At this lab section, we will experiment different implementation of the ADT List in Java.

Lists

Asst. Prof. Dr. Feriştah DALKILIÇ
Res. Asst. Fatih DİCLE

PART 1 – Lists

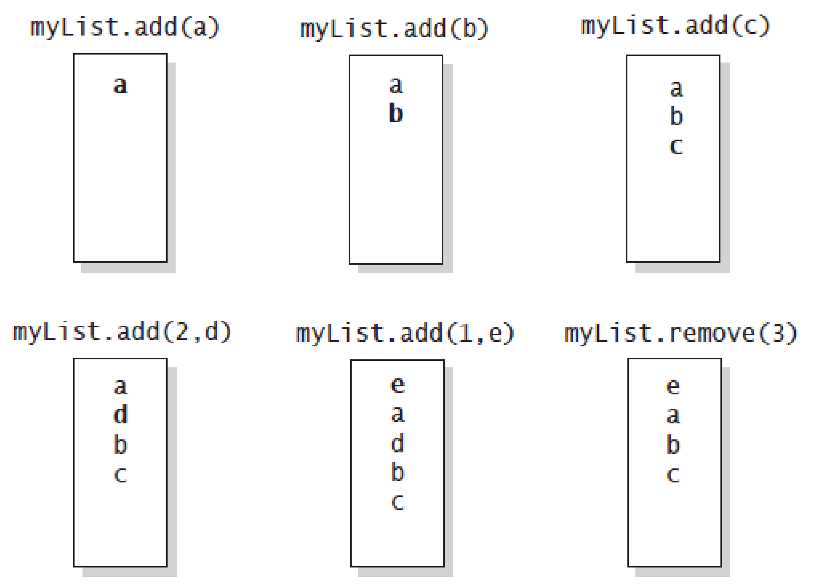
A list is a collection that provides a way to organize data. We can have to-do lists, gift lists, address lists, grocery lists, even lists of lists. Each list has a first item, a last item, and usually items in between. That is, the items in a list have a position: first, second, and so on. An item's position might be important to you, or it might not. When adding an item to your list, you might always add it at the end, or you might insert it between two other items already in the list.

The ADT list is more general and has entries that are objects of the same type.

Specifications for the ADT List

- Typically, you add a new entry at the end of the list.
- You can add a new entry anywhere: at the beginning, at the end, or in between items.
- You can remove an entry.
- You can remove all entries.
- You can replace an entry.
- You can look at any entry.
- You can look at all the entries.
- You can find out whether the list contains a particular entry.
- You can count the number of entries in the list.
- You can see whether the list is empty.

The effect of ADT list operations on an initially empty list



Exercise – 1

In this section, you will experiment with array-based ADT List implementation.

Step – 1

Create a new Java Project. Add the interface "ListInterface.java" given in *src* folder.

Step – 2

Add the class “AList.java” given in *src* folder. This class implements the ListInterface by using a resizable array. Fill in the blanks in `makeRoom` and `removeGap` methods.

The private method `makeRoom` that is called from `add` method, shifts list entries toward the end of the array, beginning with the last entry to open a gap for the new entry.

The private method `removeGap` that is called from `remove` method, shifts entries that are beyond the entry to be removed to the next lower position, beginning with the entry after the one to be removed and continuing until the end of the list.

Step – 3

Add a new class with the name of “Test.java” and paste the following code. This code prints the list content after each add, replace, and remove operation.

Test.java

```java
public class Test {

	public static void main(String[] args) {

		ListInterface<String> myList = new AList<>();

		myList.add("apple");
		printList(myList);
		myList.add("mango");
		printList(myList);
		myList.add(2, "banana");
		printList(myList);
		myList.add(3, "orange");
		printList(myList);
		myList.add(1, "kiwi");
		printList(myList);
		myList.replace(3, "kiwi");
		printList(myList);
		myList.remove(1);
		printList(myList);
		myList.replace(1, "banana");
		printList(myList);
	}

	//Generic Method
	public static <T> void printList(ListInterface<T> list) {
		for (int i = 1; i <= list.getLength(); i++) {
			System.out.print(list.getEntry(i) + " ");
		}
		System.out.println();
	}

}
```

Step – 4

Paste the output of the Test.java.

```
apple
apple mango
apple banana mango
apple banana orange mango
kiwi apple banana orange mango
kiwi apple kiwi orange mango
```

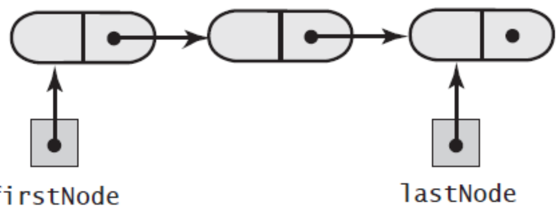
```
apple kiwi orange mango
banana kiwi orange mango
```

Exercise – 2

In this section, you will experiment with linked-based ADT List implementation.

We can implement ADT List by using a chain of linked nodes that has only a head reference. When we use such a data structure, the add method, which adds a new entry at the end of the chain, must traverse the chain from its beginning to end to locate at the end of the chain. We can improve the time efficiency of this add method by maintaining a reference to the end of the chain, as well as a reference to the beginning of the chain. In this way, we avoid a traversal of the entire chain each time the add method is called.

A linked chain with both a head reference and a tail reference



Step – 1

Add the “LList.java” given in *src* folder. This class implements the ListInterface by using a chain of linked nodes. Fill in the blanks in `add` and `replace` methods.

Step – 2

Modify Test.java and experiment same operation in Exercise 1. But this time you are expected to use the LList class.

Modified Test.java

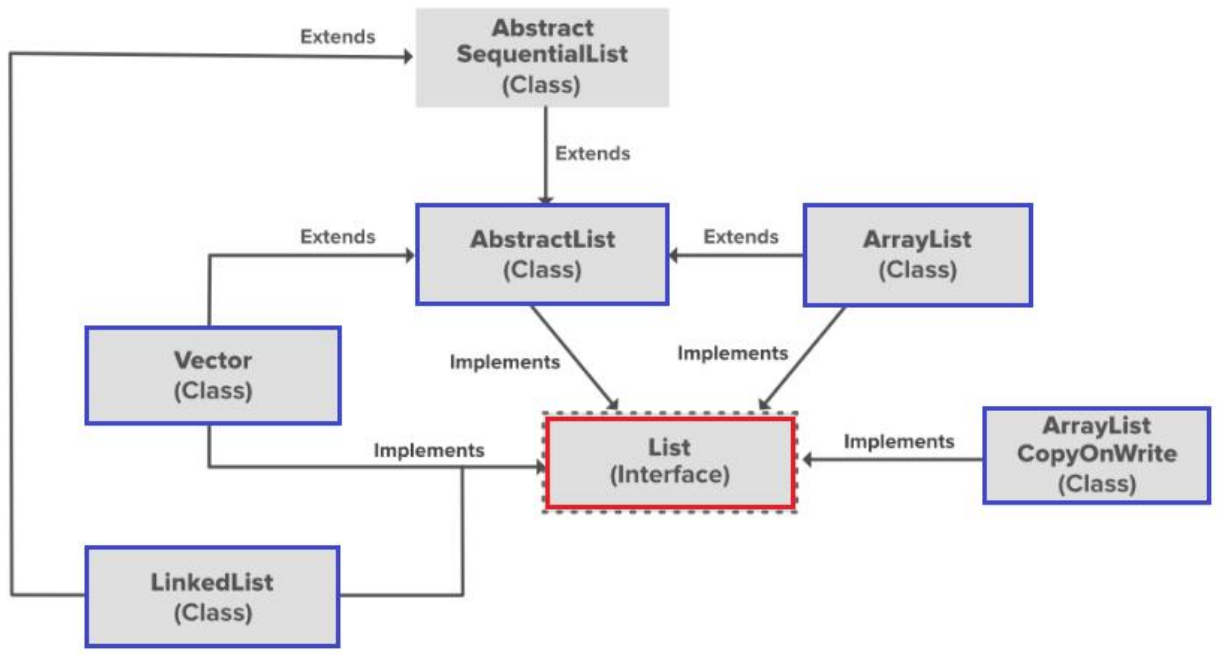
```
public class Test {
    public static void main(String[] args) {

        ListInterface<String> myList = new LList<>();

        myList.add("apple");
        printList(myList);
        myList.add("mango");
        printList(myList);
        myList.add(2, "banana");
        printList(myList);
        myList.add(3, "orange");
        printList(myList);
        myList.add(1, "kiwi");
        printList(myList);
        myList.replace(3, "kiwi");
        printList(myList);
        myList.remove(1);
        printList(myList);
        myList.replace(1, "banana");
        printList(myList);
    }

    //Generic Method
    public static <T> void printList(ListInterface<T> list) {
        for (int i = 1; i <= list.getLength(); i++) {
            System.out.print(list.getEntry(i) + " ");
        }
        System.out.println();
    }

}
```

Your Output

```
apple
apple mango
apple banana mango
apple banana orange mango
kiwi apple banana orange mango
kiwi apple kiwi orange mango
apple kiwi orange mango
banana kiwi orange mango
```

Exercise – 3

The Java Class Library contains the interface java.util.List. This interface is like our ListInterface, but it declares more methods. Also, some methods have different names or specifications, and the list entries begin at position 0 instead of 1.



Step – 1

The same package java.util contains the class ArrayList that implements the interface List. Modify Test.java and experiment same operation in Exercise 1 by using Java ArrayList.

Modified Test.java

```
import java.util.ArrayList;
public class Test {
    public static void main(String[] args) {

        ArrayList<String> myList = new ArrayList<>();

        myList.add("apple");
        printList(myList);
        myList.add("mango");
        printList(myList);
        myList.add(1, "banana");
        printList(myList);
        myList.add(2, "orange");
        printList(myList);
        myList.add(0, "kiwi");
        printList(myList);
        myList.remove(2);
        myList.add(2,"kiwi");
        printList(myList);
        myList.remove(0);
        printList(myList);
        myList.remove(0);
        myList.add(0,"banana");
        printList(myList);
    }

    //Generic Method
    public static <T> void printList(ArrayList list) {
        for (int i = 0; i < list.size(); i++) {
            System.out.print(list.get(i) + " ");
        }
        System.out.println();
    }

}
```

Your Output

```
apple
apple mango
apple banana mango
apple banana orange mango
kiwi apple banana orange mango
kiwi apple kiwi orange mango
apple kiwi orange mango
banana kiwi orange mango
```

Step – 2

The same package java.util contains the class LinkedList that implements the interface List. Modify Test.java and experiment same operation in Exercise 1 by using Java LinkedList.

Modified Test.java

```
import java.util.LinkedList;
public class Test {
    public static void main(String[] args) {

        //ArrayList<String> myList = new ArrayList<>();
        LinkedList<String> myList = new LinkedList<>();
        myList.add("apple");
        printList(myList);
        myList.add("mango");
        printList(myList);
        myList.add(1, "banana");
        printList(myList);
        myList.add(2, "orange");
        printList(myList);
        myList.add(0, "kiwi");
        printList(myList);
        myList.remove(2);
        myList.add(2,"kiwi");
        printList(myList);
        myList.remove(0);
        printList(myList);
        myList.remove(0);
        myList.add(0,"banana");
        printList(myList);
    }

    //Generic Method
    public static <T> void printList(LinkedList list) {
        for (int i = 0; i < list.size(); i++) {
            System.out.print(list.get(i) + " ");
        }
        System.out.println();
    }

}
```

Your Output

```
apple
apple mango
apple banana mango
apple banana orange mango
kiwi apple banana orange mango
kiwi apple kiwi orange mango
apple kiwi orange mango
banana kiwi orange mango
```

Step – 3

Is your output in Exercise 3 the same as the output in Exercise 1? Why?

Your Answer

During the implementation of java.util's list I decreased the index of the myList.add indexes by -1 so the built in list can work properly like our homemade lists. Maybe the complexities differ but the list methods stayed still.