BCS405 – Data structures
Dr. Sophea PRUM
sopheaprum@gmail.com

****

If a programmer codes just for fun he has all his skill.
If he codes for score his hand tremble and his breath is uneasy

---

Part 1. Array

An *array* is a container that holds a fixed number of values of a single type. An *array* is an indexed data structure, which means *array* elements can be accessed by their index numbers using the *subscript operator* [ ].

PROPERTIES OF ARRAYS

- Arrays are objects
  - The method *clone* must be used to duplicate an array
- Array is created dynamically (at run time).
- Array holds a fixed number of values of a single type (primitive type, class)
- Each value is called element or component of the array
- If the component type is T, then the array itself has type T[]
- The length of an array is its number of components
- An array's length is set when the array is created, and it cannot be changed
- Each element of the array can be accessed by its index number using the subscript operator []
- Index values must be integers in the range 0 to array's length-1
- Variables of type short, byte, or char can be used as indexes.

However, you can't do the following with an array object:

- Increase or decrease its length, which is fixed.
- Add an element at a specified position without shifting the other elements to make room. Remove an
- element at a specified position without shifting the other elements to fill in the resulting gap.

Declaring Array Variables

A variable array must be declared to a specific type (primitive type or class) before being used in the program.

Syntax:

```
dataType[] arrayRefVar;
```

Example: declaring a variable name *myList*, an array of double

```
double[] myList;
```

By running this statement, JVM will create **an array reference variable myList**. However, the array hasn't been created yet.

Creating an array: an array can be created by using new operator

Syntax:

```
arrayRefVar = new dataType[arraySize];
```

Example: create an array containing 10 elements of double values

```
myList = new double[10];
```

This statement will:

- It creates an array of 10 value of type double.
- assigns the reference of the newly created array to the variable myList.

Declaring, creating an array and assigning the reference of the array to a variable ca be combined:

Syntax:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Example:

```
double[] myList = new double[10];
```

There is an alternative way to create an array and directly assign the value to each element

```
double[] myList = {5.6, 4.5, 3.3, 13.2, 4.0, 34.33, 34.0, 45.45, 99.993, 11123};
```

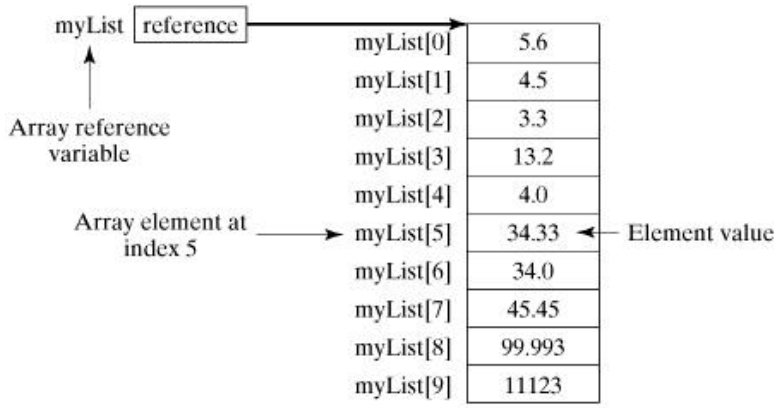


Figure 1: array myList

Figure 1. illustrate the array myList

- myList: array reference variable containing the reference to the object array
- Each element can be accessed by its index value using the subscript operator []. For example, access to value of element at index 5: myList[5]
- Caution: index value of array starts from 0 to array length-1

Processing Arrays

we often use either **for** loop or **foreach** loop to process the array elements.

Example 1: view all the elements in an array using **for** loob

```
public class PraticalLab4 {
    public static void main(String[] args) {
        // TODO code application logic here
        double[] myList = {5.6, 4.5, 3.3, 13.2, 4.0};
        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
           System.out.println(myList[i]);
        }
    }
}
```

Example 2: summing all elements using **foreach** loop

```
public class PraticalLab4 {
    public static void main(String[] args) {
        double[] myList = {5.6, 4.5, 3.3, 13.2, 4.0};
        double sum = 0;
        for (double element : myList) {
           sum = sum + element;
        }
        System.out.println(sum);
    }
}
```

Example 3: find the maximum element in myList array

```
public class PraticalLab4 {

    public static void main(String[] args) {
        double[] myList = {5.6, 4.5, 3.3, 13.2, 4.0};
        double max = myList[0];

        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) {
                max = myList[i];
            }
        }
        System.out.println(max);
    }
}
```

Example 4: passing an array to methods

```
public class PraticalLab4 {

    public static void viewArray(double[] inList){
        for (int i = 1; i < inList.length; i++) {
            System.out.println(inList[i]);
        }
    }
    public static void main(String[] args) {
        double[] myList = {5.6, 4.5, 3.3, 13.2, 4.0};
        viewArray(myList);
    }
}
```

Insert an element

Delete an elements

Part 2. Recursive function

A *recursive* method is the method that calls itself. This powerful technique produces repetition without using loops (e.g., while loops or for loops). Most of the time, recursive algorithm is used to replace loops to make an elegantly simple solution to solve a complex problem. In some cases, recursive algorithm can solve the problem that cannot not be done by using other ways.

Example2: write a method for summing the value from 1 to N

```
public class PraticalLab4 {

    public static int summing(int n){
        int sum = 0;
        for (int i = 1; i <= n; i++) {
            sum = sum + i;
        }
        return sum;
    }
    public static void main(String[] args) {

        int sum = summing(10);
        System.out.println(sum);

    }
}
```

```
public static int summing(int n){
    //stop condition
    if(n == 1){
        return 1;
    }
    return (n + summing(n-1));
}

public static void main(String[] args) {
    int sum = summing(10);
    System.out.println(sum);
}
```

How to think recursively?

To work correctly, every recursive function must have a *basis* and a *recursive part*. The basis is what stops the recursion. We call it stop condition. The recursive part is where the function calls itself. Each recursive case must make progress toward the base case.

In the summing method, the stop condition is when n=0. Otherwise, process the recursive part to call the summing method again with smaller value of n. Each recursive case, the value of n decrease toward 0.

Let's set n=4.

Mathematically we can write:

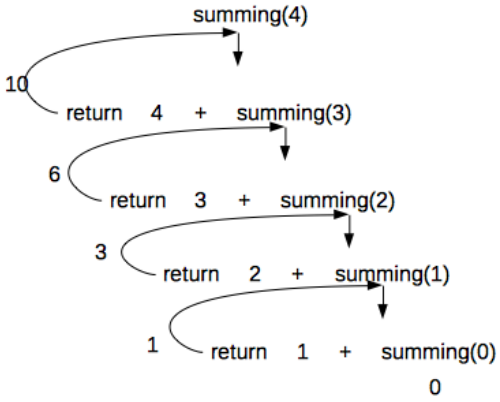
sum = 1 + 2 + 3 + 4

Alternatively we can write:

sum = (1 + 2 + 3) + 4

In the algorithm we can write:

sum = summing(3) + 4



Example2: write a method for multiplying the values in an array of integers

```
public static int multiplyLoop(int[] array){
    int m = 1;
    for(int i=0; i<array.length; i++){
        m = m * array[i];
    }
    return m;
}

public static int multiplyRecursive(int[] array, int index){
    if(index == array.length - 1){
        return array[index];
    }
    return array[index]*multiplyRecursive(array, index+1);
}
```

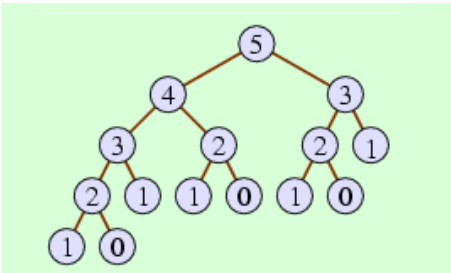
Example 3: Computer fibonancci number

The definition:

fibonancci(0) = 0
fibonancci(1) = 1
fibonancci(n) = fibonancci (n-1) + fibonancci(n-2)

```
public static int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

public static void main(String[] args) {
    int fib = fibonacci(10);
    System.out.println(fib);
}
```



References:

https://www.tutorialspoint.com/java/java_arrays.htm
http://www.toves.org/books/java/ch18-recurex/