee(1) e(1)
cos(1) cos(30) cos(12) cos(15) cc(1) abc(1) a(1)
!ch 0
!show
Document: zero
```

```
!reverse
Document: zero
!ch 2
!ld 3ddfg
incorrect ID
!ld ABC
!show
Document: abc
abc(4)
!ch 1
!show
Document: first
a(1) abc(1) cc(1) cos(15) cos(12) cos(30) cos(1) e(1) eee(1) eeee(14)
eeee(15) ggg(1) ggg(5) ggg(4) kkk(1) ok(123) zzz(1)
!remall cos
!show
Document: first
a(1) abc(1) cc(1) e(1) eee(1) eeee(14) eeee(15) ggg(1) ggg(5) ggg(4)
kkk(1) ok(123) zzz(1)
!ha
END OF EXECUTION
```