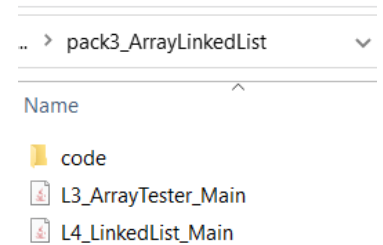


Objective(s) : understanding handling operations on an array.

### task 0:

Create a folder `pack3_ArrayLinkedList`. Inside the folder create package `code`



Given `MyArrayBasic` class in package `code` with the following methods

- void `add(int d)` – append `d` into an array
- void `insert(int d, int index)` – insert value `d` into the array at position `index`. Keep the order of the data unchanged.

```
package code;
public class MyArrayBasic {
    protected int MAX_SIZE = 5;
    protected int data[] = new int[MAX_SIZE];
    protected int size = 0;

    ...
}
```

- int `find(int d)` – return the index of value `d` in the array, else -1 (either ordered or unordered)
- int `binarySearch(int d)` – binary search in ordered array. return the index of value `d` in the array, else -1
- void `delete(int index)` – delete from ordered array i.e. the order of the data remains unchanged.
- `MyArray(int ... a)` – a constructor creating the first `MAX_SIZE`

Understand its mechanism through the following test code (`L3_ArrayTester.java`)

```
// import code.MyArray;
import code.MyArrayBasic;

class L3_ArrayTester_Main {
    public static void main(
        String[] args) {
        println("-demo1-----");
        arrayBasic_demo1();
        println("-demo2-----");
        arrayBasic_demo2();
        println("-demo3-----");
        arrayBasic_demo3();
    }
    ...
}
```

```
static private void arrayBasic_demo1() {
    MyArrayBasic demo =
        new MyArrayBasic(7,6,8,1,2,3);
    println(demo);
}
static private void arrayBasic_demo2() {
    MyArrayBasic demo = new MyArrayBasic();
    demo.insert(9, 0);
    demo.insert(7,0);
    demo.insert(5,0);
    println(demo);
    println("5 is at " + demo.find(5));
    println("5 is at " + demo.binarySearch(5));
    demo.delete(1);
    println(demo);
}
static private void arrayBasic_demo3() {
    MyArrayBasic demo = new MyArrayBasic(null);
    demo.add(3); demo.add(7);
    demo.add(5); demo.add(4);
    demo.add(6);
    //index out of bound due to overflow
    demo.add(1);
}
```

**task 1:** implement class MyArray which extends MyArrayBasic

with the following enhancements:

- MyArray() – a constructor with default MAX\_SIZE = 100\_000
- MyArray(int max) – a constructor with with supplied MAX\_SIZE;
- boolean isFull() – return true if there is not available cell to insert d (insertion would cause an exception)
- Boolean isEmpty() – return true if there is no data in the array (deletion would cause an exception)
- int [] expandByK(int k) – implicitly allocate a k \* MAX\_SIZE array to prevent overflow addition (add() method)
- int [] expand() – default k = 2 i.e. call expandByK(2); i.e. double the array's capacity

```
import code.MyArray;
import code.MyArrayBasic;

class L3_ArrayTesterPackX {
    public static void main(
        String[] args) {
        ...
        println("-demo4-----");
        myArray_demo4();
        println("-task2-----");
        task2();
    }
}
```

```
static private void myArray_demo4() {
    MyArray demo = new MyArray(5);
    demo.delete(0);
    demo.add(3);
    demo.add(7);
    demo.add(5);
    demo.add(4);
    demo.add(6);
    demo.add(1);
    println(demo);
}
```

**task 2:** use

System.currentTimeMillis()

to measure time

performance. Notice the time it takes for each data size.

```
static private void task2() {
    for (int N = 200_000;
        N <= 10 * 200_000; N += 200_000) {
        long start = System.currentTimeMillis();
        MyArray mArray = new MyArray(N);
        for (int n = 1; n < N; n++)
            mArray.add((int) (Math.random()*1000));

        long time = (System.currentTimeMillis() - start);
        println(N + "\t\t" + time);
    }
    println("with expansion");
    for (int N = 200_000;
        N <= 10 * 200_000; N += 200_000) {
        long start = System.currentTimeMillis();
        MyArray mArray = new MyArray();
        for (int n = 1; n < N; n++)
            mArray.add((int) (Math.random()*1000));

        long time = (System.currentTimeMillis() - start);
        println(N + "\t\t" + time);
    }
}
```

Run task2() 3 times. Write down the result execution time to the bellowed table.

If you adjust the size of the initial N (and step size), correct it to the table as well.

N	MyArray(N)			MyArray()		
	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>
200_000	14	15	13	8	10	11
400_000	14	14	16	20	23	26
600_000	56	27	49	24	27	29
800_000	26	25	22	24	45	29
1_000_000	29	32	30	35	48	40
1_200_000	36	36	37	42	66	49
1_400_000	38	44	39	41	48	44
1_600_000	48	49	47	45	49	60
1_800_000	51	53	52	54	61	68
2_000_000	58	57	60	56	62	59

In your opinion,

1. Given the different characteristic between the fixed sized and the expandable array

(MyArray(N) vs. MyArray()), which type of array's execution time should be faster?

**MyArray(N) with specific initial capacity is faster. If size is not unsure MyArray() can be a reasonable choice.**

2. In your opinion, how this experiment should be improved such that the execution time should reflect its true characteristic.

**You can create a more robust and reliable experiment to analyze the true characteristics of the array data structure and gain insights to its performance.**

**submission:** MyArray\_XXYYYYY.java and this pdf.

Due date: TBA