AP Computer Science A
Java Programming Essentials [Ver. 2.0]

Unit 3: Basic Data Structure

WEEK 11: CHAPTER 7 ARRAYS (PART 2: ARRAY PROCESSING)

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Array Processing III: Enhanced Loop, range function, occurrence array, non-recurring array.
- Array Class: Utility Class to serve arrays.
  - Shallow Copy Versus Deep Copy: copy of reference or copy of objects.
  - Passing array as parameter and returning array as return value.
  - Array Searching Techniques: Linear Search/Binary Search
- Student Score Report

Array Processing III

LECTURE 1

Enhanced for Loop (for-each loop)

- JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the **Array** myList: (not limited to array type, but all class implement Iterator Interface, such as **ArrayList**)

```
for (double value: myList)
   System.out.println(value);
```

Enhanced for Loop (for-each loop)

- In general, the syntax is

```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

- You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Demonstration Program

Rewrite the Stats02.java into Enhanced For-Loop format

ARRAYPROCESSINGIII.JAVA

Array Processing III

- When index needed to traverse through an array, (**end** not included)
  - **range(begin, end, step):** create an array of indexes from **begin** to **end-1(end+1)** and each spaced by **step**.
  - **range(begin, end):** create an array of indexes from **begin** to **end-1 (end+1)**and each spaced by **1** / **-1**.
  - **range(end):** create an array of indexes from **0** to **end-1** and each spaced by **1**. (range() array is a very good integer arithmetic sequence generator.)

Demonstration Program

RANGE.JAVA

Non-recurring Set List

- Use an array as a set.
- Use a running index **top** as the length keeper. It can also be used as the **next empty spot** keeper.
- When the set array is generated, the occurrence array is also generated. (If an item has never shown up, we add it to the list. If an item has shown up, then we increase its occurrence count.)

```java
int len = ary.length;
int top = 0;
int[] occurrence = new int[len];
int[] set = new int[len];

for (int i=0; i<ary.length; i++){
    boolean found = false;
    for (int j=0; j<top; j++){
        if (set[j]==ary[i]) {
            found = true;
            occurrence[j]++;
        }
    }
    if (!found){
        set[top] = ary[i];
        occurrence[top] = 1;
        top++;
    }
}
```

```
Source Array: [8, 6, 5, 3, 6, 7, 7, 4, 4, 5, 3, 1, 9, 0, 6, 6]
Set Array:            [8, 6, 5, 3, 7, 4, 1, 9, 0]
Occurrence Array: [1, 4, 2, 2, 2, 2, 1, 1, 1]
```

Demonstration Program

ARRAYSET.JAVA

Demo Program: LetterCount.java

LECTURE 1

Demo Program: LetterCount.java

- Read in the usindependence.txt file (US. Declaration of Independence) and count the occurrence of each letter (ignore case) in the file.

Pseudo Code

```
1. Read in the usindependence.txt file line by line and append all the input
string together into a bigger string named text.

2. Convert text into all uppercase letter.

3. Traverse through each character in text string.

    if the character is not a uppercase letter or space, then remove the letter.

4. Traverse through each character the text string again.
   If the character is in the alphabet string (all uppercase letter), then increase
   the letter count variable of current letter.

5. Print out each letter's occurrence times in usindependence.txt
```

What we learned from this demo program?

1. Occurrence array (or frequency array) of the alphabet letter. If you want to count the occurrence of some data set (collection of data) such digits, letter, names, words, you should set up an array of occurrence (or frequency array). And, then, traverse through the mother sample set (in this case, ***text*** string) and tally for the occurrence of each letter.

2. Convert character ch to string “”+ch to empty string “” in ***text*** string.

Demonstration Program

LETTERCOUNT.JAVA

Demo Program: Bible's Word Occurrence Count

LECTURE 1

Demo program: WordCount.java

- Read in the bible.txt file. And count the total number of words (in more accurate way), calculate each word’s occurrence and print out these statistics for bible.
- The output information will be shown on both screen and a text file “bibilecount.txt”.

Rough Pseudo Code

1. Read in the **bible.txt** file line by line and then append all of them into a long string text.

2. Convert all words in text string to lowercase.

3. Clean up all of numbers, symbols and special characters.

4. use **text.split()** to split the text string to an array of strings (words). Set found to **false**.

5. For all words, set **found** to **false** first. If it is not in the **dict** (dictionary list), then add it in **dict** and set this word's occurrence counter (**wordCount[i]** to 1). Move top pointer of **dict** to next one.   If it is in the **dict**, then increase the word's **wordCount** by 1. and set **found** to be true (no need to go through the whole dictionary dict).

6. For all words in dictionary (**dict**), print out its **wordCount** and its string value.

7. print the dictionary's size (total words used in bible.) and the total words count in bible.

Demonstration Program

WORDCOUNT.JAVA

Arrays Class

(Non-AP Topics)

LECTURE 1

Arrays Class

- Class **Arrays** helps you avoid reinventing the wheel by providing static methods for common array manipulations
- Methods Include
  - sort(array) : Arranges array elements into increasing order.
  - binarySearch(array , element) : Determines whether an array contains a specific value and, if so, returns where the value is located.
  - equal(array) : Compares arrays.
  - fill(array , element) : Places Values into an array.
  - toString() : Converts array to String.

The Arrays Class (Non-AP Topic)

- Java contains a special utility class that makes it easier for you to perform many often used array operations like copying and sorting arrays, filling in data, searching in arrays etc. The utility class is called Arrays and is located in the standard Java package java.util. Thus, the fully qualified name of the class is:

```
java.util.Arrays
```

The Arrays Class (Non-AP Topic)

- I will cover a few of the methods found in this class in the following sections. Remember, in order to use java.util.Arrays in your Java classes you must import it. Here is how importing java.util.Arrays could look in a Java class of your own:

```
package myjavaapp;
import java.util.Arrays;
public class MyClass{
 public static void main(String[] args) {
  }
}
```

- Notice the import java.util.Arrays; statement in bold. It is this statement that imports the classjava.util.Arrays into your Java class.

Copying Arrays (Copying an Array by Iterating the Array)

You can copy an array into another array in Java in several ways.

- The first way to copy an array in Java is to iterate through the array and copy each value of the source array into the destination array. Here is how copying an array looks using that method:

- First two int arrays are created. Second, the source array is initialized with values from 0 to 9 (0 to the length of the array minus 1). Third, each element in the source array is copied into the destination array.

```
int[] source = new int[10];
int[] dest = new int[10];
for (int i=0; i < source.length; i++) {
   source[i] = i;
}
for (int i=0; i < source.length; i++) {
   dest[i] = source[i];
}
```

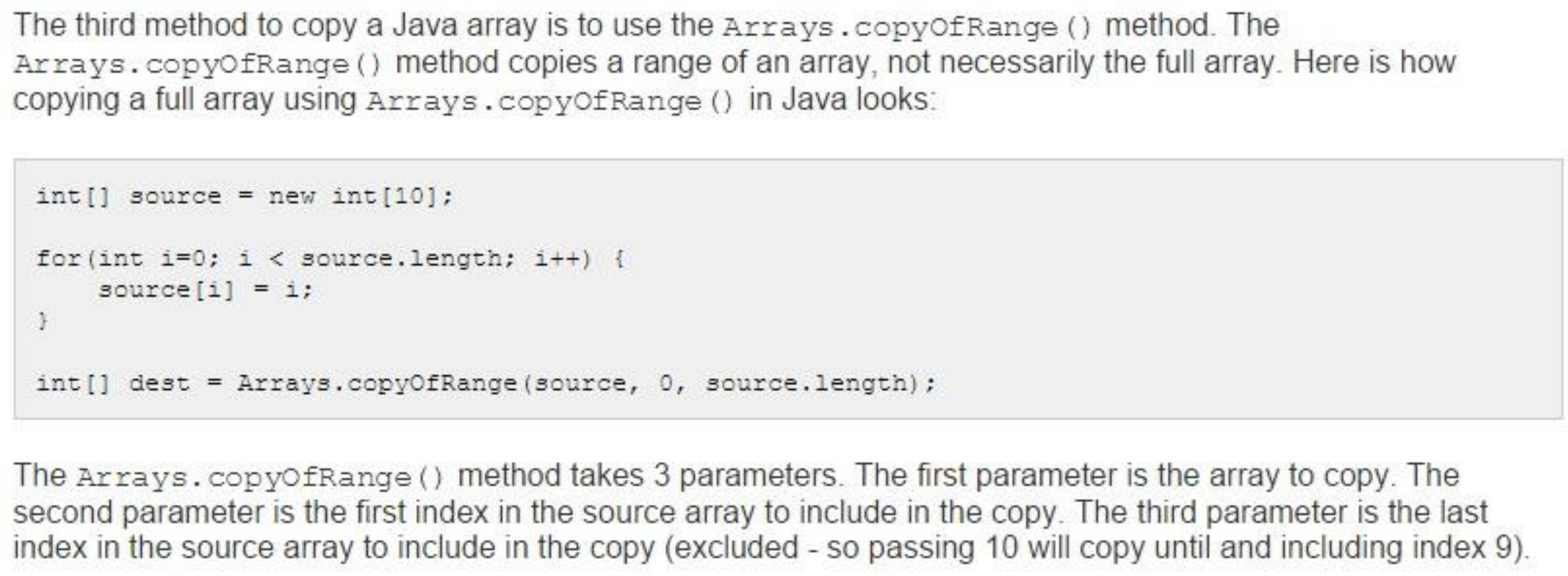
Copying an Array Using Arrays.copyOf()

The second method to copy a Java array is to use the Arrays.copyOf() method. Here is how copying an array using Arrays.copyOf() looks:

```
int[] source = new int[10];

for(int i=0; i < source.length; i++) {
    source[i] = i;
}

int[] dest = Arrays.copyOf(source, source.length);
```

- The Arrays.copyOf() method takes 2 parameters. The first parameter is the array to copy. The second parameter is the length of the new array. This parameter can be used to specify how many elements from the source array to copy.

Copying an Array Using Arrays.copyOfRange()

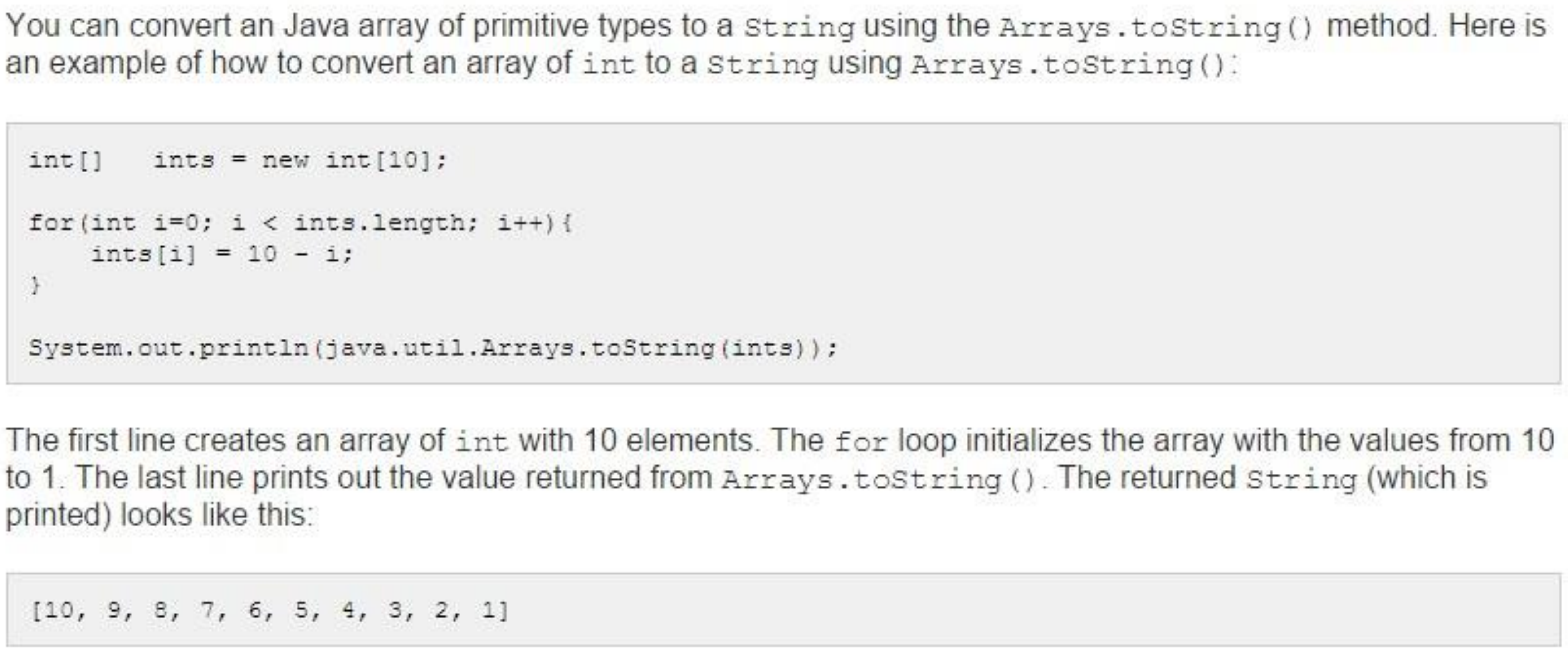


The third method to copy a Java array is to use the `Arrays.copyOfRange()` method. The `Arrays.copyOfRange()` method copies a range of an array, not necessarily the full array. Here is how copying a full array using `Arrays.copyOfRange()` in Java looks:

```
int[] source = new int[10];

for(int i=0; i < source.length; i++) {
    source[i] = i;
}

int[] dest = Arrays.copyOfRange(source, 0, source.length);
```

The `Arrays.copyOfRange()` method takes 3 parameters. The first parameter is the array to copy. The second parameter is the first index in the source array to include in the copy. The third parameter is the last index in the source array to include in the copy (excluded - so passing 10 will copy until and including index 9).

Converting Arrays to Strings With Arrays.toString()



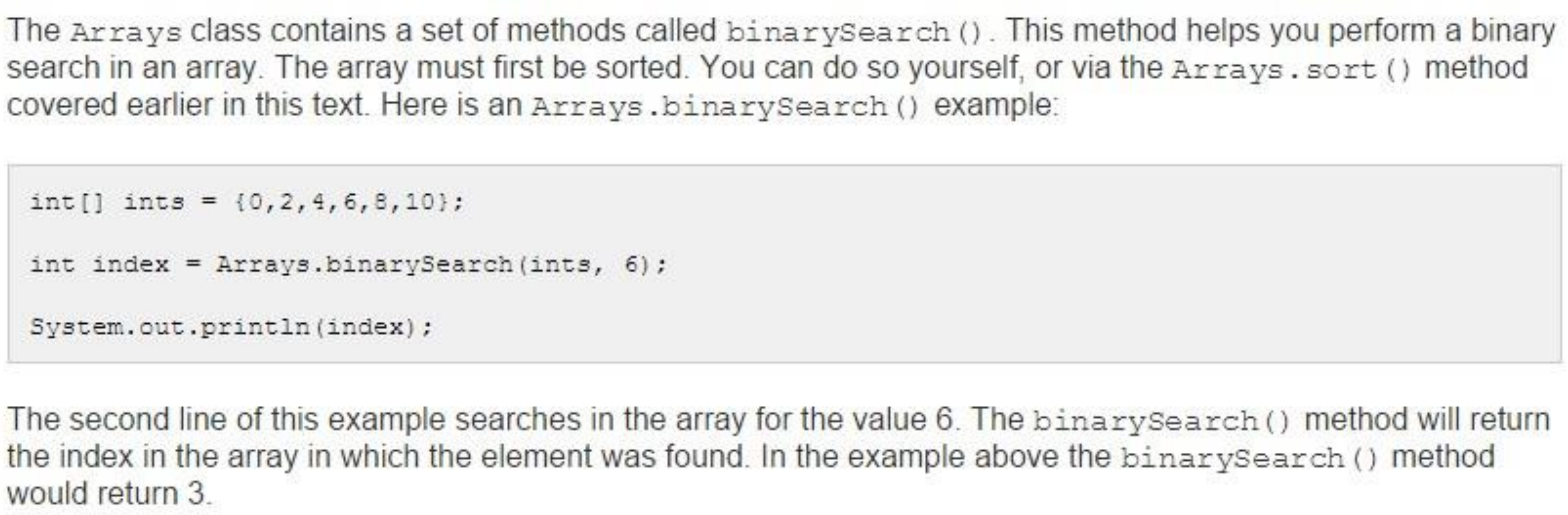
You can convert an Java array of primitive types to a `String` using the `Arrays.toString()` method. Here is an example of how to convert an array of `int` to a `String` using `Arrays.toString()`:

```
int[]   ints = new int[10];

for(int i=0; i < ints.length; i++){
    ints[i] = 10 - i;
}

System.out.println(java.util.Arrays.toString(ints));
```

The first line creates an array of `int` with 10 elements. The `for` loop initializes the array with the values from 10 to 1. The last line prints out the value returned from `Arrays.toString()`. The returned `String` (which is printed) looks like this:

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Searching Arrays with Arrays.binarySearch()

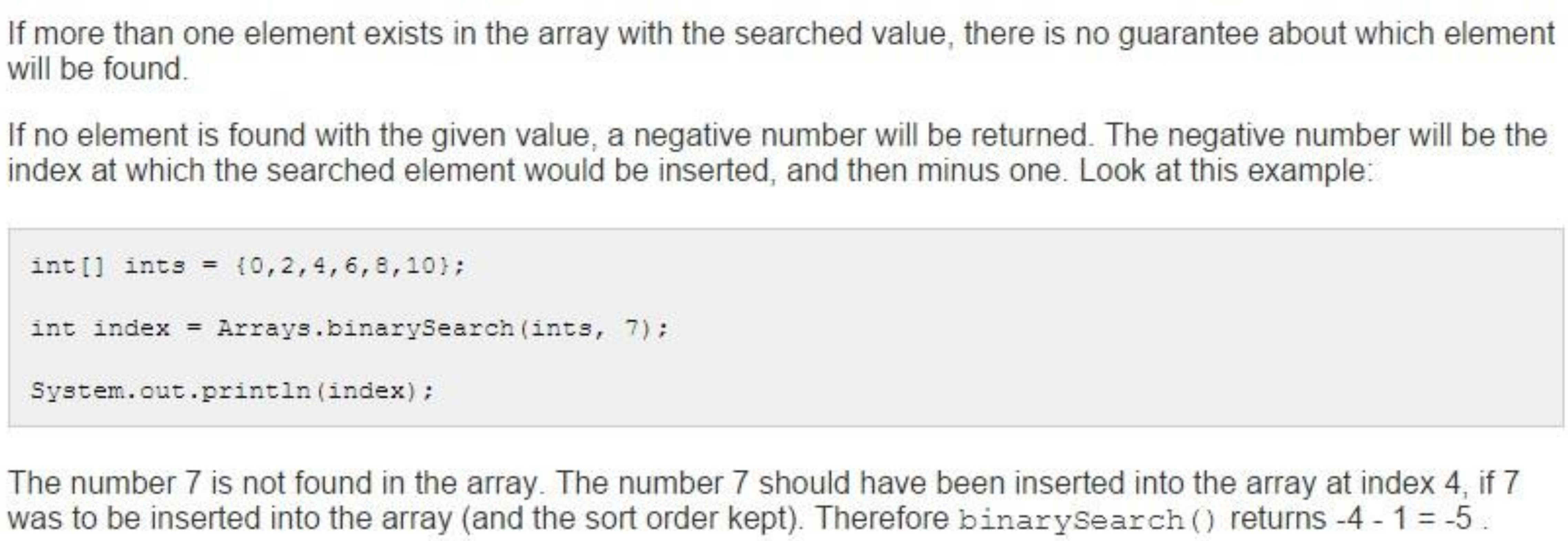


The `Arrays` class contains a set of methods called `binarySearch()`. This method helps you perform a binary search in an array. The array must first be sorted. You can do so yourself, or via the `Arrays.sort()` method covered earlier in this text. Here is an `Arrays.binarySearch()` example:

```
int[] ints = {0,2,4,6,8,10};

int index = Arrays.binarySearch(ints, 6);

System.out.println(index);
```

The second line of this example searches in the array for the value 6. The `binarySearch()` method will return the index in the array in which the element was found. In the example above the `binarySearch()` method would return 3.

Searching Arrays with Arrays.binarySearch()



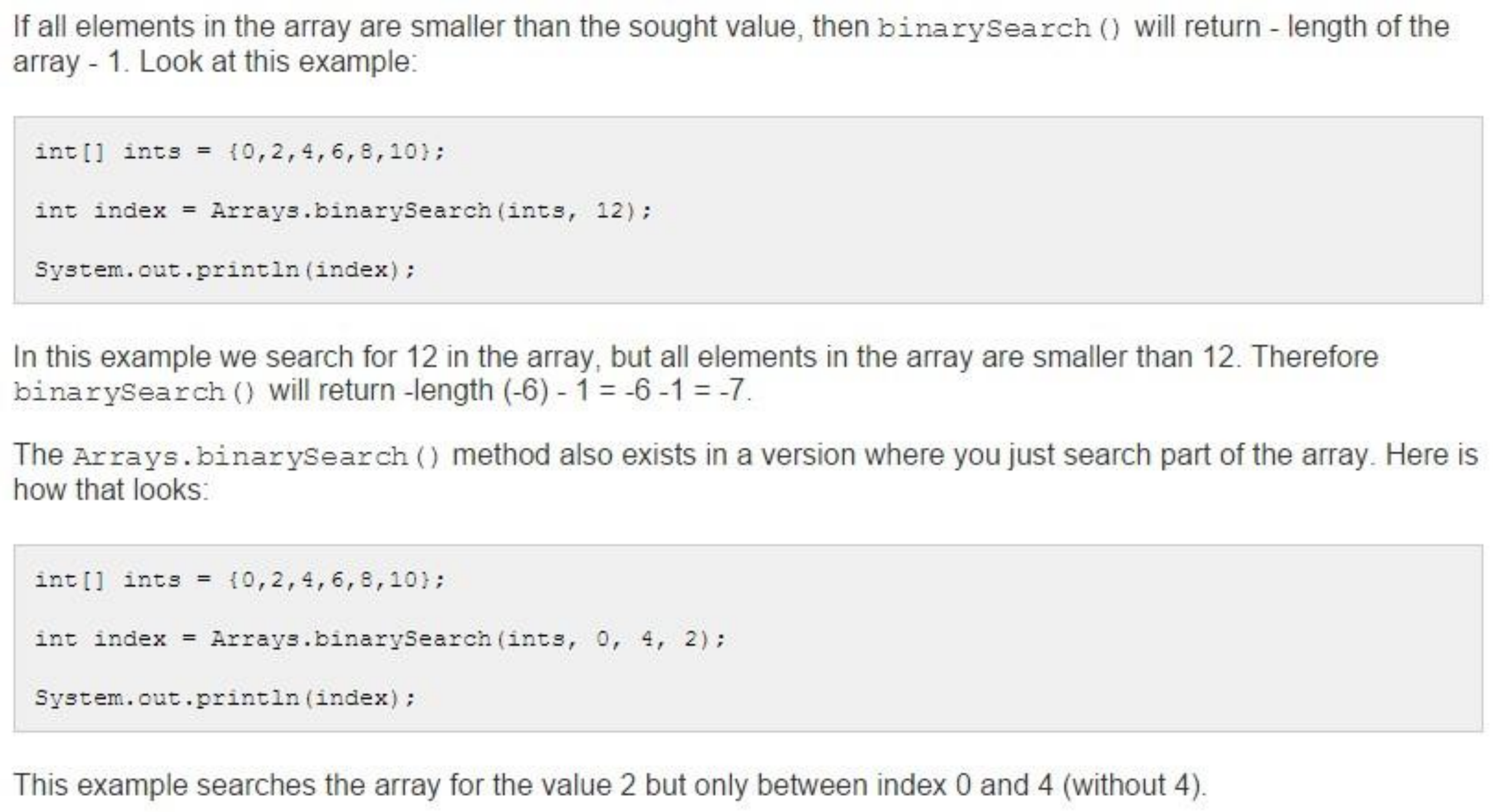
If more than one element exists in the array with the searched value, there is no guarantee about which element will be found.

If no element is found with the given value, a negative number will be returned. The negative number will be the index at which the searched element would be inserted, and then minus one. Look at this example:

```
int[] ints = {0,2,4,6,8,10};

int index = Arrays.binarySearch(ints, 7);

System.out.println(index);
```

The number 7 is not found in the array. The number 7 should have been inserted into the array at index 4, if 7 was to be inserted into the array (and the sort order kept). Therefore `binarySearch()` returns -4 - 1 = -5 .

Searching Arrays with Arrays.binarySearch()



If all elements in the array are smaller than the sought value, then `binarySearch()` will return - length of the array - 1. Look at this example:

```
int[] ints = {0,2,4,6,8,10};

int index = Arrays.binarySearch(ints, 12);

System.out.println(index);
```

In this example we search for 12 in the array, but all elements in the array are smaller than 12. Therefore `binarySearch()` will return -length (-6) - 1 = -6 -1 = -7.

The `Arrays.binarySearch()` method also exists in a version where you just search part of the array. Here is how that looks:

```
int[] ints = {0,2,4,6,8,10};

int index = Arrays.binarySearch(ints, 0, 4, 2);

System.out.println(index);
```

This example searches the array for the value 2 but only between index 0 and 4 (without 4).

Searching Arrays with Arrays.binarySearch()

This version of `binarySearch()` works just like the other version, except in the cases where no matching element is found. If no element is found matching within the index interval, then `binarySearch()` will still return the index of where the value should have been inserted. But, if all values in the interval are smaller than the sought value, `binarySearch()` will return -toIndex -1 , and not -array length - 1. Thus, this `binarySearch()` example

```
int[] ints = {0,2,4,6,8,10};

int index = Arrays.binarySearch(ints, 0, 4, 12);
```

will return -5 and not -7 like `binarySearch(ints, 12)` would have.

List of Arrays Class Methods

Methods (Class static Methods):

Arrays.copyOf()

Arrays.copyOfRange()

Arrays.toString()

Arrays.sort()

Arrays.fill()

Arrays.binarySearch()

Arrays.equals()

Array Class
java.util.Arrays class (equals)

- You can use the **equals** method to check whether two arrays are strictly equal. Two arrays are strictly equal if their corresponding elements are the same. In the following code, list1 and list2 are equal, but list2 and list3 are not.

```
int[] list1 = {2, 4, 7, 10};

int[] list2 = {2, 4, 7, 10};

int[] list3 = {4, 2, 7, 10};

System.out.println(java.util.Arrays.equals(list1, list2));   //true

System.out.println(java.util.Arrays.equals(list2, list3));  //false
```

Array Class
java.util.Arrays class (fills)

- You can use the **fill** method to fill in all or part of the array. For example, the following code fills list1 with 5 and fills 8 into elements list2[1] through list2[5-1].

```
int[] list1 = {2, 4, 7, 10};

int[] list2 = {2, 4, 7, 7, 7, 10};

java.util.Arrays.fill(list1, 5);          // Fill 5 to the whole array  → {5, 5 ,5, 5}

java.util.Arrays.fill(list2, 1, 5, 8);

// Fill 8 to partial array (from index 1 to 4 (5-1))) → {2, 8, 8, 8, 8, 10}
```

Copying of Arrays

LECTURE 1

Copying Arrays

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

Five ways of Copying Arrays for Java

int[] array1 = new Array[5]; int[] array2;

(1) **Array Pointer Assignment**: (redirection, not physical copy)

```
array2 = array 1;
```

(2) **Duplicated clone copy**: (shallow or deep, **java.lang.Cloneable** interface)

```
array1.clone();
```

(3) **copy to a new Array**: (shallow)

```
array1.copyOf();
```

(4) **arraycopy() function from System Class:** (image copy, no allocation of new memory, **System** class)

see definition and example below.

(5) **array copy by traversal of array**: (deep or shallow depending on programmer.)

```
for (i=0; i< array1.length; i++) { array2[i] = array1[i]; }
```

clone()

Clone is a method that exists on all classes (and arrays) that is generally thought to produce a copy of the target object. However:

- The specification of this method deliberately does not say whether this is a shallow or deep copy (assuming that is a meaningful distinction).
- In fact, the specification does not even specifically state that clone produces a new object.

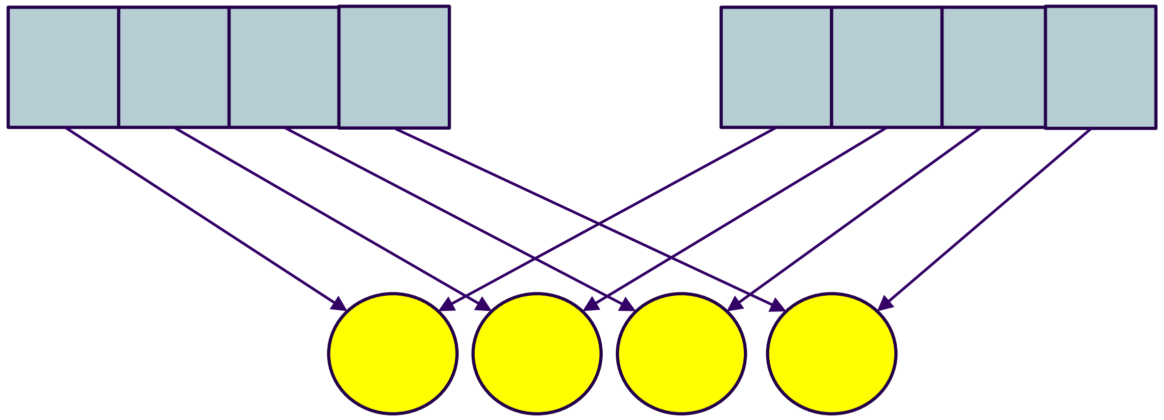
clone()

- Here's what the javadoc says:
  - *"Creates and returns a copy of this object. The precise meaning of "copy" may depend on the class of the object. The general intent is that, for any object x, the expression x.clone() != x will be true, and that the expression x.clone().getClass() == x.getClass() will be true, but these are not absolute requirements. While it is typically the case that x.clone().equals(x) will be true, this is not an absolute requirement."*

- Note, that this is saying that at one extreme the clone *might* be the target object, and at the other extreme the clone *might not* equal the original. And this assumes that clone is even supported.
- In short, clone potentially means something different for every Java class.

copyOf() method

- It produces a shallow copy, i.e. a *new* array that contains "old" references (to the same objects, those are not being copied).
- In particular, if you have nested arrays, those will not be copied. You will just get a new array whose "top level" points to the same "second level" arrays as the original did. Any changes inside those nested arrays will be reflected in both copy and original.



Shallow copy will make one array can change the contents and the other will also observe the change.

The arraycopy Utility

```
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Example:

```
System.arraycopy(sourceArray, 0, targetArray, 0,
 sourceArray.length);
```

Copy array contents to another array of same or different size.

Array copy by traversal of array:

Manual copy: can be deep copy or shallow copy. It all depends on the programmer.

Deep copy and Shallow copy may make a difference in Array of References.

Beyond AP-CS, people who has interests may find the following links useful:

https://www.cs.utexas.edu/~scottm/cs307/handouts/deepCopying.htm

https://en.wikipedia.org/wiki/Object_copying

Passing Array to Methods and Return Array from Methods

LECTURE 1

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Invoke the method

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Invoke the method

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

Anonymous Array

- The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

- There is no explicit reference variable for the array. Such array is called an ***anonymous array***.

Pass By Value

- Java uses ***pass by value*** to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.
- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.
- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

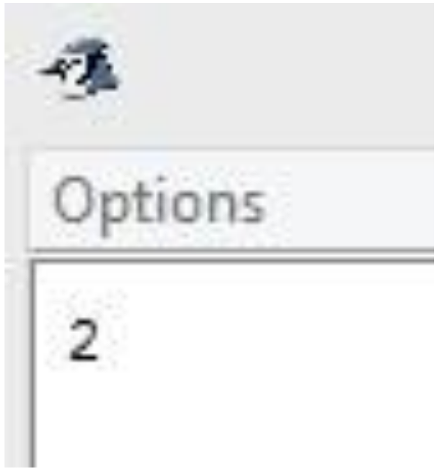
Single Array Item for **Call by Reference**

a as pointer, a[0] as body (CallByReference.java)

- Java does not support call by reference, but we can use single array item as call by reference.

```
 public static void main(String[] args){
    int[] a = {1};
    increase(a);
    System.out.println(a[0]);
 }
```

```
public static void increase(int[] b) { b[0]++; }
```

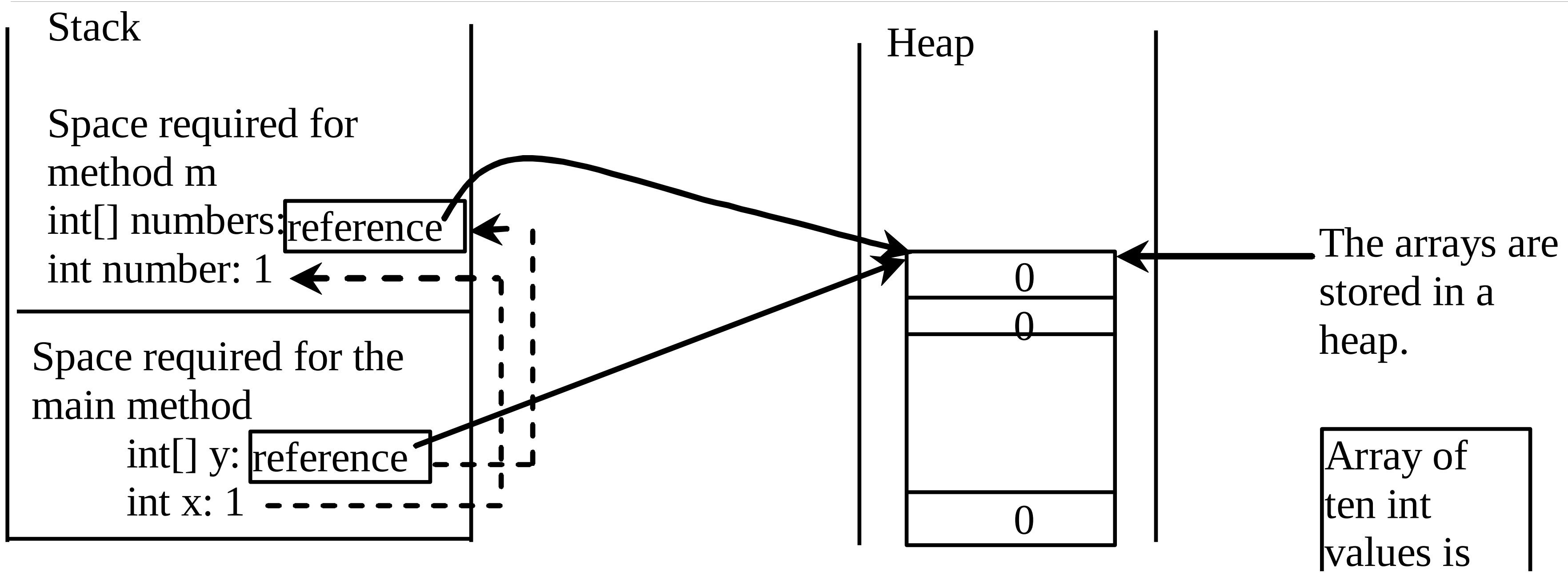


```
2
```

Simple Example

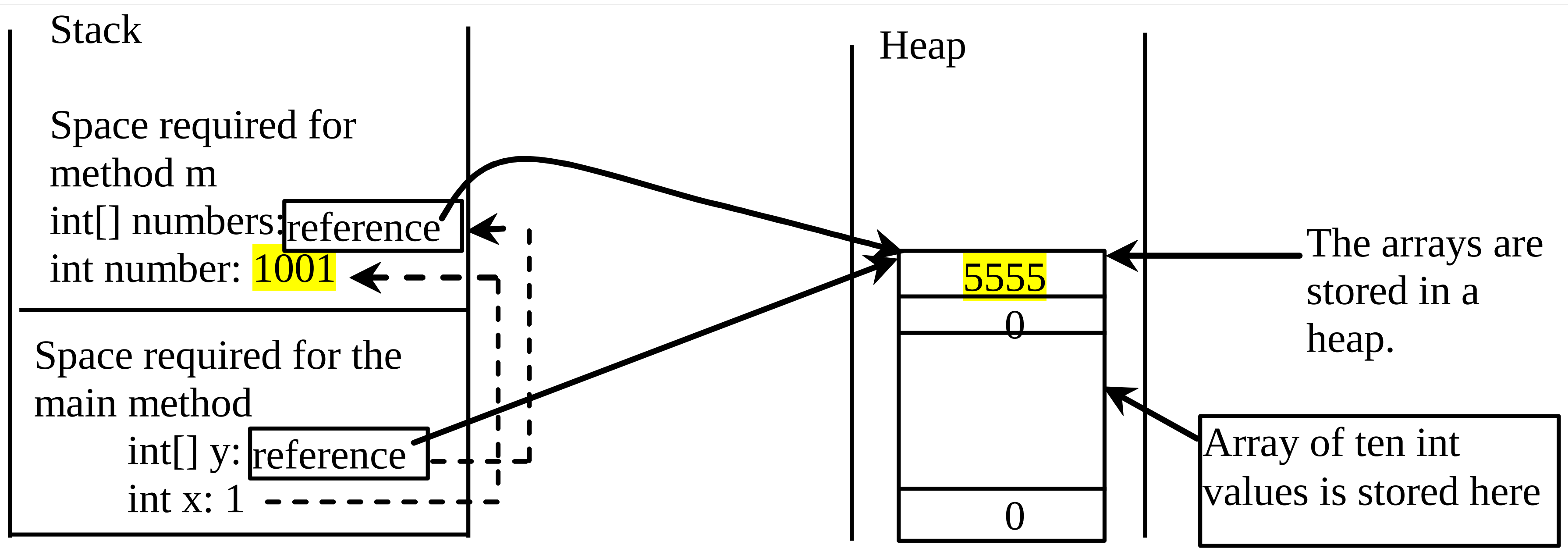
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int
  values
    m(x, y); // Invoke m with arguments x and y
    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }
  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

Call Stack


Stack
Space required for method m
int[] numbers: reference
int number: 1
Space required for the main method
int[] y: reference
int x: 1
Heap
0
0
0
The arrays are stored in a heap.
Array of ten int values is

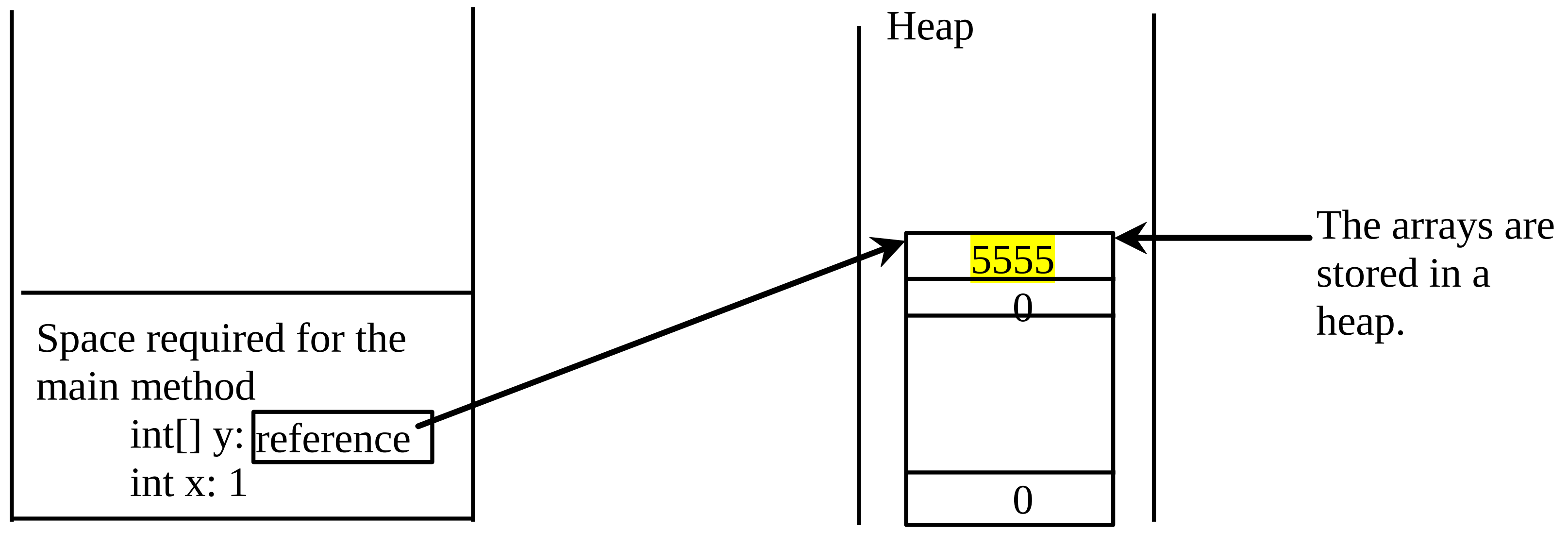
When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Call Stack


Stack
Space required for method m
int[] numbers: reference
int number: 1001
Space required for the main method
int[] y: reference
int x: 1
Heap
5555
0
0
The arrays are stored in a heap.
Array of ten int values is stored here

When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Heap


Heap
5555
0
0
The arrays are stored in a heap.
Space required for the main method
int[] y: reference
int x: 1

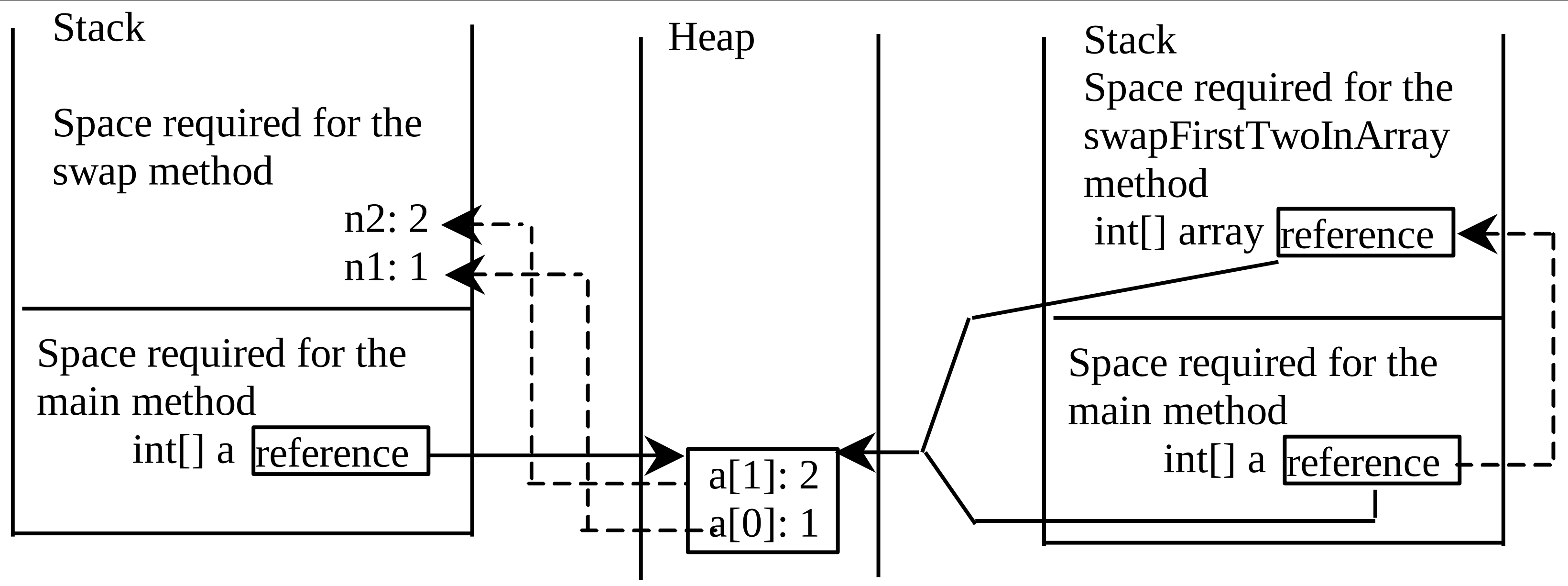
The JVM stores the array in an area of memory, called *heap*, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.

Passing Arrays as Arguments

Objective: Demonstrate differences of passing primitive data type variables and array variables.

TestPassArray.java

Example, cont.



Stack
Space required for the swap method
n2: 2
n1: 1
Space required for the main method
int[] a
reference
Heap
a[1]: 2
a[0]: 1
Stack
Space required for the swapFirstTwoInArray method
int[] array
reference
Space required for the main method
int[] a
reference

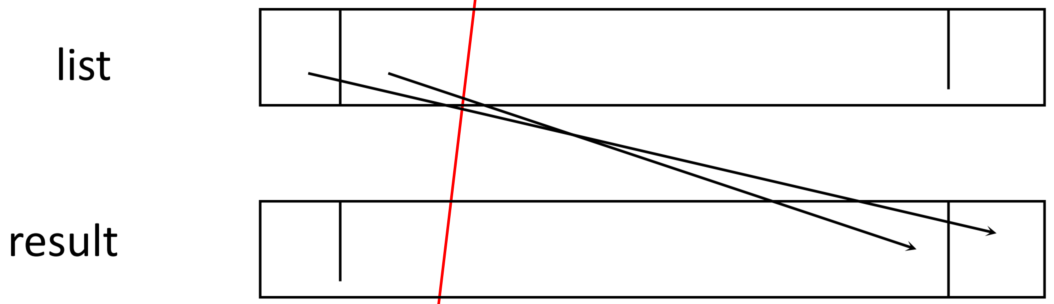
Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

The arrays are stored in a heap.

Invoke swapFirstTwoInArray(int[] array). The reference value in a is passed to the swapFirstTwoInArray method.

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



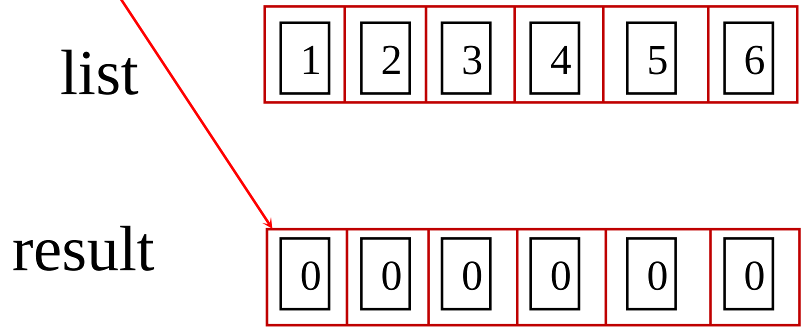
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Trace the reverse Method (reverse-copy)

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Declare result and create array

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

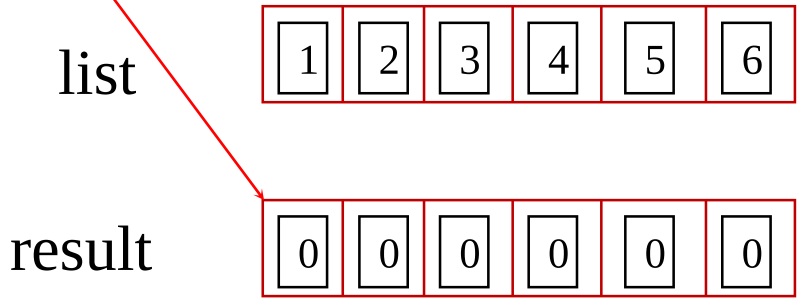


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i = 0 and j = 5

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

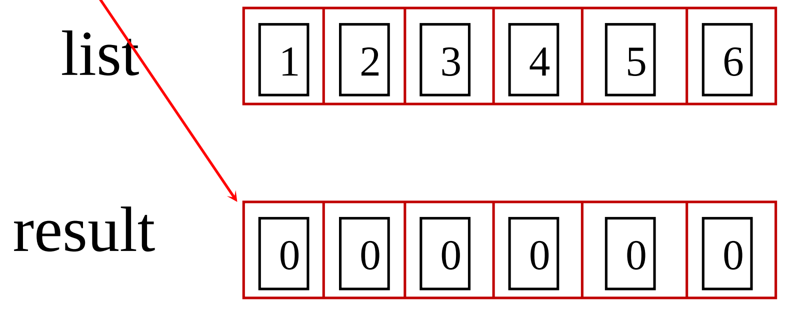


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (= 0) is less than 6

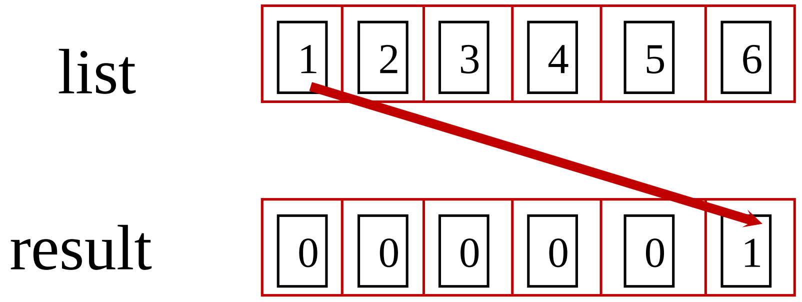
```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 0 and j = 5
Assign list[0] to result[5]



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 1 and j becomes 4

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length   1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

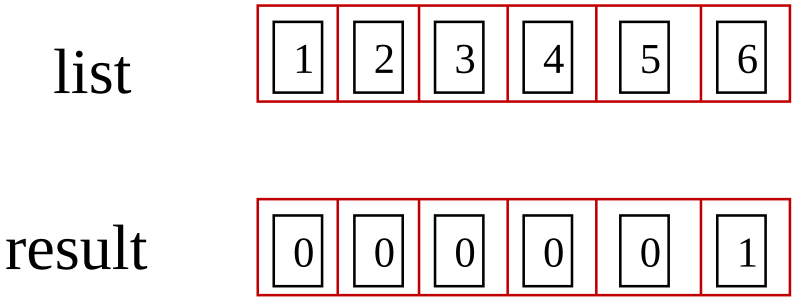
list	1	2	3	4	5	6
result	0	0	0	0	0	1

Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

i (=1) is less than 6

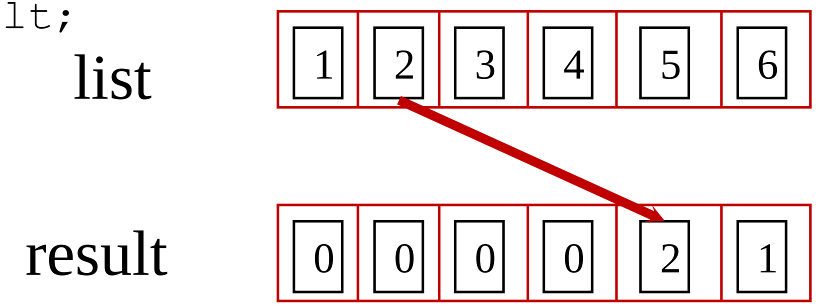


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

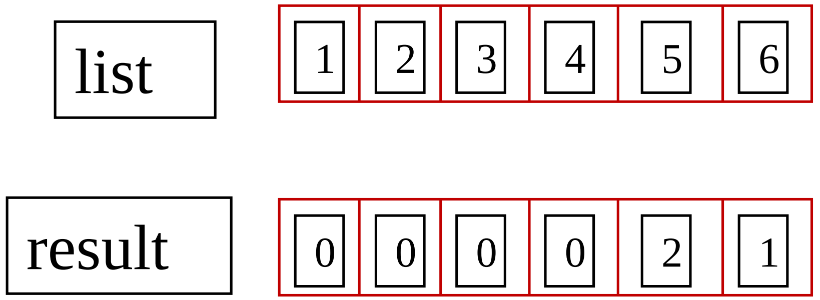
i = 1 and j = 4
Assign list[1] to result[4]



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length    1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

After this, i becomes 2 and j becomes 3

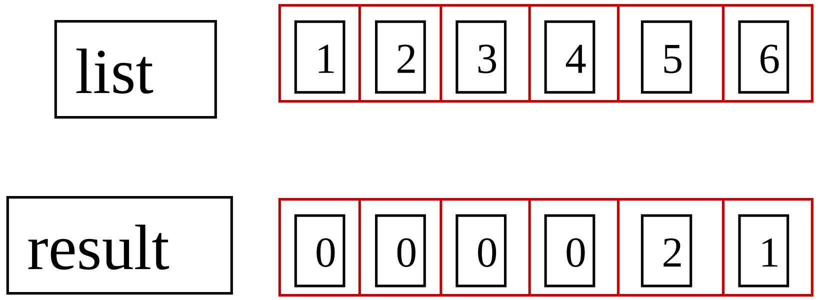


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=2) is still less than 6

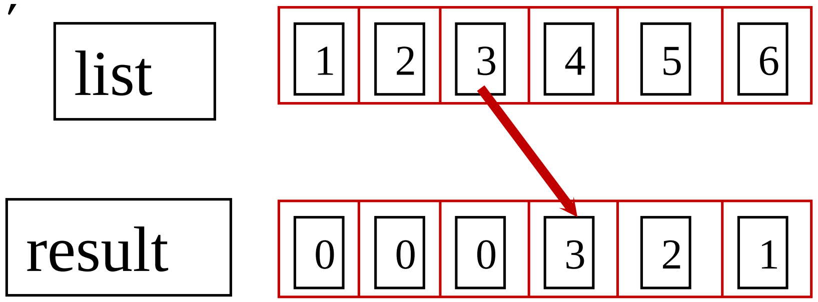
```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 2 and j = 3
Assign list[i] to result[j]

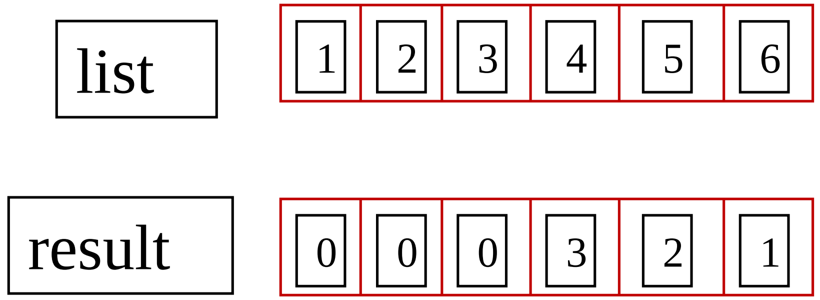


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

After this, i becomes 3 and j becomes 2

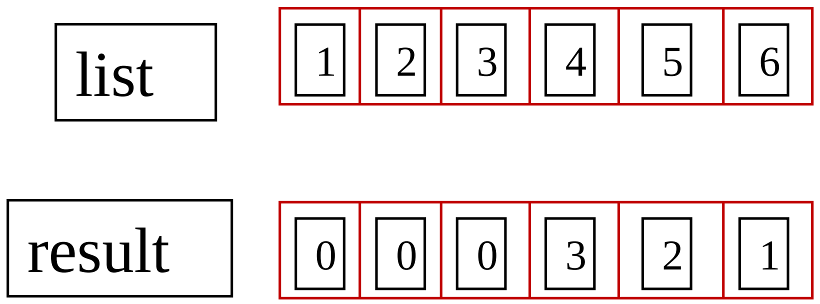
```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length   1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

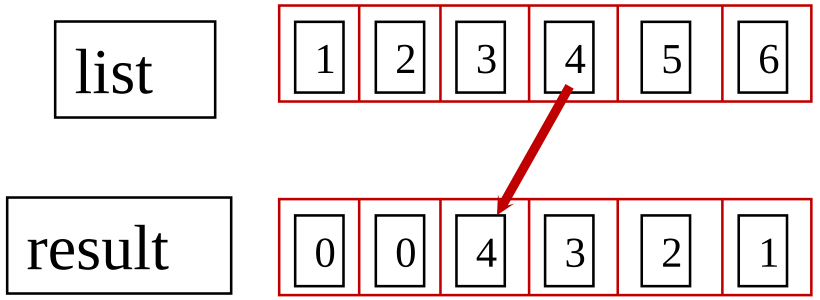
i (=3) is still less than 6



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 3 and j = 2
Assign list[i] to result[j]

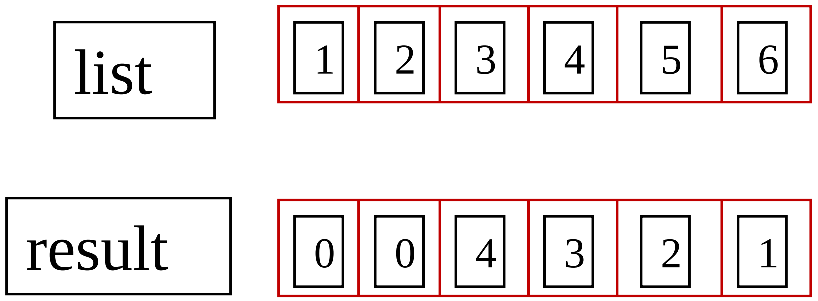


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

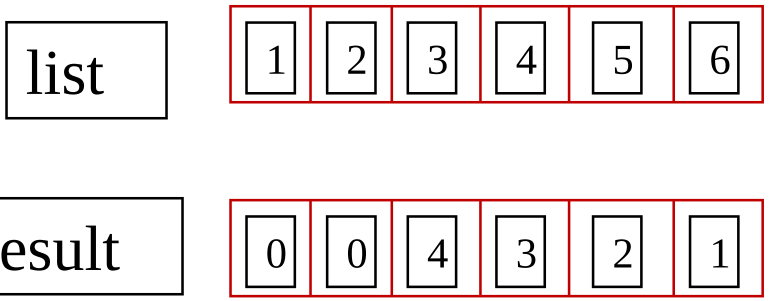
After this, i becomes 4 and j becomes 1



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length   1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

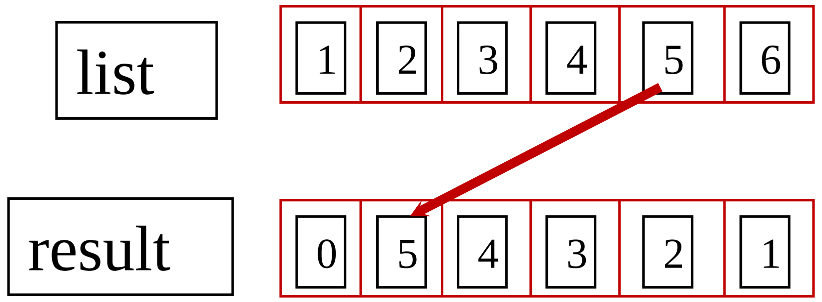
i (=4) is still less than 6



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

i = 4 and j = 1
Assign list[i] to result[j]

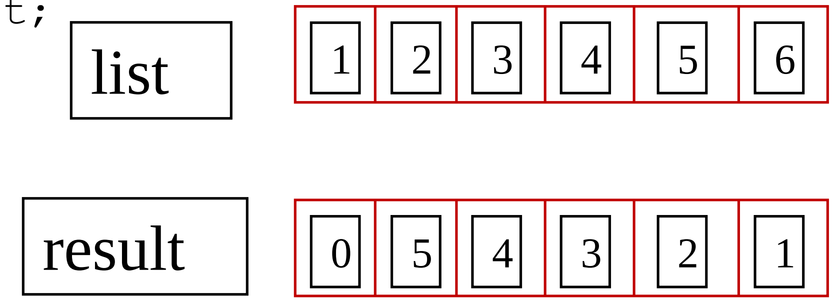


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length     1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

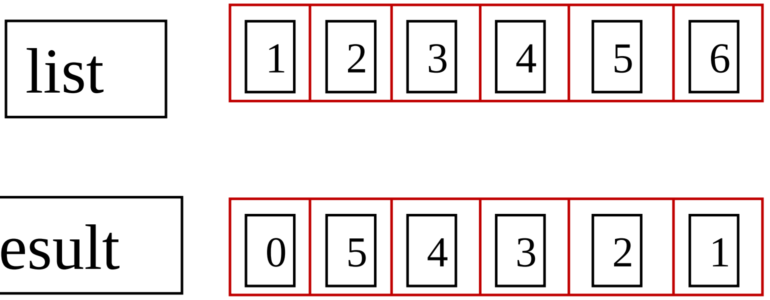
After this, i becomes 5 and j becomes 0



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

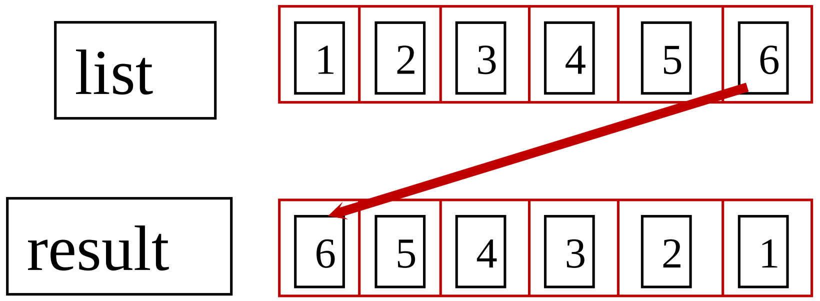
i (=5) is still less than 6



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

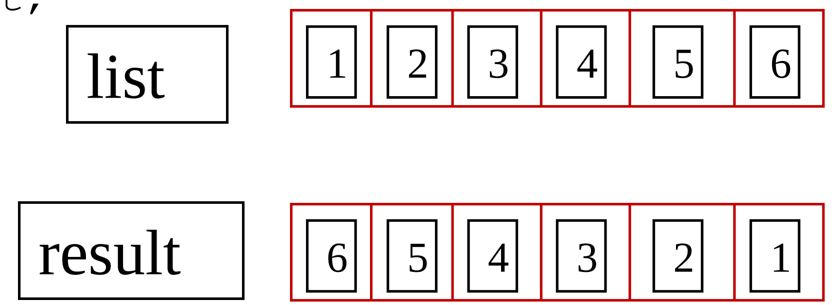
i = 5 and j = 0
Assign list[i] to result[j]



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

After this, i becomes 6 and j becomes -1

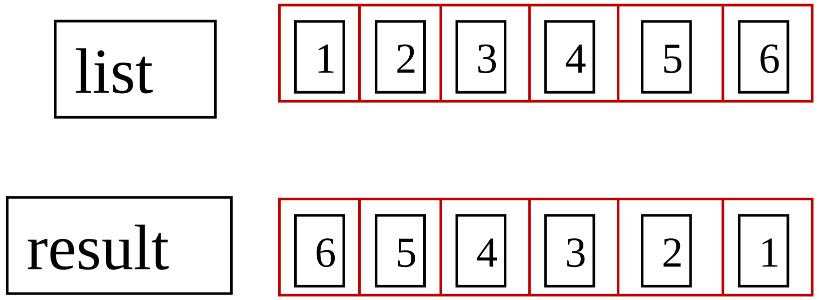


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

i (=6) < 6 is false. So exit the loop.

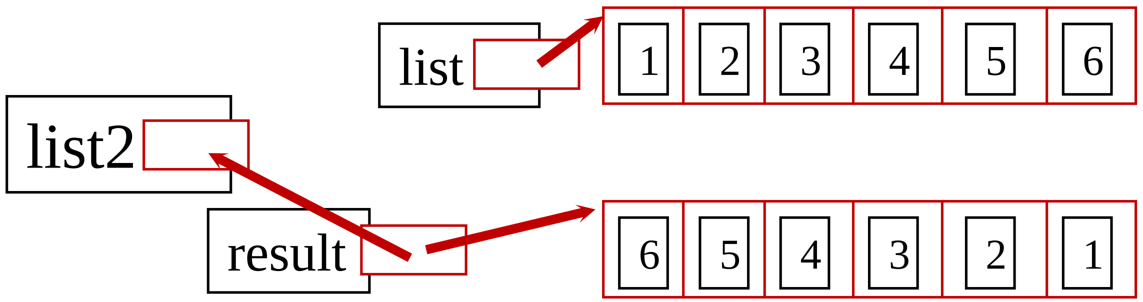
```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);


  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];

    for (int i = 0, j = result.length - 1;
         i < list.length; i++, j--) {
      result[j] = list[i];
    }

    return result;
  }
```

Return result



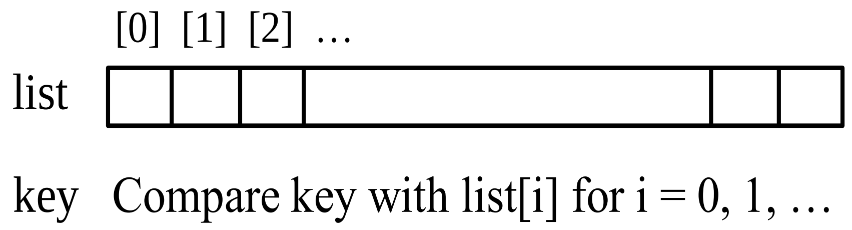
Linear Search (Basic Algorithm)

LECTURE 1

Searching Arrays

- Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, ***linear search*** and ***binary search***.

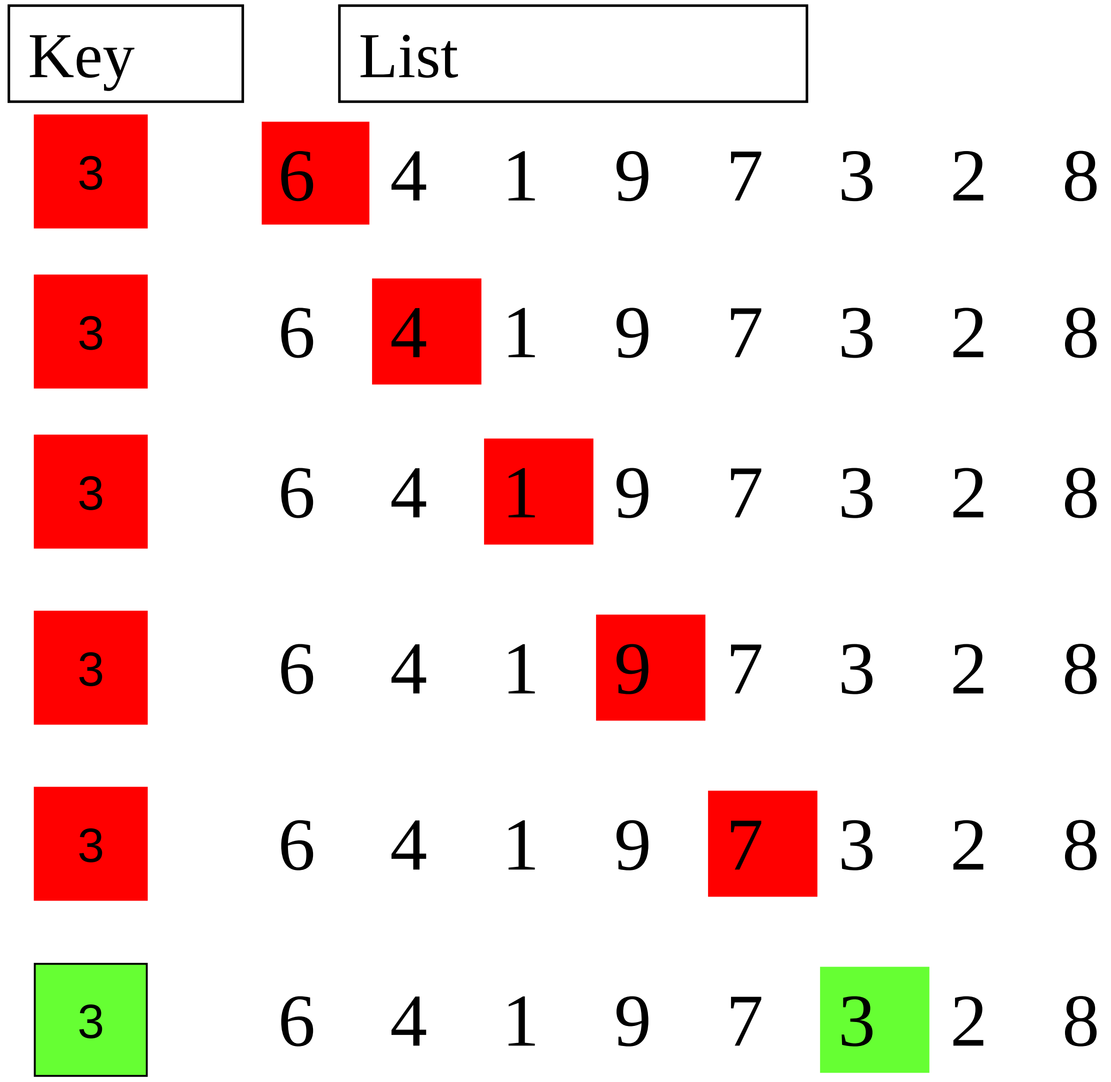
```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
```



Linear Search

- The linear search approach compares the key element, **key**, *sequentially* with each element in the array **list**. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found.
- If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns **-1**.

Linear Search Animation



From Idea to Solution

```
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

Trace the method

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Word Occurrence Count and Letter Occurrence Count in a String are a Typical Linear Search Problems.

Two linear searching exemplary program will be presented in the next two lectures.

Binary Searching Algorithm

LECTURE 1

Binary Search (on sorted array)

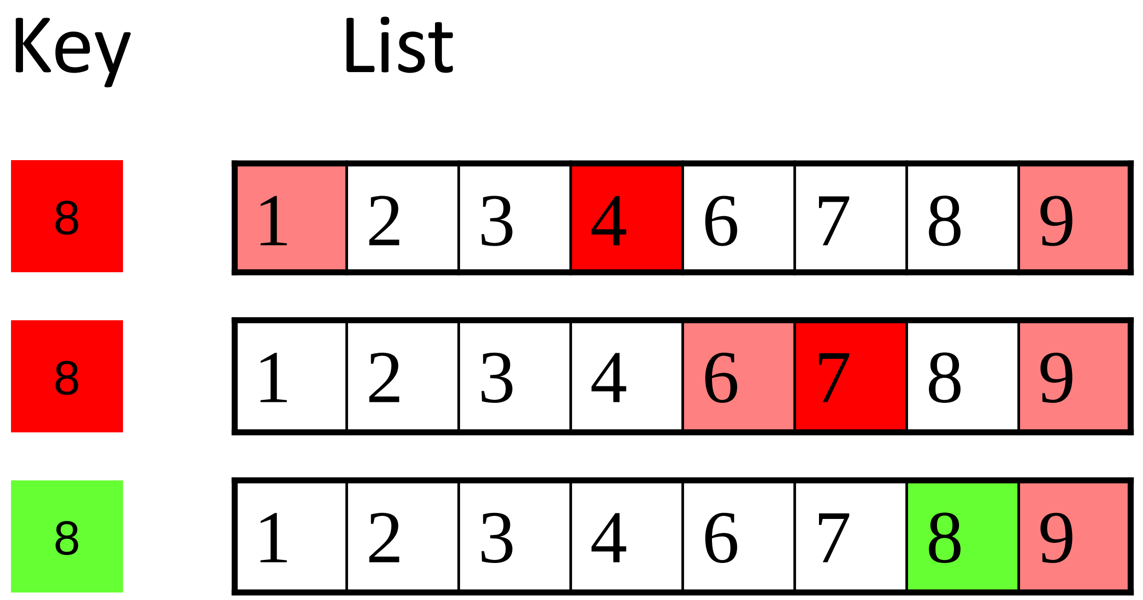
- For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.
  - e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79
- The binary search first compares the key with the element in the middle of the array.

Binary Search, cont.

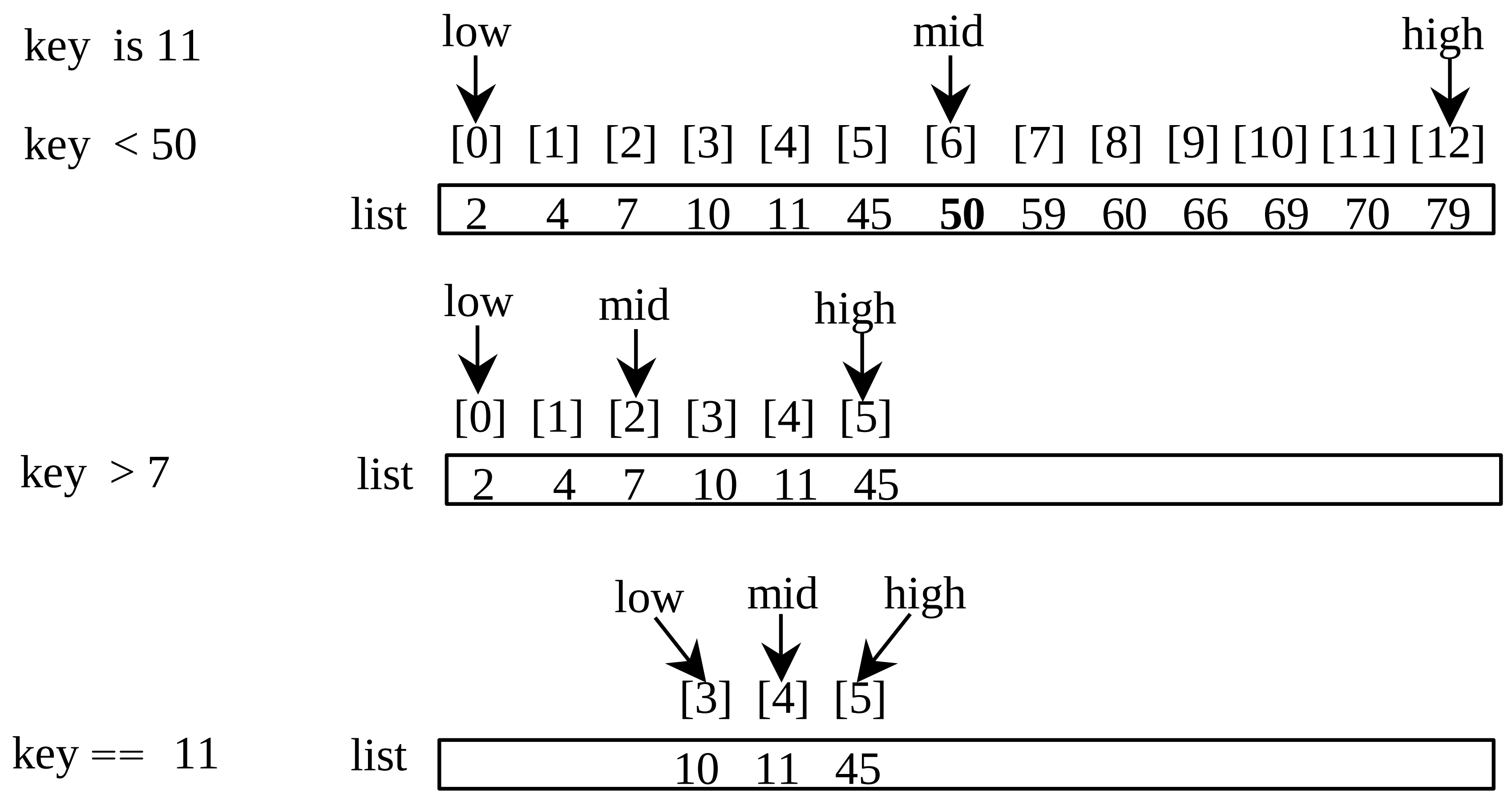
Consider the following three cases:

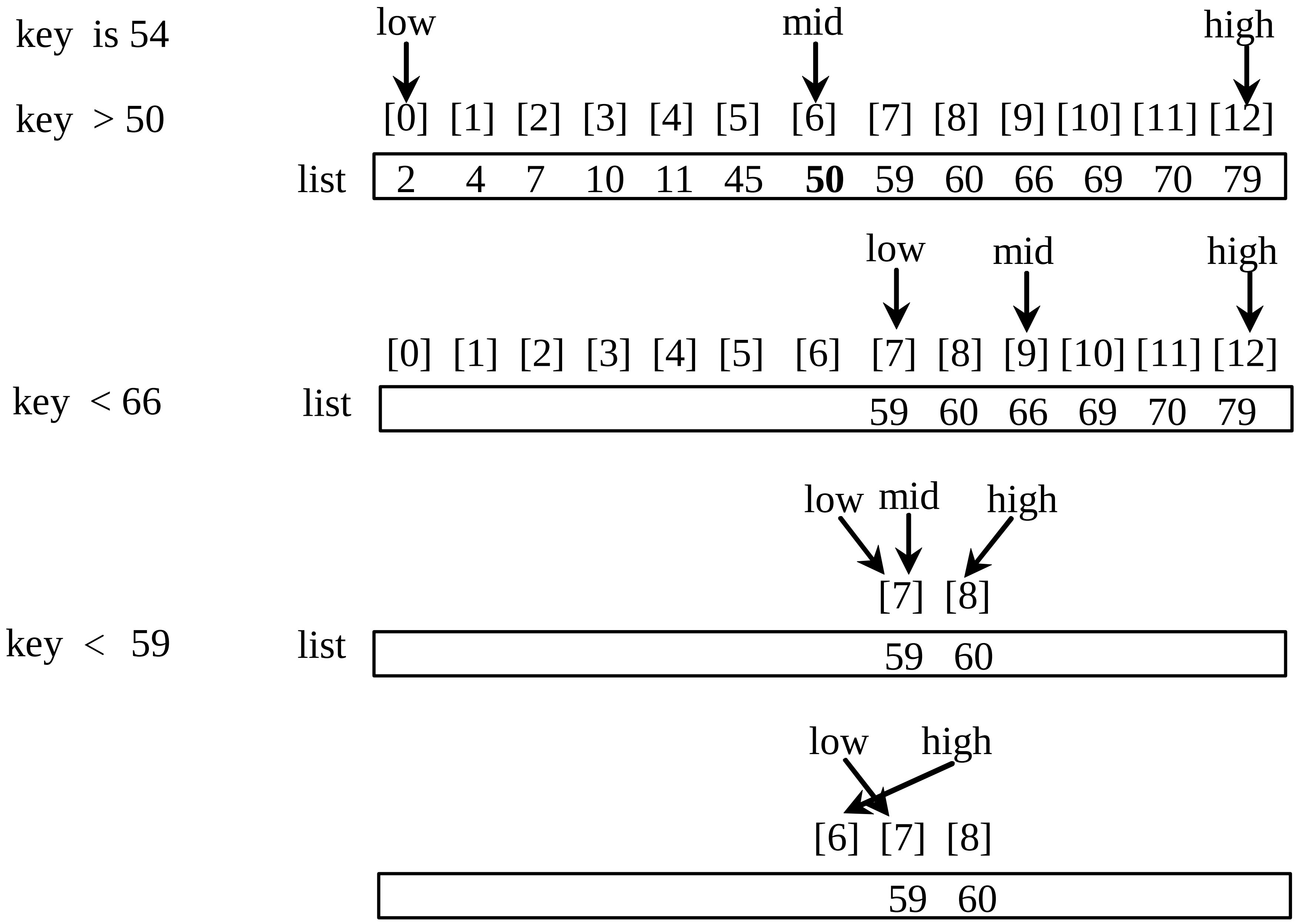
- If the **key** is less than the middle element, you only need to search the key in the first half of the array.
- If the **key** is equal to the middle element, the search ends with a match.
- If the **key** is greater than the middle element, you only need to search the key in the second half of the array.

Binary Search



Binary Search, cont.




key is 54
key > 50
low
mid
high
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
list
2 4 7 10 11 45 **50** 59 60 66 69 70 79
low
mid
high
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]
key < 66
list
59 60 66 69 70 79
low
mid
high
[7] [8]
key < 59
list
59 60
low
high
[6] [7] [8]
59 60

Binary Search, cont.

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns

-insertion point - 1.

insertion point = -(return+1)

The insertion point is the point at which the key would be inserted into the list.

Exemplary Binary Search Method

```
public static int binarySearch(int[] list, int key)
{
  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;
  }
  return -1 - low;
}
```

The Arrays.binarySearch Method

- Since binary search is frequently used in programming, Java provides several overloaded **binarySearch** methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code searches the keys in an array of numbers and an array of characters.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66,
 69, 70, 79};
System.out.println("Index is " +
  java.util.Arrays.binarySearch(list, 11));
```

Return is 4

The Arrays.binarySearch Method

```
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +
java.util.Arrays.binarySearch(chars, 't'));
```

For the binarySearch method to work, the array must be **pre-sorted** in increasing order.

Return is –4

(insertion point is 3, so return is -3-1)

Lab: StudentScore.java

LECTURE 1

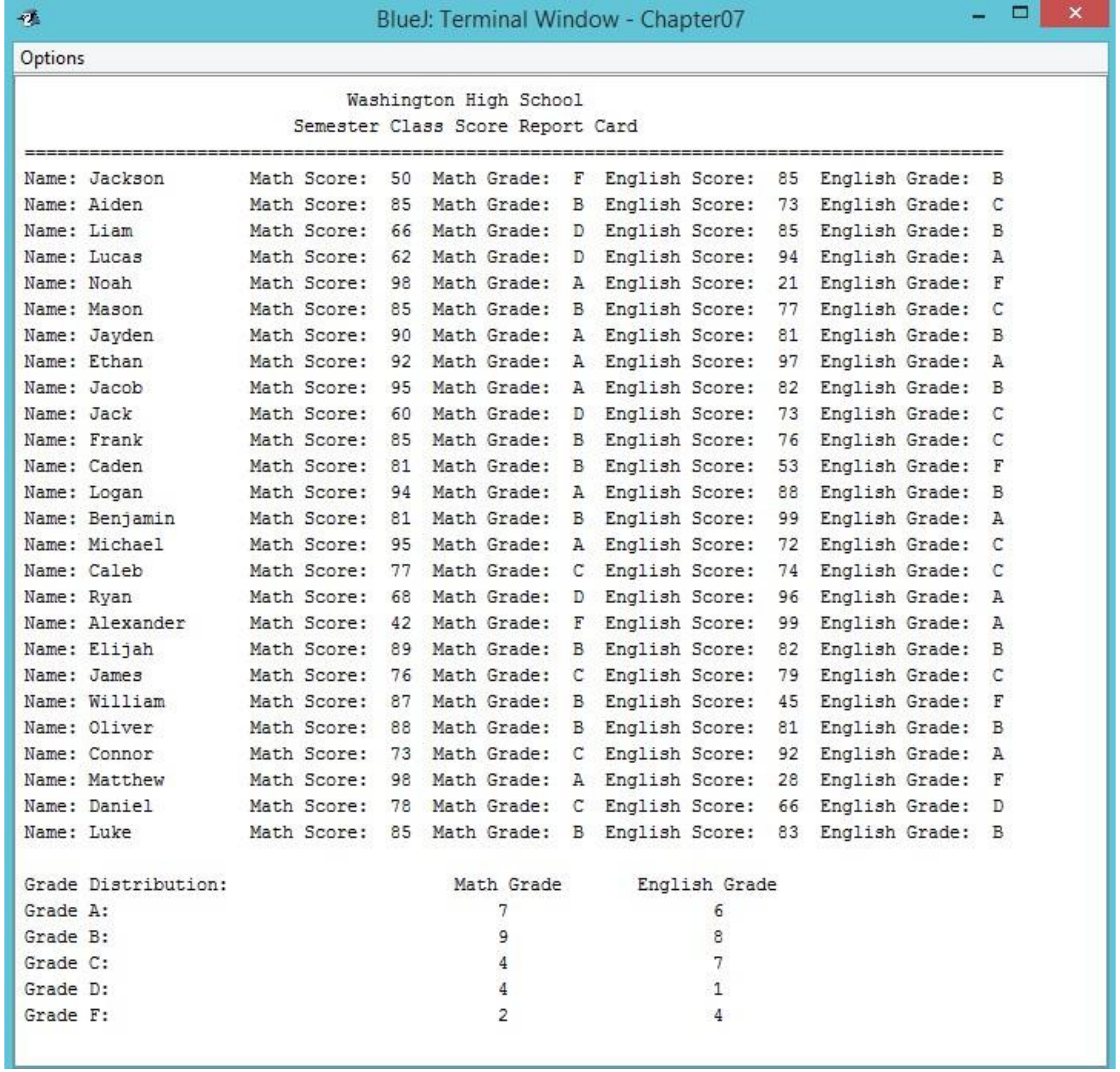
Lab: StudentScore.java

- Write a program to read in student score information from score.txt. In the score.txt file, each line contains student name, student math score, and student English score.
- At the beginning, print out a header message for ***Washington High School semester class score report***.

Lab: StudentScore.java

- Then, read in the student name and score information line by line. In each line, after reading in a single student information, find their corresponding grade for math and English. Then print ***out name, math score, math grade, English score and English grade***.
- After all of the student's information has been read in, tally for all student got A grade, B grade, C grade, D grade and F grade for math and English. Then, print out the information.

Expected Results:



```
                              Washington High School
                         Semester Class Score Report Card
===========================================================================================
Name: Jackson       Math Score:  50  Math Grade:  F  English Score:  85  English Grade:  B
Name: Aiden         Math Score:  85  Math Grade:  B  English Score:  73  English Grade:  C
Name: Liam          Math Score:  66  Math Grade:  D  English Score:  85  English Grade:  B
Name: Lucas         Math Score:  62  Math Grade:  D  English Score:  94  English Grade:  A
Name: Noah          Math Score:  98  Math Grade:  A  English Score:  21  English Grade:  F
Name: Mason         Math Score:  85  Math Grade:  B  English Score:  77  English Grade:  C
Name: Jayden        Math Score:  90  Math Grade:  A  English Score:  81  English Grade:  B
Name: Ethan         Math Score:  92  Math Grade:  A  English Score:  97  English Grade:  A
Name: Jacob         Math Score:  95  Math Grade:  A  English Score:  82  English Grade:  B
Name: Jack          Math Score:  60  Math Grade:  D  English Score:  73  English Grade:  C
Name: Frank         Math Score:  85  Math Grade:  B  English Score:  76  English Grade:  C
Name: Caden         Math Score:  81  Math Grade:  B  English Score:  53  English Grade:  F
Name: Logan         Math Score:  94  Math Grade:  A  English Score:  88  English Grade:  B
Name: Benjamin      Math Score:  81  Math Grade:  B  English Score:  99  English Grade:  A
Name: Michael       Math Score:  95  Math Grade:  A  English Score:  72  English Grade:  C
Name: Caleb         Math Score:  77  Math Grade:  C  English Score:  74  English Grade:  C
Name: Ryan          Math Score:  68  Math Grade:  D  English Score:  96  English Grade:  A
Name: Alexander     Math Score:  42  Math Grade:  F  English Score:  99  English Grade:  A
Name: Elijah        Math Score:  89  Math Grade:  B  English Score:  82  English Grade:  B
Name: James         Math Score:  76  Math Grade:  C  English Score:  79  English Grade:  C
Name: William       Math Score:  87  Math Grade:  B  English Score:  45  English Grade:  F
Name: Oliver        Math Score:  88  Math Grade:  B  English Score:  81  English Grade:  B
Name: Connor        Math Score:  73  Math Grade:  C  English Score:  92  English Grade:  A
Name: Matthew       Math Score:  98  Math Grade:  A  English Score:  28  English Grade:  F
Name: Daniel        Math Score:  78  Math Grade:  C  English Score:  66  English Grade:  D
Name: Luke          Math Score:  85  Math Grade:  B  English Score:  83  English Grade:  B

Grade Distribution:                   Math Grade       English Grade
Grade A:                                   7                  6
Grade B:                                   9                  8
Grade C:                                   4                  7
Grade D:                                   4                  1
Grade F:                                   2                  4
```

Student Score text file (score.txt)

name, Math score, English score

```
Jackson    50 85
Aiden      85 73
Liam       66 85
Lucas      62 94
Noah       98 21
Mason      85 77
Jayden     90 81
Ethan      92 97
Jacob      95 82
Jack       60 73
Frank      85 76
Caden      81 53
Logan      94 88
Benjamin   81 99
Michael    95 72
Caleb      77 74
Ryan       68 96
Alexander  42 99
Elijah     89 82
James      76 79
William    87 45
Oliver     88 81
Connor     73 92
Matthew    98 28
Daniel     78 66
Luke       85 83
```

Copy the getGrade() method from StudentGPAMethod.java

```
public static char getGrade(int score){
     char grade = ' ';
       if (score>=90 && score<=100){grade = 'A';}
       else if (score>=80 && score<=100)  {grade = 'B';}
       else if (score>=70 && score<=100)  {grade = 'C';}
       else if (score>=60 && score<=100)  {grade = 'D';}
       else if (score<60 && score>= 0){grade = 'F';}
       else { grade = 'N';}
     return grade;
   }
```

Example data definition:

No Pseudo Code given in this project.

```
int lines = 0;
String[] names    = new String[lines];
int[] mathScore   = new int[lines];
int[] engScore    = new int[lines];
char[] mathGrade  = new char[lines];
char[] engGrade   = new char[lines];

input = new Scanner(ifile);
```

```
   String grades = "ABCDF";
   int[] gcountMath  = new int[5];
   int[] gcountEng   = new int[5];
// use similar method in
// LetterCount.java
```

Lab

STUDENT SCORE REPORT