# ET0736

## Lesson 2

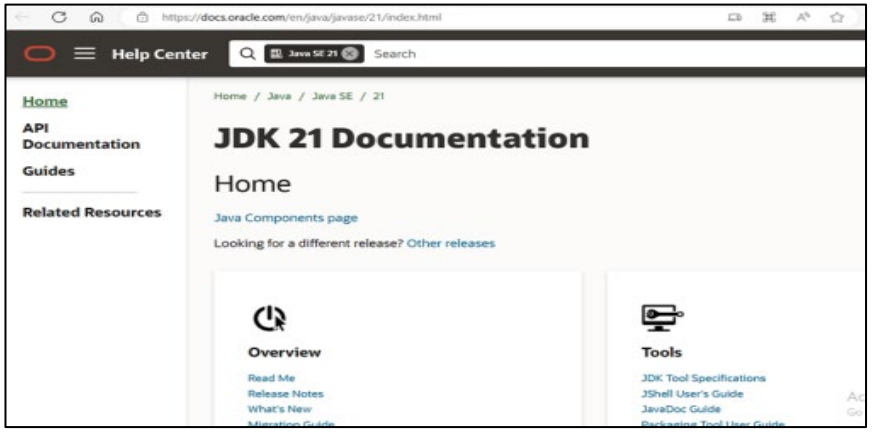
## Math API and Arrays

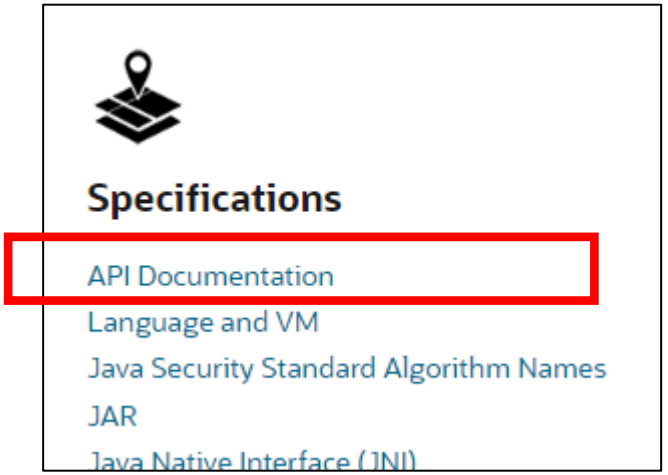
# Topics

- Math API
- Categories of data structures
- Linear data structure - Array in Java
- Bubble sort
- Linear search and Binary Search
- 2-D Array
- Arrays class in Java

# Overview of the Java API documentation

*https://docs.oracle.com/en/java/javase/21/index.html*




Scroll down to ***Specifications***
Select ***API Documentation***

# Overview of the Java API documentation


- Select ***Java SE***
- Frequently used classes are inside Module ***java.base***

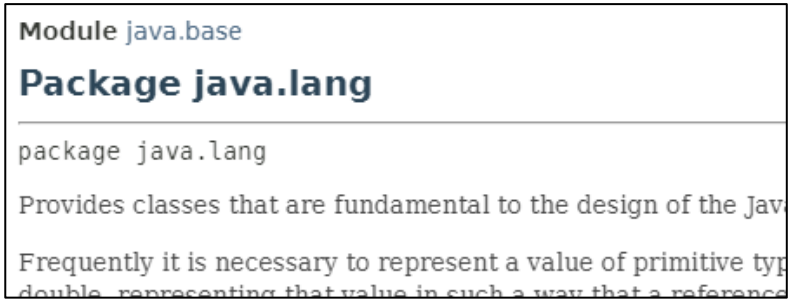
# Overview of the Java API documentation


Module ***java.base*** contains packages, such as

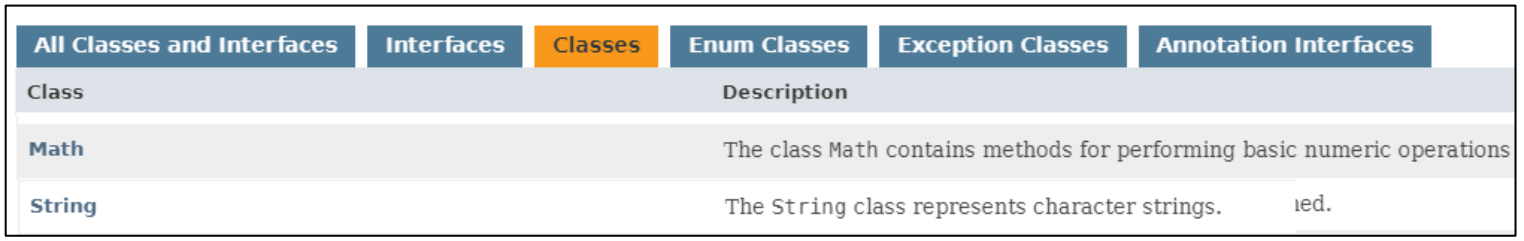
- ***java.io*** which contains classes dealing with I/O
- ***java.lang*** which contains classes that are fundamental to Java language

# Overview of the Java API documentation

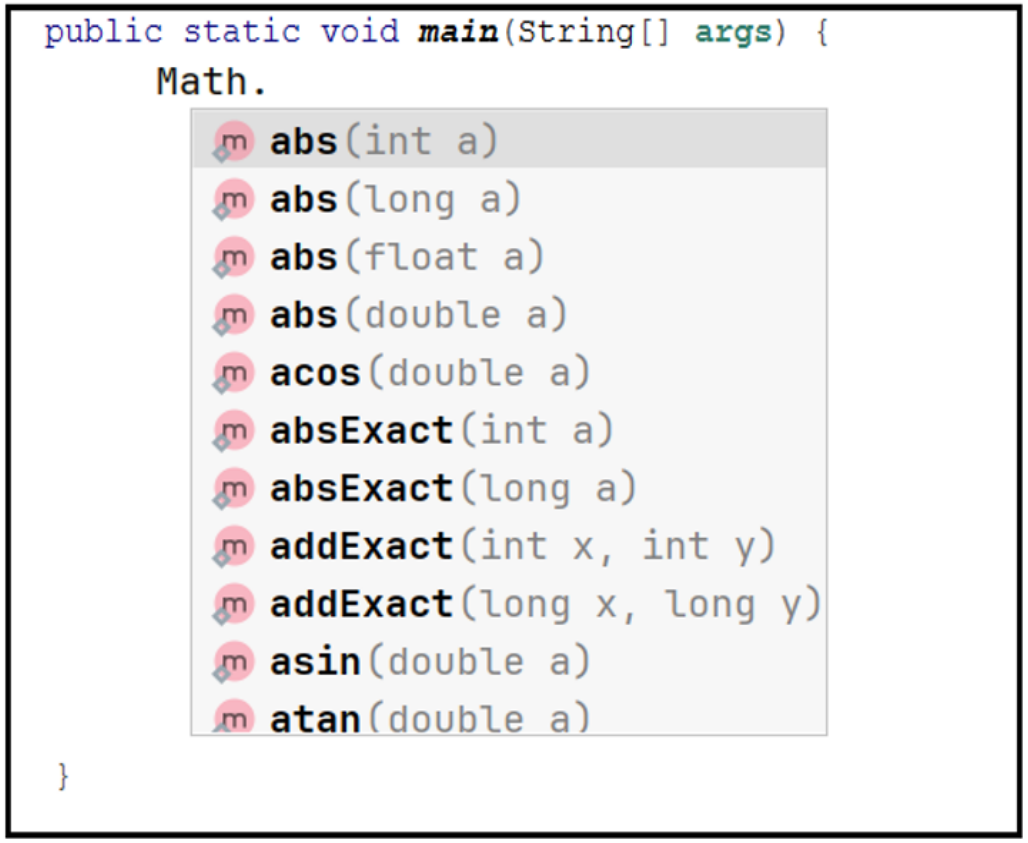
Inside package ***java.lang***.




Select ***Classes*** tab to view some of the classes that will be covered, such as ***Math***, ***String*** etc.



# Application Programming Interface (API) of Math class



# Application Programming Interface (API) of Math class

| | | |
|---|---|---|
| static double | **ceil**(double a) | Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer. |
| static int | **ceilDiv**(int x, int y) | Returns the smallest (closest to negative infinity) int value that is greater than or equal to the algebraic quotient. |
| static long | **ceilDiv**(long x, int y) | Returns the smallest (closest to negative infinity) long value that is greater than or equal to the algebraic quotient. |
| static long | **ceilDiv**(long x, long y) | Returns the smallest (closest to negative infinity) long value that is greater than or equal to the algebraic quotient. |
| static int | **ceilDivExact**(int x, int y) | Returns the smallest (closest to negative infinity) int value that is greater than or equal to the algebraic quotient. |
| static long | **ceilDivExact**(long x, long y) | Returns the smallest (closest to negative infinity) long value that is greater than or equal to the algebraic quotient. |
| static int | **ceilMod**(int x, int y) | Returns the ceiling modulus of the int arguments. |
| static int | **ceilMod**(long x, int y) | Returns the ceiling modulus of the long and int arguments. |
| static long | **ceilMod**(long x, long y) | Returns the ceiling modulus of the long arguments. |

| | | |
|---|---|---|
| static double | **max**(double a, double b) | Returns the greater of two double values. |
| static float | **max**(float a, float b) | Returns the greater of two float values. |
| static int | **max**(int a, int b) | Returns the greater of two int values. |
| static long | **max**(long a, long b) | Returns the greater of two long values. |
| static double | **min**(double a, double b) | Returns the smaller of two double values. |
| static float | **min**(float a, float b) | Returns the smaller of two float values. |
| static int | **min**(int a, int b) | Returns the smaller of two int values. |
| static long | **min**(long a, long b) | Returns the smaller of two long values. |

# Application Programming Interface (API) of Math class

Try out different methods under ***Math***:

**Math.round**

**Math.random**

# Math.random()

```
for (int i = 1; i<50; i++)
    System.out.println(Math.random());
```

Output:

```
:
0.21566982690582281
0.7532400683432676
0.05404106983273815
0.6682701086094087
0.6014272511431576
0.133937096003526
0.3031234408341005
0.3967109106121941
0.25361670171443096
0.9193446048000583
0.4280431527726417
:
```

# Math.random()

```
for (int i = 1; i<50; i++)
    System.out.println(Math.random()*100);
```

Output:

```
:
21.566982690582281
75.32400683432676
5.404106983273815
66.82701086094087
0.06014272511431576
13.3937096003526
91.93446048000583
28.0431527726417
:
```

}

# Math.random()

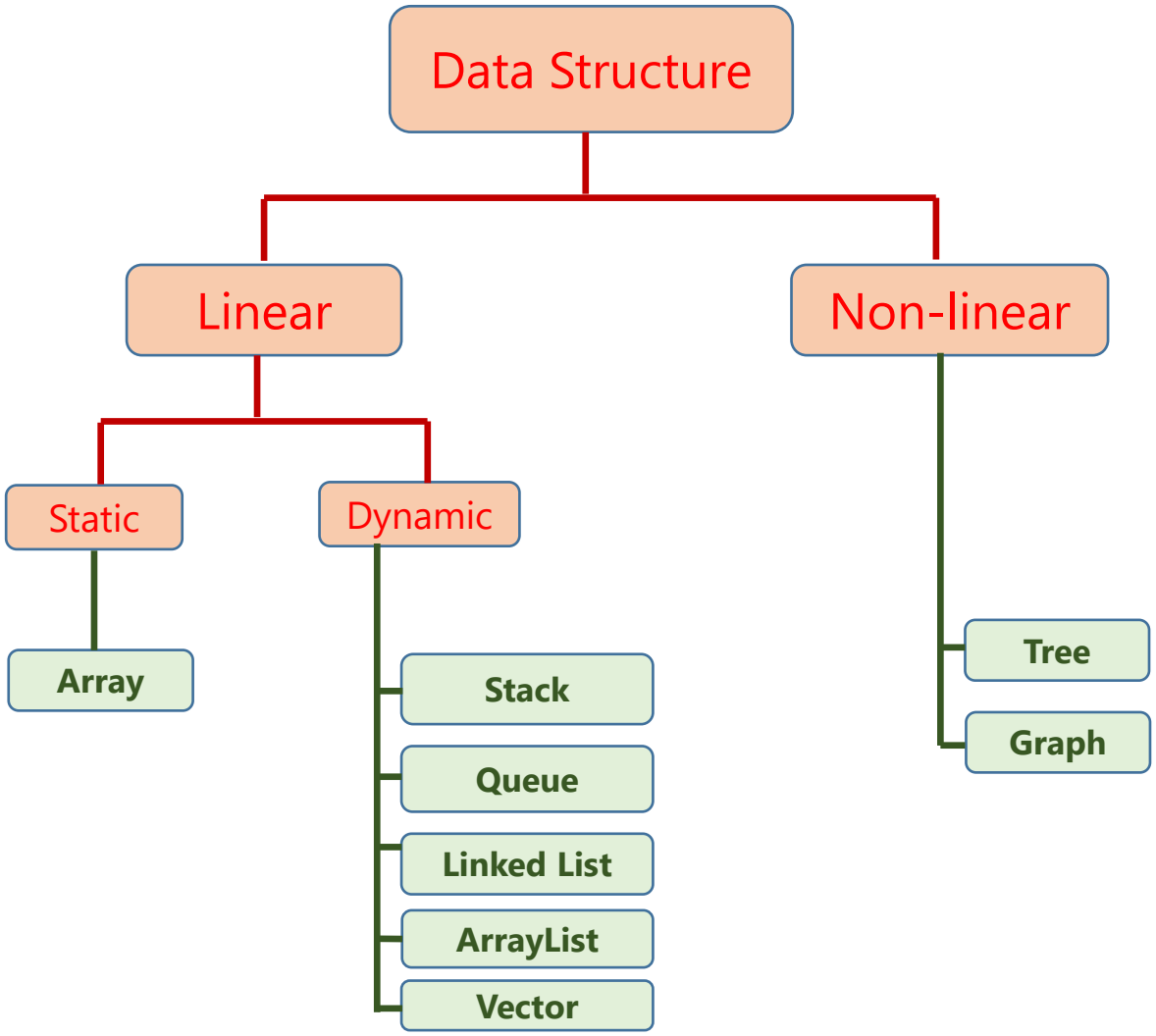
Generate random integers:

```
for (int i = 1; i<500; i++) {
    int x = (int) (Math.random()*100);
    System.out.println(x);
}
```

```
for (int i = 1; i<500; i++) {
    double x = Math.random()*100;
    System.out.println(Math.round(x));
}
```

# Categories of Data Structures

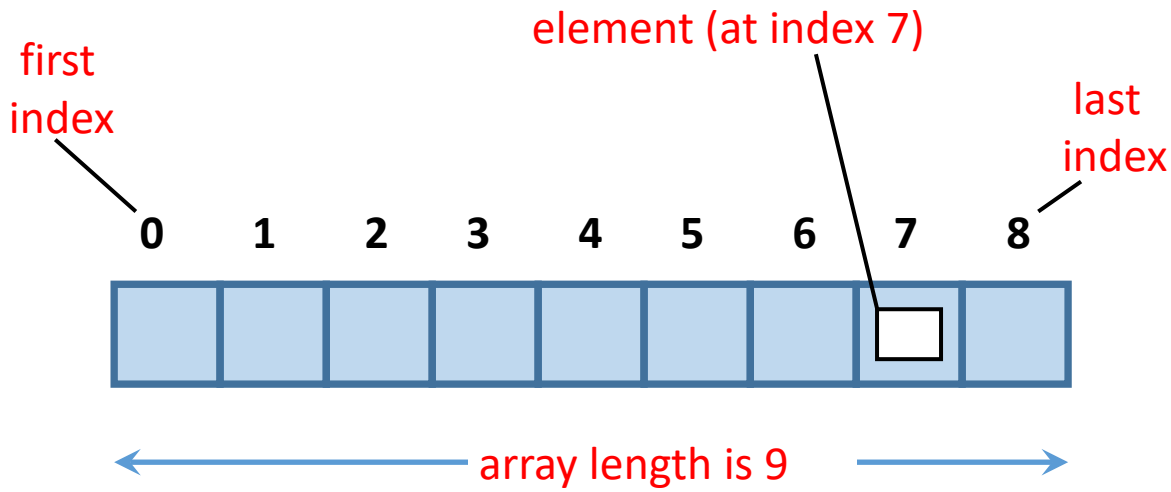
- Linear
- Non-linear



# Array

- Linear and static
- Stores a group of like-typed item under one variable name
- Each item is called *element*
- Stores data in consecutive memory location
- Storage memory is dynamically allocated but the size cannot be changed once initialised
- Array size must be specified by *int* or *short* value (not long)
- Size is fixed, cannot be changed, once it is defined
- Has a positioning index beginning with 0. (e.g. $5^{th}$ element would be accessed at index 4)
- Is a subclass of Object.
- Supports randomly access for the elements stored

# Array


element (at index 7)
first
index
last
index
0
1
2
3
4
5
6
7
8
array length is 9

# Array

Arrays can be declared using one of these syntax:

```
data-type[] variable-name[];     // preferred – [ ] identify array type
                                 // should appear with the data-type

data-type variable-name[];
```

Example:

```
int[] x ;
    or
int x[];

// x is just a reference
// without any memory allocation
```

# Array

Create actual array (with memory allocation):

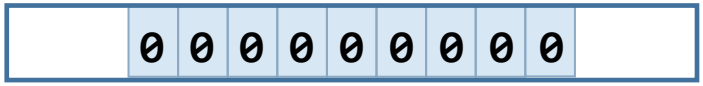
```
new data-type[length];
```

Example:

```
Int[] x;

x = new int[9];

// Memory space allocated
// Elements not yet initialised
// All int elements default 0
```




Memory allocated

| array element | default value |
|---|---|
| int | 0 |
| boolean | false |
| String | null |
| char | \u0000 |

# Array

Create actual array (with memory allocation and initialisation):

```
int[] x = {11,22,33,44,55,66,77,88,99} ;

System.out.println(x.length);   // array length is 9
```

However, this will cause error (must be in 1 statement):

```
int[] x ;

x = {11,22,33,44,55,66,77,88,99} ;  // error!
```

# Accessing Array Elements

Accessing the elements in an array (using loop)

```
int[] c = {11,22,33};
for (int x : c)
    System.out.println(x);
```

or

```
int[] c = {11,22,33};
for (int x=0; x<c.length; x++)
    System.out.println(x);
```

Output:

```
11
22
33
```

# Sorting 3 Strings / store in Array

```
String x = "I Love SG";
String y = "I Love SP";
String z = "Go Love SP";
String result[] = new String[3];

if (x.compareTo(y)<=0) {
    if (x.compareTo(z)<=0) {
        result[0] = x;
        if (y.compareTo(z) <=0){
            result[1]=y;
            result[2]=z;
        }
        else {
            result[1]=z;
            result[2]=y;
        }
    }
    else {
       result[0]=z;
       result[1]=x;
       result[2]=y;
    }
}
```

```
else {
    if (x.compareTo(z)<=0) {
        result[0] = y;
        result[1] = x;
        result[2] = z;
    }
    else {
        result[0] = y;
        result[1] = z;
        result[2] = x;
    }

}

for (String i : result){
    System.out.println (i);
}
```

# Processing Array Elements

Change the elements in an array (using loop)

```
int[] c = {11,22,33};

for (int x=0; x<c.length; x++)
   c[x] *= 10;   // multiple element by 10

for (int x : c)
   System.out.println(x);
```

Output:

```
110
220
330
```

# Passing/Returning Array (method)

```
public static void main(String[] args) {
   double x[] = {1.1,2.2,3.3,4.4,5.5,6.6};
   for (int t: roundUp(x))
       System.out.print (t +" ");
}

static int [] roundUp( double z[]){
    int r[];
    r = new int[z.length];
    for (int i=0; i<z.length; i++)
        r[i] = (int)(Math.round(z[i]));
    return(r);
}
```

Output:

**1 2 3 4 6 7**

# System.arraycopy()

java.lang.Object
java.lang.System

Class **System** offers an **arraycopy()** method to copy from a source array to another destination array:

| Modifier and Type | Method and Description |
|---|---|
| static void | arraycopy(Object src, int srcPos, Object dest, int destPos, int length) |

src = source array

dest = destination array

srcPos = index in src to start copying from

destPos = index in dest to start copying to

length = number of elements to be copied

What are the other ways to copy an array?

# System.arraycopy()

| Modifier and Type | Method and Description |
|---|---|
| static void | arraycopy(Object src, int srcPos, Object dest, int destPos, int length) |

Example:

```
int[] c = {11,22,33};
int[] d = {44,55,66,77};

System.arraycopy(c,0,d,0,3);   // array d becomes {11,22,33,77}


System.out.println(d[2]);
```

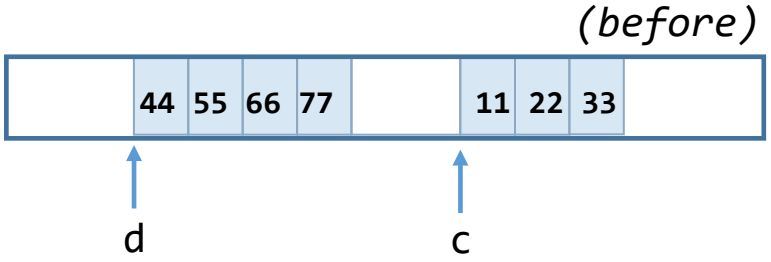
Output:

**33**

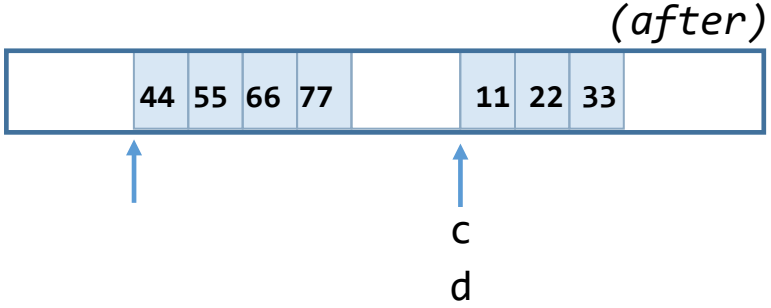
# Referencing Array

The code below merely changes the reference of variable **d**.
There is *no copying* of content from one array to another.

```
int[] c = {11,22,33};
int[] d = {44,55,66,77};
```

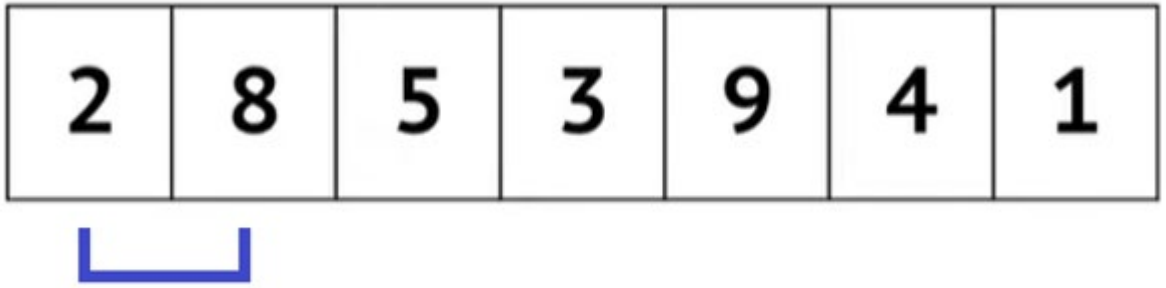



```
d=c;        // d now referencing {11,22,33}
```




```
System.out.println(d[2]); // 33
System.out.println(d[3]); // run-time error
```

# Bubble sort - concept




1. Work from left to right
2. Examine each item and compare it with the next item on the right
3. Swap them if item > right item
4. At the end of Round 1, the biggest number will be on the right-most position as a sorted number.
5. Repeat the above from left to the 2nd last item

# Bubble sort - concept

Round 1

| 2 | 8 | 5 | 3 | 9 | 4 | 1 |
|---|---|---|---|---|---|---|
| 2 | 5 | 8 | 3 | 9 | 4 | 1 |
| 2 | 5 | 3 | 8 | 9 | 4 | 1 |
| 2 | 5 | 3 | 8 | 9 | 4 | 1 |
| 2 | 5 | 3 | 8 | 4 | 9 | 1 |
| 2 | 5 | 3 | 8 | 4 | 1 | 9 |

Round 2

| 2 | 5 | 3 | 8 | 4 | 1 | 9 |
|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 4 | 1 | 9 |
| 2 | 3 | 5 | 8 | 4 | 1 | 9 |
| 2 | 3 | 5 | 4 | 8 | 1 | 9 |
| 2 | 3 | 5 | 4 | 1 | 8 | 9 |

Round 3

| 2 | 3 | 5 | 4 | 1 | 8 | 9 |
|---|---|---|---|---|---|---|
| 2 | 3 | 5 | 4 | 1 | 8 | 9 |
| 2 | 3 | 4 | 5 | 1 | 8 | 9 |
| 2 | 3 | 4 | 1 | 5 | 8 | 9 |

End of Round 4

| 2 | 3 | 1 | 4 | 5 | 8 | 9 |
|---|---|---|---|---|---|---|

End of Round 5

| 2 | 1 | 3 | 4 | 5 | 8 | 9 |
|---|---|---|---|---|---|---|

End of Round 6

| 1 | 2 | 3 | 4 | 5 | 8 | 9 |
|---|---|---|---|---|---|---|

sorted partition

* Array of size of 7 needs 6 rounds

# Bubble Sort – pseudocode

N is size

```
for i from 0 to N-1
  for j from 0 to N-1
    if a[j] > a[j+1]
       swap (a[j],a[j+1])
```

# Searching

- Linear or Sequential search
- Binary Search

# Linear (or Sequential) Search

- search key is compared with each element of the collection, one by one.
- If there is a match, it returns the index of the array between [0, size-1];
- otherwise, it returns -1 or the size of the array (some implementations deal with only positive indexes).
- Linear search has worst-case complexity of O(n).
- Is used when data is not sorted.

# Linear Search – array example

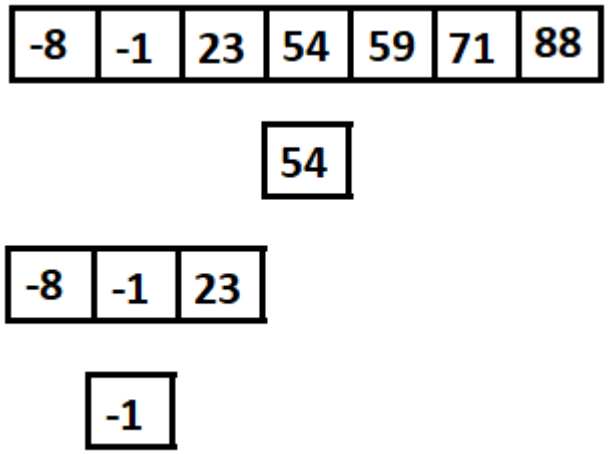
```
class TestLinearSearch {
  public static void main(String[] args) {
     int[] nums = {2, 12, 15, 11, 88, 19, 45};
     int key = 88;
     System.out.println(search(nums, key));
  }


  public static int search(int[] nums, int key) {
     for (int i=0; i<nums.length; i++){
        if (nums[i]==key) return(i);
     }
     return(-1);
  }
}
```

# Binary Search

- is applicable only to a sorted collections
- it compares the search key with the middle element
- If there is a match, it returns the element's index
- If the search key is less then the middle element, repeat searching on the left half;
- otherwise, search the right half.
- If the remaining element to be searched is zero, return -1 (in general)
- has worst-case complexity of $O(\log_2 n)$

# Binary Search – array example



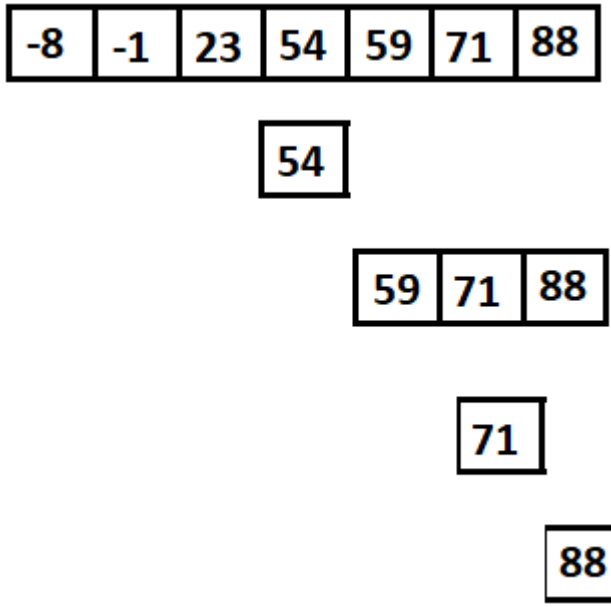

Key = -1

Middle: 54, Key = -1 ➔ no match

Key < Middle, search in lower half.

Middle: -1, Key = -1 ➔ Bingo!

# Binary Search – array example




Key = 88

Middle: 54, Key = 88 ➔ no match

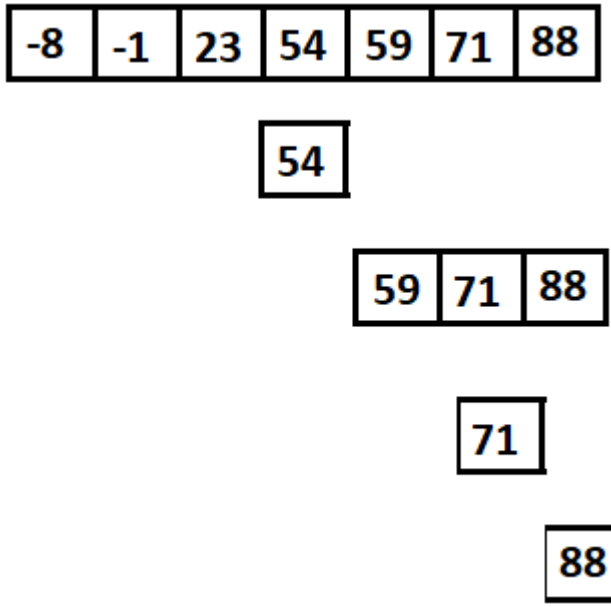
Key > Middle, search in upper half.

Middle: 72, Key = 88 ➔ no match

Key > Middle, search in upper half.

Upper half left with 1 element.
Key found!

# Binary Search – array example




Key = 100

Middle: 54, Key = 100 ➔ no match

Key > Middle, search in upper half.

Middle: 72, Key = 100 ➔ no match

Key > Middle, search in upper half.

Upper half left with 1 element.
It is still not the key.
Upper limit hit.
Key not found. Return -1.

# 2-D Array

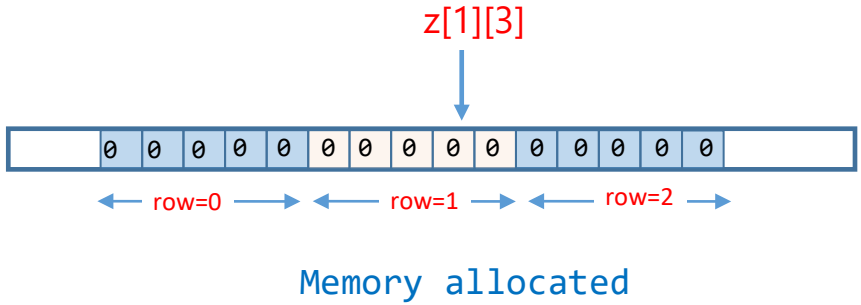
A two-dimensional is stored in the computer's memory one row following another.

Declaration syntax for two dimensional array:

```
data-type variable-name[][] = new data-type[rows][columns];
```

Example:

```
int z[][] = new int[3][5];
```



# 2-D Array

Create a two-dimension array (with memory allocation and initialisation) of 4 rows x 3 columns:

```
int[][] a = {{11,22,33},{44,55,66},{77,88,99}, {100,200,300}} ;
```

Obtain the rows and columns:

```
System.out.println( "rows = "  + a.length);
System.out.println( "columns = "  + a[0].length);
```

Output:

```
rows = 4
columns = 3
```

# Processing 2-D Array

```
int[][] a = {{11,22,33},{44,55,66},{77,88,99}, {100,200,300}} ;

for (int row =0 ; row<a.length; row++) {
    for (int col = 0; col < a[row].length; col++)
        System.out.print( a[row][col] + "\t" );
    System.out.println();
}
```

Output:

```
11      22      33
44      55      66
77      88      99
100     200     300
```

# Passing 2-D Array to method

```
public static void main(String[] args) {
    int[][] a = {{11,22,33},{44,55,66},{77,88,99}, {100,200,300}} ;
    System.out.print( "Total  = " + sumAll(a) );
}

static int sumAll(int a[][]){
    int sum=0;
    for (int row =0 ; row<a.length; row++) {
        for (int col = 0; col < a[row].length; col++)
            sum += a[row][col];
    }
    return(sum);
}
```

Output:

**Total = 495**

# Ragged 2-D Array

In a two-dimensional array, only the first dimension needs to be allocated. The second dimension for each row, can be allocated with different numbers, if needed, separately.

Example:

```
int z[][]=new int[3][]; // three rows

z[0] = new int[2]; // first row - 2 elements
z[1] = new int[5]; // second row - 5 elements
z[2] = new int[9]; // third row - 9 elements

for (int i=0; i<z.length; i++) {
   for (int j=0; j<z[i].length; j++)
      System.out.print(j);
   System.out.println();
}
```

What is the output from the code above?

# Arrays class

- From java.util package
- Has methods that can be used to fill, sort, search, etc in arrays.
- Example:

```
public static void main(String[] args) {
    int [] a= { 10, 20, 15, 22, 35};
    whereAmI(a, 20);
    Arrays.sort(a);
    whereAmI(a, 20);
}

public static void whereAmI(int []x, int key){
    for (int i=0; i<x.length; i++)
        if (x[i]==key)
           System.out.println(key + " found at index = " + i);
}
```

**Output:**

```
20 found at index = 1
20 found at index = 2
```

# Arrays class

Explore other useful methods of the Arrays class by researching the list provided after typing Arrays and followed by a dot: