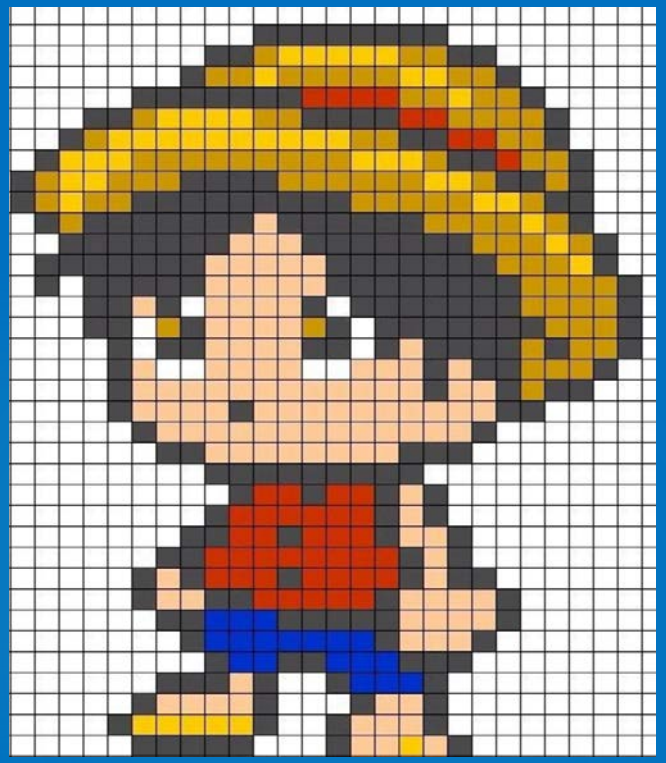
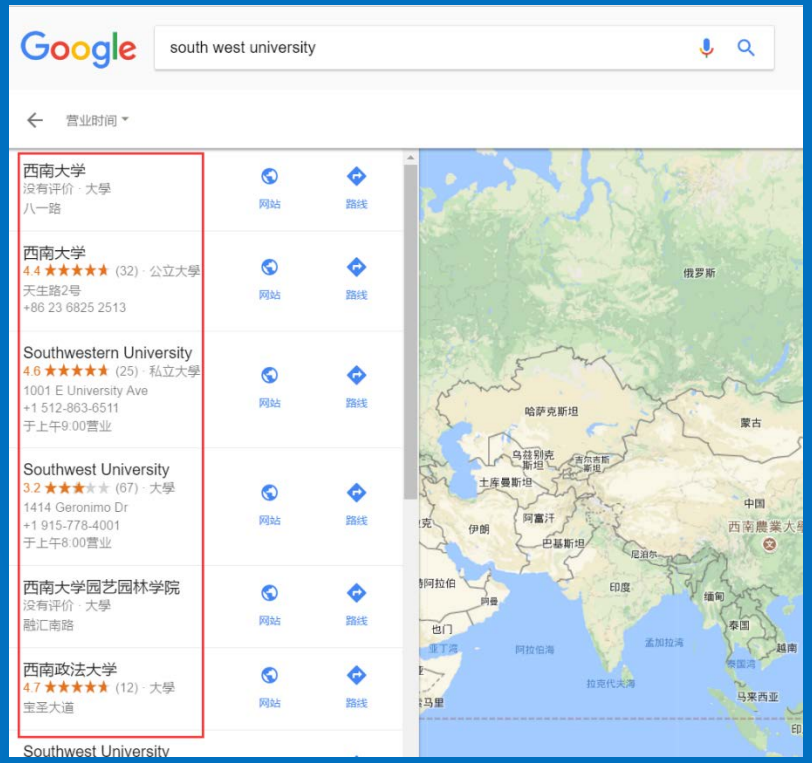
Chapter 5 Arrays

Why do we need arrays?

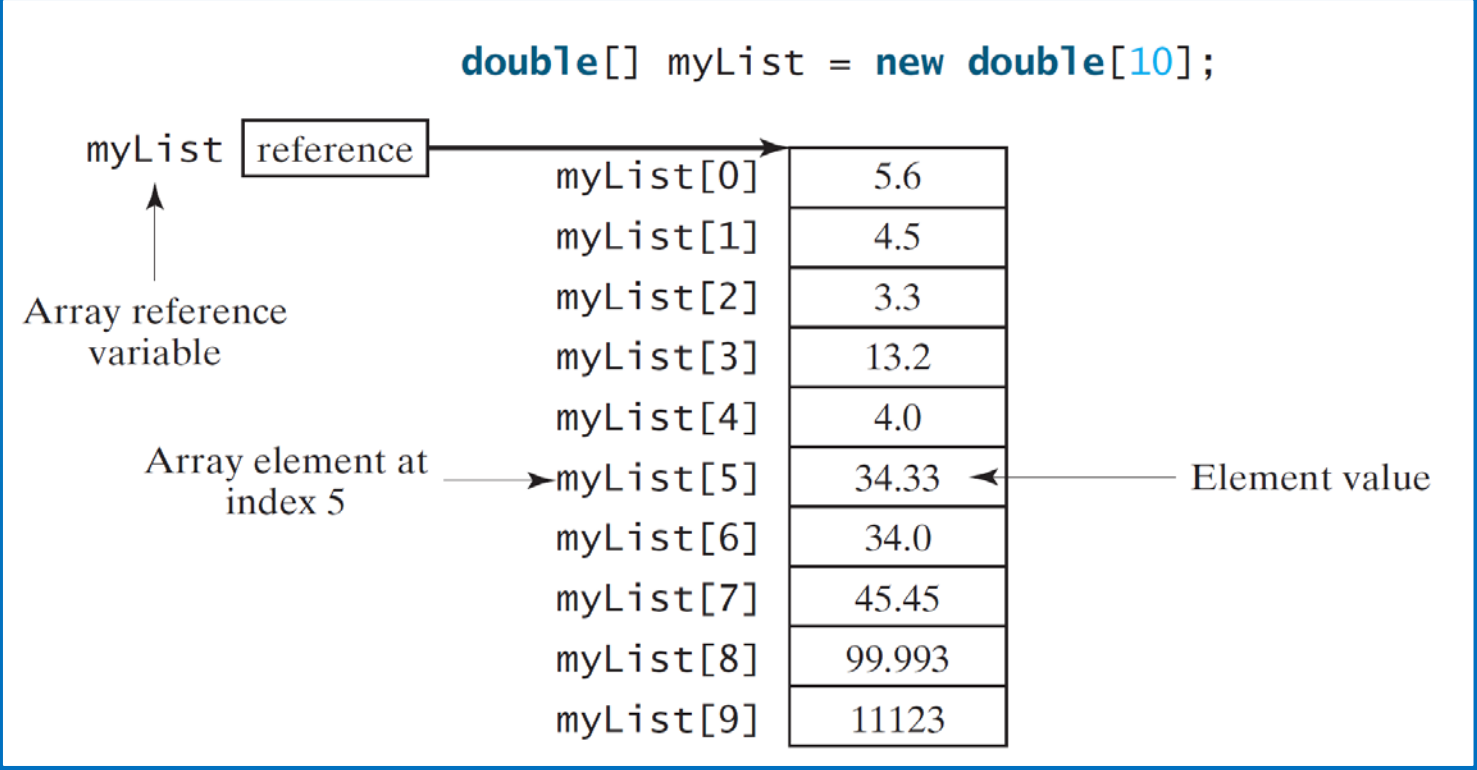




1.572163064	66.83692783	80.27714911	1.210725853
57.69031639	16.32088305	92.36730201	2.329799378
41.92264332	13.68081716	50.63282271	63.56540185
79.19182139	11.43948781	63.79962001	4.034407807
16.11631979	40.25748777	78.16780907	20.74061703
35.04598605	36.86097306	63.64822405	78.8683574
27.89138662	10.41356427	48.70148547	59.07889267
81.78941794	59.27703901	7.373904541	49.99449009
42.39733711	87.65180846	81.74575615	35.46606042
0.647455729	90.90445371	1.626061126	39.50546049
65.96523879	29.51652436	70.69875291	31.274867
28.34921145	69.11417575	82.61957755	51.29020047
19.13275283	33.55556616	49.43853921	31.83921395
30.07716603	87.59738004	80.88860627	68.61532239
97.71741627	84.24522292	69.23614168	34.85079658
65.05900463	27.39300058	72.82367253	9.581966499

Introducing Arrays

Array is a data structure that represents a collection of *the same types* of data. Once an array is created, its *size* is *fixed*. An array *reference variable* is used to access the elements in an array *using an index*.



Declaring Array Variables:

```
a. datatype[] arrayRefV;
double[] myList;
b. datatype arrayRefV[];
double myList[];
```

Creating Arrays:

```
arrayRefV = new type[size];
myList = new double[10];
```

Array Size and Default Values

When space for an array is allocated, the array size must be given, specifying the number of elements that can be stored in it. The *size* of an array *cannot be changed* after the array is created. Size can be obtained using ***arrayRefV.length***. For example:

```
double[] myList = new double[10];
System.out.println(myList.length); // 10
```

When an array is created, its elements are assigned the ***default value*** of ***0*** for the ***numeric*** types, ***\u0000*** for ***char*** types, ***false*** for ***boolean*** types, and null for *reference* types.

```
double[] list1 = new double[2]; // {0.0, 0.0}
int[] list2 = new int[2]; // {0, 0}
char[] list3 = new char[2]; // {'\u0000', '\u0000'}
boolean[] list4 = new boolean[2]; // {false, false}
Object[] list5 = new Object[2]; // {null, null}
```

Initializing and Accessing Arrays

The array elements are accessed through the index. Array indices are *0* based; that is, they range from *0* to *arrayRefVar.length-1*. We can initialize and access arrays using loop statements.

```
public class Client {
    public static void main(String args[]) {
        Scanner input = new Scanner(System.in);
        double[] list = new double[10];
        for(int i = 0; i < list.length; i++)
            list[i] = input.nextDouble();
        for(int i = 0; i < list.length; i++)
            System.out.printf("%-5.1f", list[i]);
    }
}
```

Array Initializers

Declaring, creating, initializing in one step:

```
public class Client {
    public static void main(String args[]) {
        // Array Initializers
        int[] list1 = {1,2,3,4,5,6};
        // Wrong
        int[] list2;
        list2 = {1,2,3};
        // Anonymous array
        int[] list3;
        list3 = new int[]{1,2,3,4,5};
    }
}
```

Access arrays using *for-each* loop


```
// for (elementType value: arrayRefVar) {}
public class Client {
    public static void main(String args[]) {
        int[] list1 = {1,2,3,4,5,6};
        for(int e: list1){
            System.out.println(e);
        }

        for(int e,i = 0; i < list1.length; i++){
            e = list1[i];
            System.out.println(e);
        }
    }
}
```

Case 1: Analyze Numbers

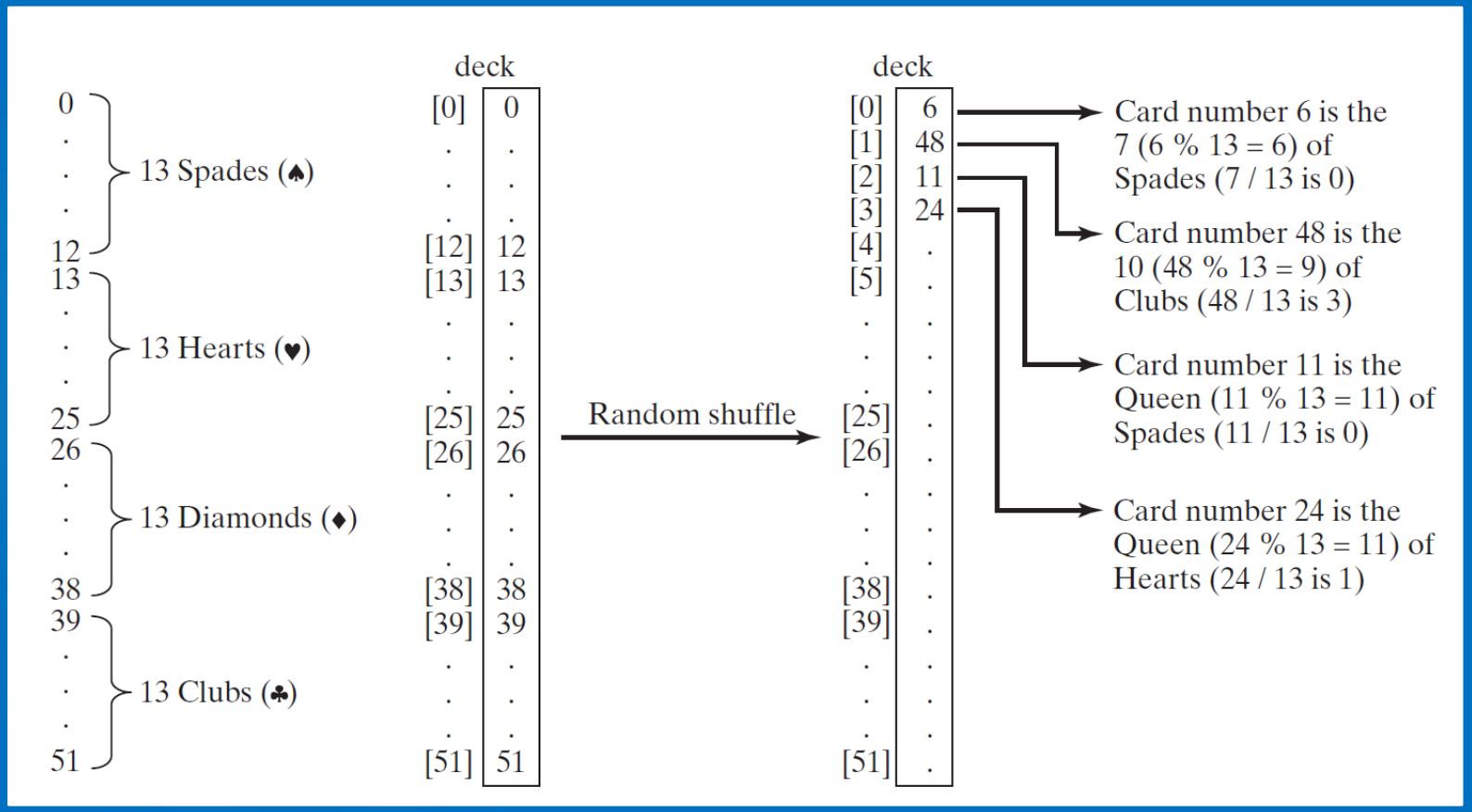
The problem is to write a program that finds the number of items *above the average* of all items.

```
I:\2020\Chapter05 Arrays>java AnalyzeNumbers
Enter the number of items: 10
Enter the numbers: 1 2 3 4 5 6 7 8 9 10
Average is 5.5
Number of elements above the average is 5
```

AnalyzeNumbers

Case 2: Deck of Cards

The problem is to create a program that will *randomly* select four cards from a deck of cards.

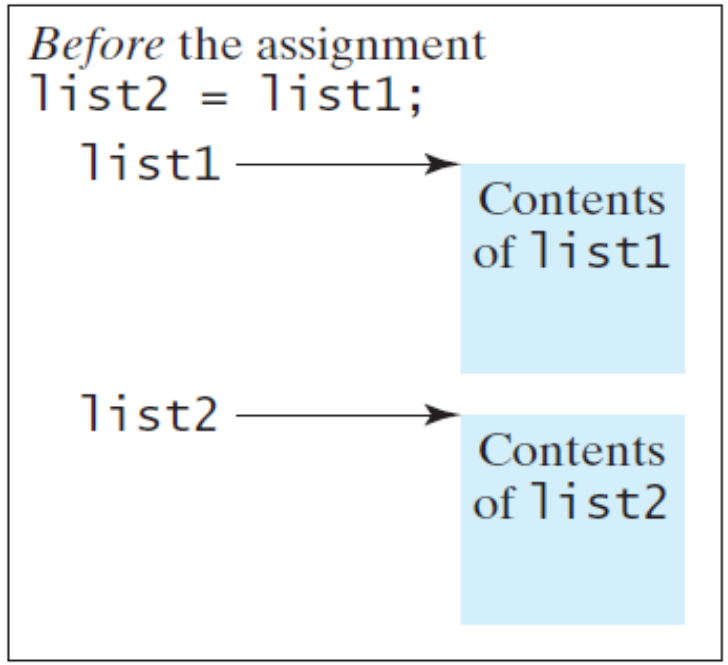


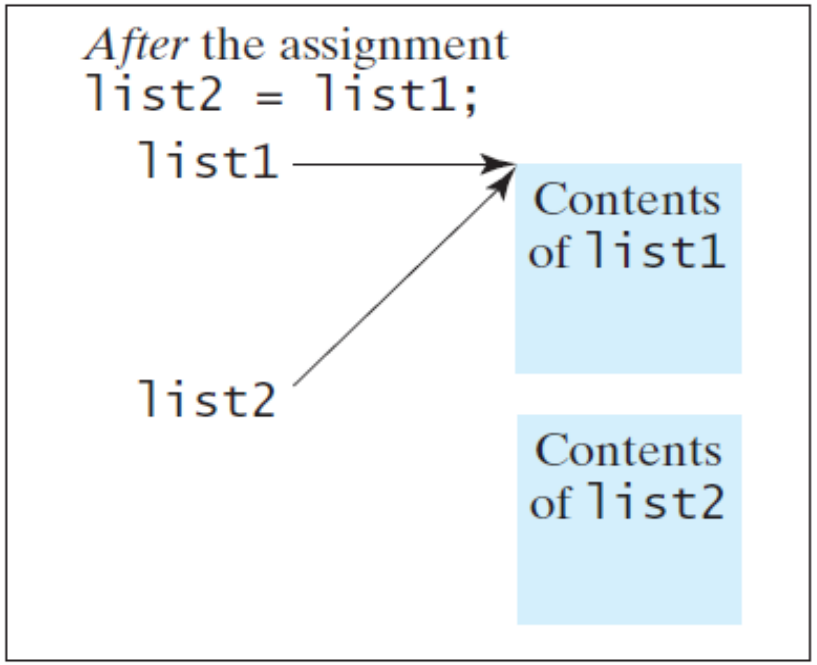

```
Card number 20: 8 of Hearts
Card number 42: 4 of Clubs
Card number 14: 2 of Hearts
Card number 16: 4 of Hearts
```

DeckOfCards

Copying Arrays

To copy the *contents* of one array into another, you have to copy the array's individual elements into the other array. Often, in a program, you need to *duplicate* an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as : `list2 = list1`





Assigning one array variable to another array variable *actually copies* one *reference* to another and makes *both* variables *point* to the *same* memory location.

Copying Arrays Using a *loop*

Use a loop to copy individual elements *one by one*

```
public class Test {
    public static void main(String[] args) {
        int[] sourceArray = {2, 3, 1, 5, 10};
        int[] targetArray = new int[sourceArray.length];
        for (int i = 0; i < sourceArray.length; i++)
            targetArray[i] = sourceArray[i];
    }
}
```

Copying Arrays Using *arraycopy()*

Use the static *arraycopy* method in the *System* class

```
public class Test {
    public static void main(String[] args) {
        int[] srcArray = {2, 3, 1, 5, 10};
        int[] tarArray = new int[srcArray.length];
        // arraycopy(srcArray, srcPos, tarArray, tarPos, length);
        System.arraycopy(srcArray, 0, tarArray, 0, srcArray.length);
        for(int e:tarArray) {
            System.out.printf("%-4d", e);
        }
    }
}
```

Passing Arrays to Methods

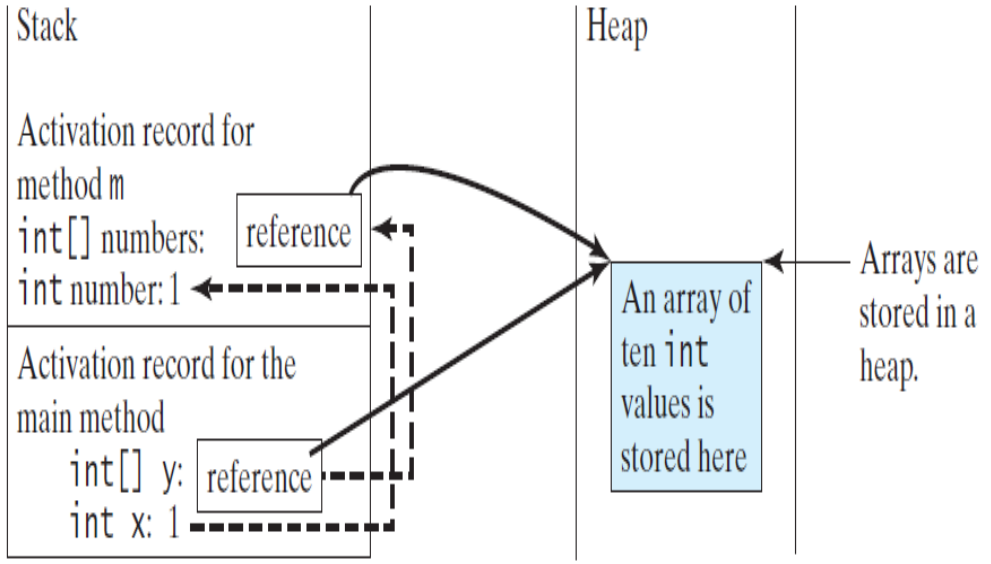
When passing an array to a method, the *reference* of the array is passed to the method.

```
public class Test {
    public static void main(String[] args) {
        int[] list = {3, 1, 2, 6, 4, 2};
        printArray(list);
        // anonymous array: new elementType[]{value1, value2, ..., valueN};
        printArray(new int[]{3, 1, 2, 6, 4, 2});
    }
    public static void printArray(int[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Passing Arrays to Methods: *pass-by-value*

For an argument of an array type, the value of the argument is a *reference* to an array; this reference value is passed to the method. Semantically, it can be best described *as pass-by-sharing*, that is, the array in the method is the same as the array being passed. Thus, if you change the array in the method, you will see the change outside the method.

```
public class Test {
    public static void main(String[] args) {
        int x = 1;
        int[] y = new int[10];
        m(x, y);
    }
    public static void m(int number, int[] numbers) {
        number = 1001;
        numbers[0] = 5555;
    }
}
```

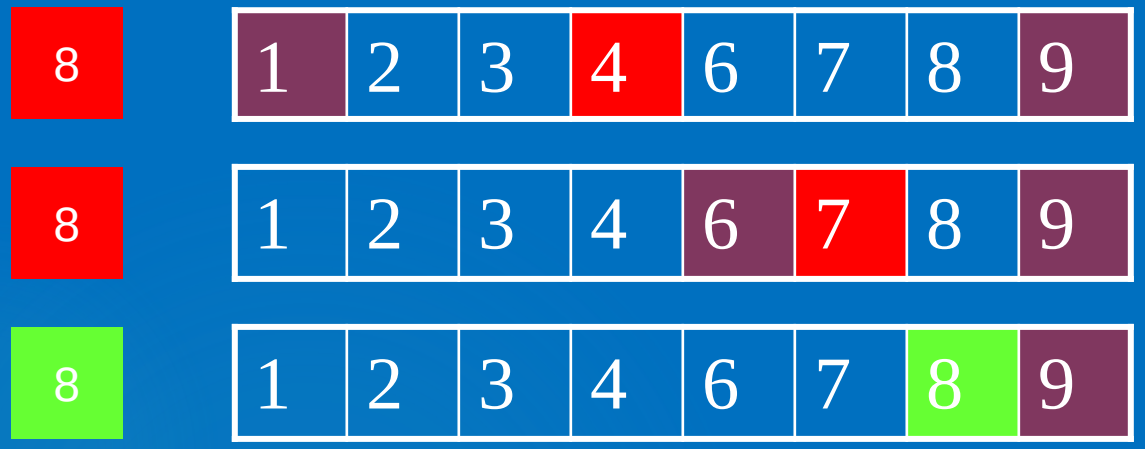


Linear Search


```
for (int i = 0; i < list.length; i++) {
    if (key == list[i])
    return i;
}
return -1;
```

Binary Search

key list



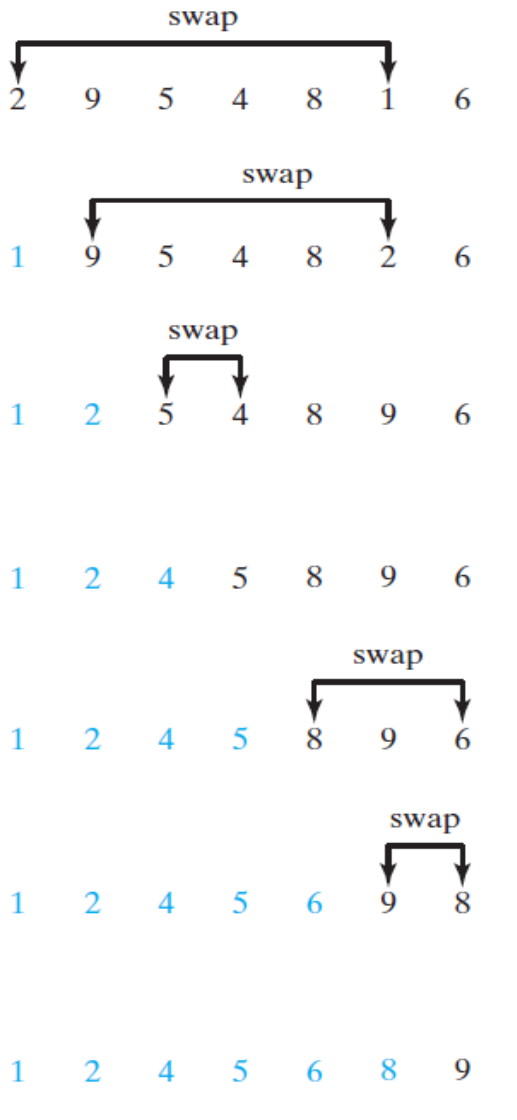

```
int low = 0;
int high = list.length - 1;
while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
}
// Now high < low
return -low - 1;
```

Binary Search using *java.util.Arrays.binarySearch*

Searches the specified array for the specified value using the *binary search algorithm*. The array must be sorted (as by the *sort(type[])* method) prior to making this call. If it is not sorted, the results are undefined.

```
// public static int binarySearch(type[] a, type key)
public static void main(String[] args) {
    int[] list = {1,2,3,4,5,6};
    int index = Arrays.binarySearch(list, 3)
    System.out.println(index);
}
```

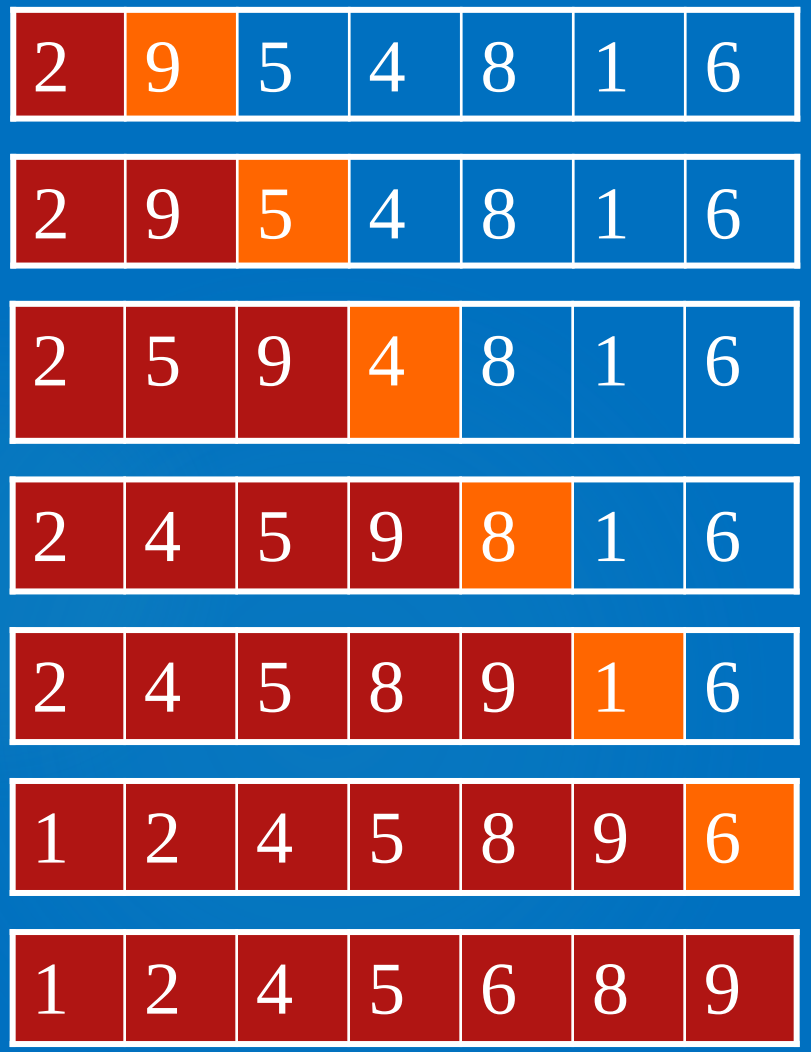
Selection Sort



Selection sort repeatedly selects the smallest number and swaps it with the first number in the list.

```
for (int i = 0; i < list.length - 1; i++) {
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
        if (currentMin > list[j]) {
            currentMin = list[j];
            currentMinIndex = j;
        }
    }
    if (currentMinIndex != i) {
        list[currentMinIndex] = list[i];
        list[i] = currentMin;
    }
}
```

Insertion Sort




```
for (int i = 1; i < list.length; i++) {
    double curN = list[i];
    int k;
    for (k = i - 1; k >= 0 && list[k] > curN; k--) {
        list[k + 1] = list[k];
    }
    // Insert
    list[k + 1] = curN;
}
```

Sort using *java.util.Arrays.sort* Method


```
public class Test {
    public static void main(String[] args) {
        double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
        java.util.Arrays.sort(numbers);
        printArray(numbers);
    }
    public static void printArray(double[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

```
1.9 2.9 3.4 3.5 4.4 6.0
```

Command-Line Arguments

The *main* method can receive string arguments from the command line.

```
public static void main(String[] args) {
    System.out.println("The number of the arguments is " + args.length);
    for(String arg:args) {
        System.out.printf("(%s) ", arg);
    }
}
```

```
java test 1 2 a2 "db" "i am xiaoming"
```

```
The number of the arguments is 5
(1) (2) (a2) (db) (i am xiaoming)
```

Exercise

Which of the following statements are valid?

```
int i = new int(30);
double d[] = new double[30];
char[] r = new char(1..30);
int i[] = (3, 4, 3, 2);
float f[] = {2.3, 4.5, 6.6};
char[] c = new char();
```

Exercise

Identify and fix the errors in the following code:

```
public class Test {
  public static void main(String[] args) {
    double[100] r;
    for (int i = 0; i < r.length(); i++);
      r(i) = Math.random * 100;
    }
}
```

Exercise

What is the output of the following code?

```
public class Test {
  public static void main(String[] args) {
    int list[] = {1, 2, 3, 4, 5, 6};
    for (int i = 1; i < list.length; i++)
      list[i] = list[i - 1];

    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
  }
}
```

Exercise

Use the arraycopy method to copy the following array to a target array t:

```
int[] source = {3, 4, 5};
```

Once an array is created, its size cannot be changed. Does the following code resize the array?

```
int[] myList;
myList = new int[10];
// Sometime later you want to assign a new array to myList
myList = new int[20];
```

Exercise

Show the output of the following program:

```
public class Test {
  public static void main(String[] args) {
      int number = 0;
      int[] numbers = new int[1];
      m(number, numbers);
      System.out.println("number:" + number + ", numbers[0]:" + numbers[0]);
    }
  public static void m(int x, int[] y) {
    x = 3;
    y[0] = 3;
  }
}
```

Exercise

We declare the main method as `public static void main(String[] args)`, can it be replaced by one of the following lines?

```
public static void main(String args[])
public static void main(String[] x)
public static void main(String x[])
static void main(String x[])
```

Exercise

Show the output of the following program when invoked using

1. `java Test I have a dream`
2. `java Test “1 2 3”`
3. `java Test`

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Number of strings is " + args.length);
    for (int i = 0; i < args.length; i++)
    System.out.println(args[i]);
  }
}
```

Two-Dimensional Arrays

Thus far, you have used one-dimensional arrays to model linear collections of elements. You can use a *two-dimensional* array to represent a *matrix* or a *table*. For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

Declare/Create Two-dimensional Arrays


```
// Declaring: elementType[][] arrayRefVar;
int[][] matrix;
matrix = new int[5][5]; // Creating
matrix[2][1] = 7; // Accessing
int[][] array = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //Initializer
```

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

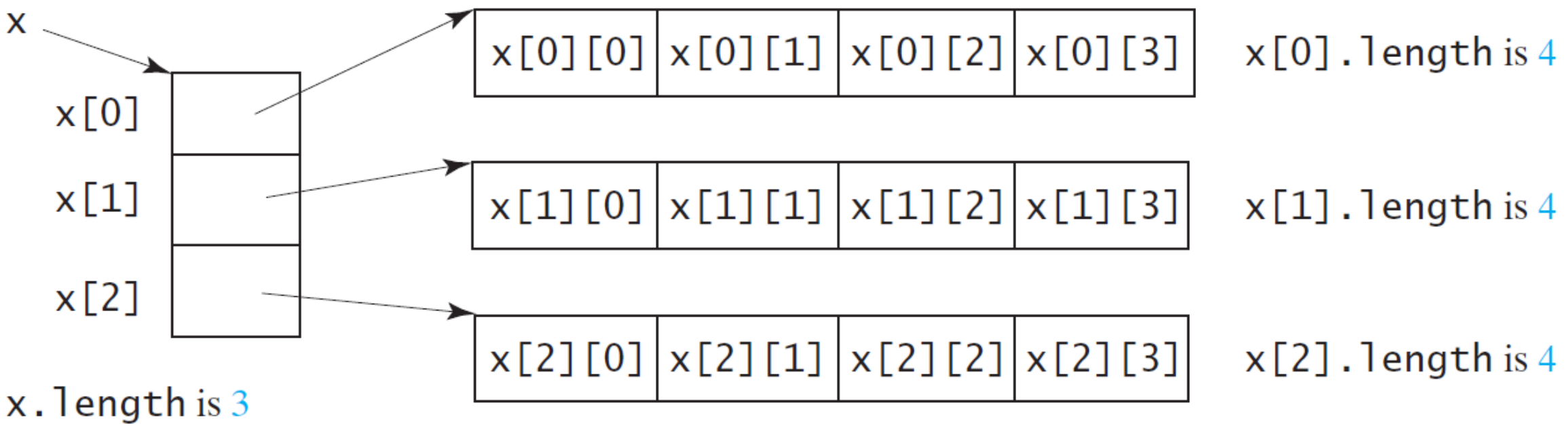
```
matrix[2][1] = 7;
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

Structure and Size of Two-dimensional Arrays

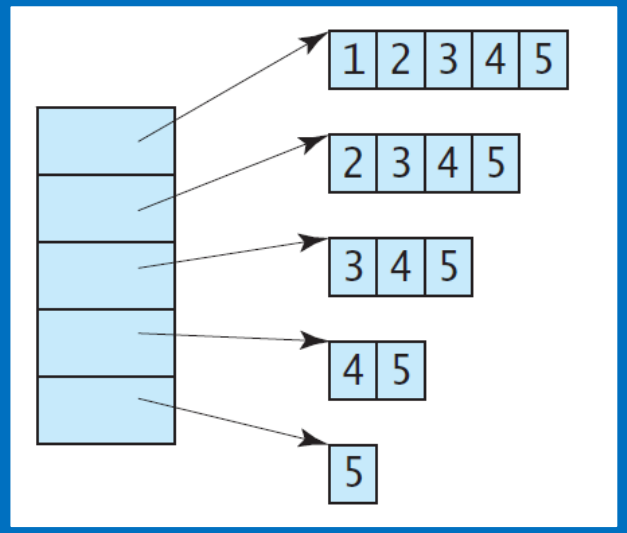
A *two-dimensional* array is a *one-dimensional* array in which each element is another one-dimensional array. For example: `int[][] x = new int[3][4];`



Ragged Arrays

Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```




```
matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1
```

Processing Two-Dimensional Arrays

- Initializing arrays with input values.
- Initializing arrays with random values.
- Printing arrays.
- Summing all elements.
- Summing elements by column.
- Which row has the largest sum?
- Random shuffling.

Two-Dimensional Arrays Processing Codes

Case 3: Grading a Multiple-Choice Test

Suppose you need to write a program that grades multiple-choice tests. Assume there are *eight students* and *ten questions*, and the answers are stored in a two-dimensional array. Each row records a student's answers to the questions. Your program grades the test and displays the result. It compares each student's answers with the key, counts the number of correct answers, and displays it.

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Key to the Questions:

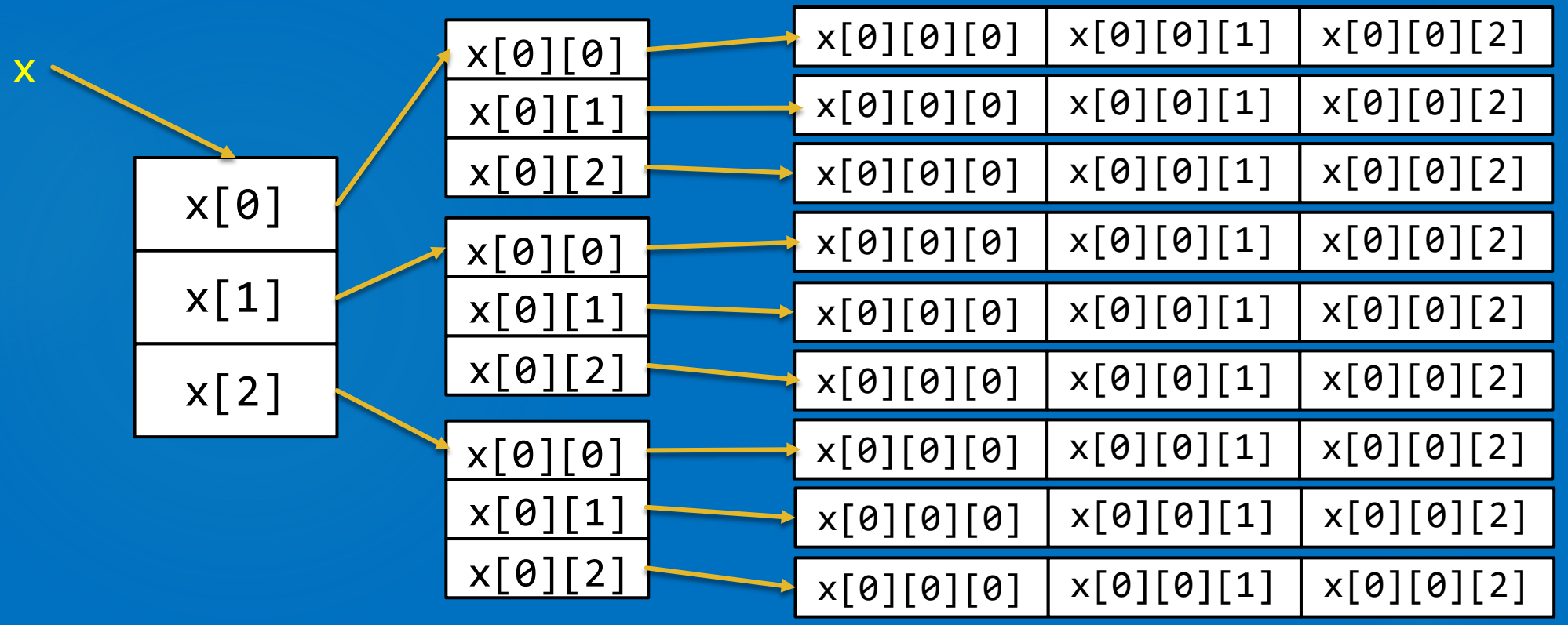
	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

GradeExam

Multidimensional Arrays

A *two-dimensional* array consists of an array of *one-dimensional* arrays and a *three dimensional* array consists of an array of *two-dimensional* arrays.

```
int[][][] x = new int[3][3][3];
```



Exercise

What is the output of the following code?

```
int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);
```

Exercise

Show the output of the following code:

```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
for (int i = array.length - 1; i >= 0; i--) {
  for (int j = array[i].length - 1; j >= 0; j--)
    System.out.print(array[i][j] + " ");
  System.out.println();
}
```

Exercise

Show the output of the following code:

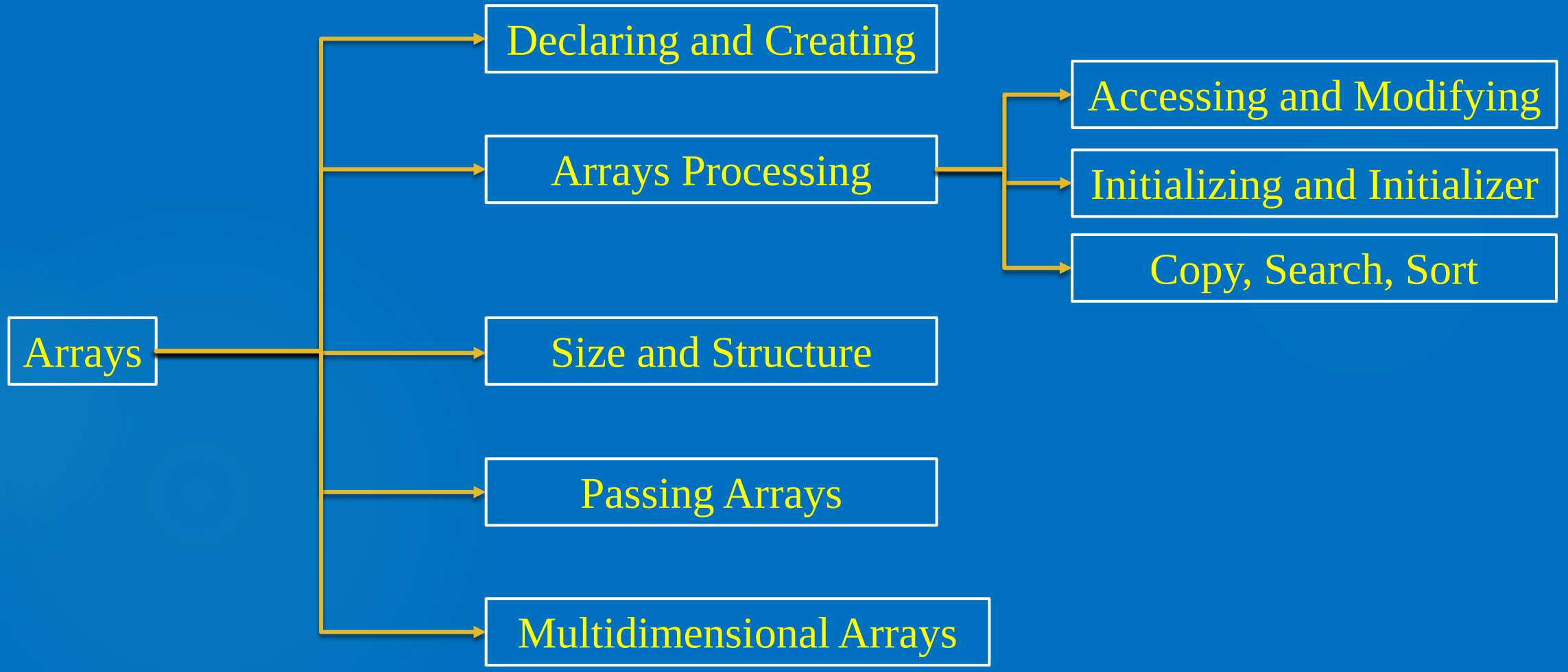
```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
int sum = 0;
for (int i = 0; i < array.length; i++)
  sum += array[i][0];
System.out.println(sum);
```

Exercise

Show the output of the following code:

```
public class Test {
  public static void main(String[] args) {
    int[][] array = {{1, 2, 3, 4}, {5, 6, 7, 8}};
    System.out.println(m1(array)[0]);
    System.out.println(m1(array)[1]);
  }
  public static int[] m1(int[][] m) {
    int[] result = new int[2];
    result[0] = m.length;
    result[1] = m[0].length;
    return result;
  }
}
```

Review


Arrays
Declaring and Creating
Arrays Processing
Accessing and Modifying
Initializing and Initializer
Copy, Search, Sort
Size and Structure
Passing Arrays
Multidimensional Arrays