ARRAYS

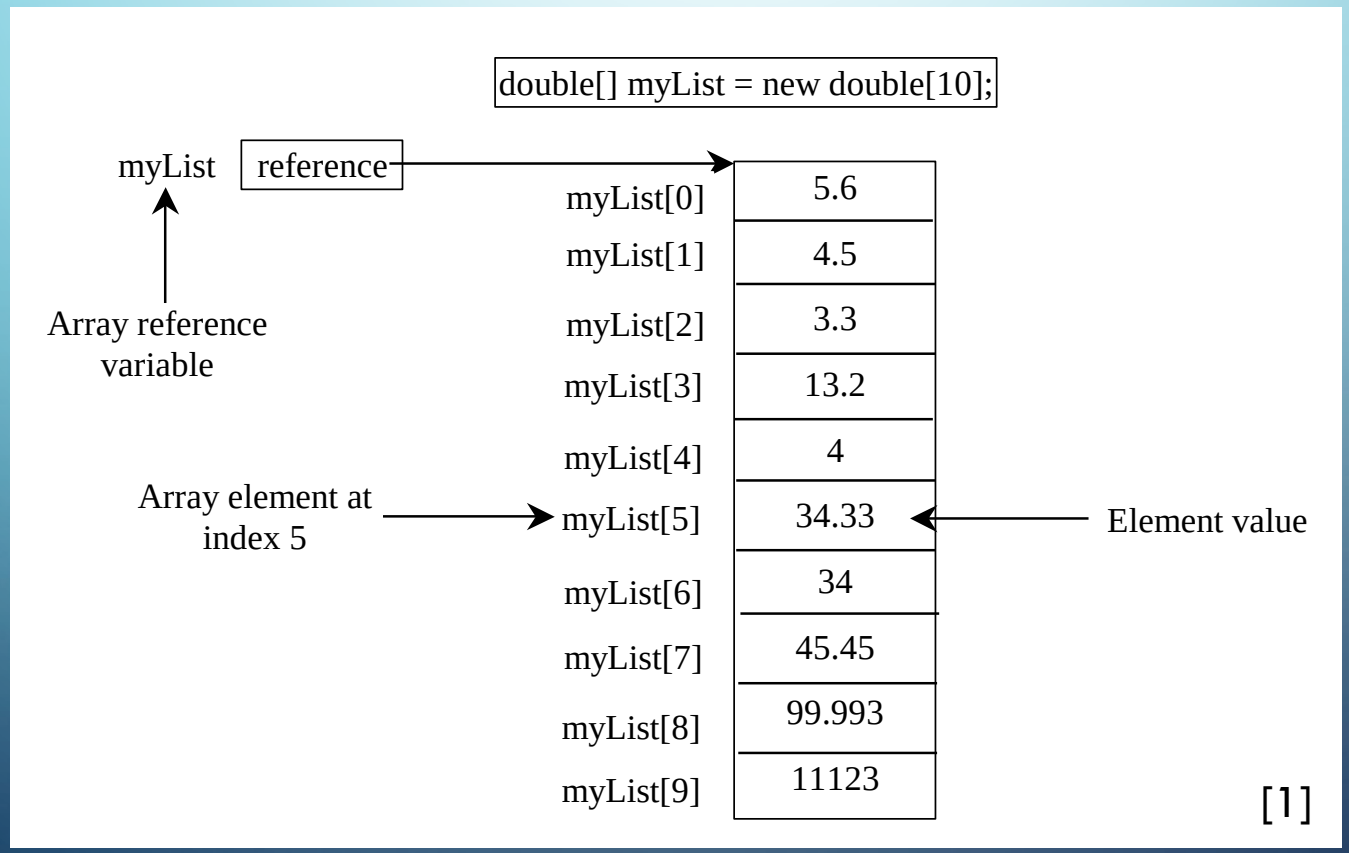
ITX 2001, CSX 3002, IT 2371

ARRAYS

- To describe why an array is necessary in programming
- To learn the steps involved in using arrays: declaring array reference variables and creating arrays
- To initialize the values in an array
- To copy contents from one array to another
- To develop and invoke methods with array arguments and return type

ARRAY

It represents a collection of the same types of data.



[1]

ARRAY DECLARATION

- `datatype[] arrayRefVar;`

  Example:

  ```
  double[] myList;
  ```

- `datatype arrayRefVar[];` // This style is allowed, but not preferred

  Example:

  ```
  double myList[];
  ```

ARRAY DECLARING AND CREATING IN ONE STEP

- ```
  datatype[] arrayRefVar = new
      datatype[arraySize];
  ```

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

- ```
  datatype arrayRefVar[] = new
    datatype[arraySize];
  ```

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

ARRAY LENGTH

- Once an array is created, its size is fixed.
- It cannot be changed.
- You can find its size using

  arrayRefVar.length
- For example,

  myList.length returns 10

DEFAULT VALUES

When an array is created, its elements are assigned the default value of

- 0 for the numeric primitive data types,
- '\u0000' for char types, and
- false for Boolean types.

INDEXED VARIABLES

- The array elements are accessed through the index.
- The array indices are *0-based*,
  - i.e., it starts from 0 to arrayRefVar.length-1.
- Each element in the array is represented using the following syntax, known as an *indexed variable*:

  arrayRefVar[index];

USING INDEXED VARIABLES

- After an array is created, an indexed variable can be used in the same way as a regular variable.

- For example, the following code adds the value in myList[0] and myList[1] to myList[2].

```
myList[2] = myList[0] + myList[1];
```

ARRAY INITIALIZERS

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement !

DECLARING, CREATING, INITIALIZING

- Using the shorthand notation

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

ARRAY INITIALIZERS

- Using the shorthand notation, you have to declare, create, and initialize the array all in one statement.
- Splitting it would cause a syntax error.
- For example, the following is wrong:

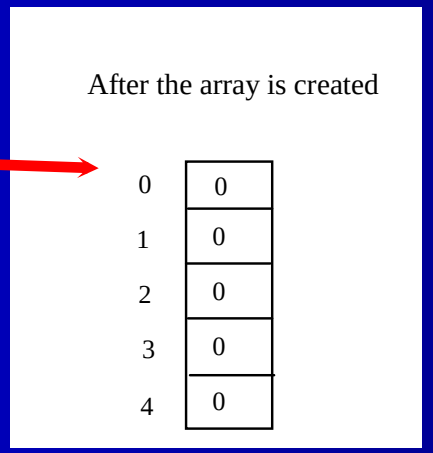
```
double[] myList;

myList = {1.9, 2.9, 3.4, 3.5};
```

animation

Trace Program with Arrays

Declare array variable values, create an array, and assign its reference to values

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = values[i] + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



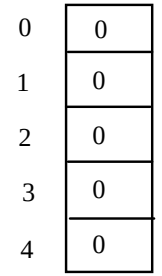
animation

Trace Program with Arrays

i becomes 1

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = values[i] + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



animation

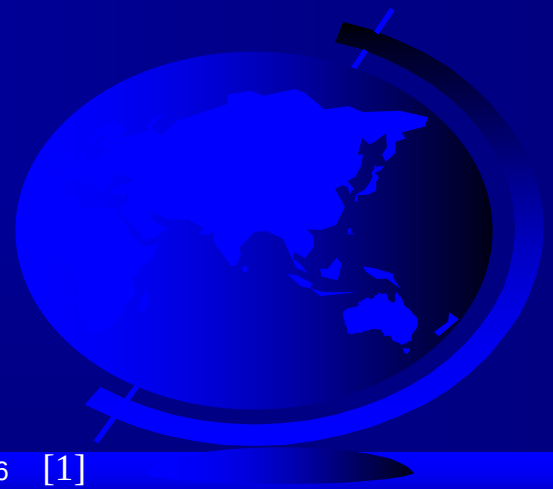
Trace Program with Arrays

i (=1) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = values[i] + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

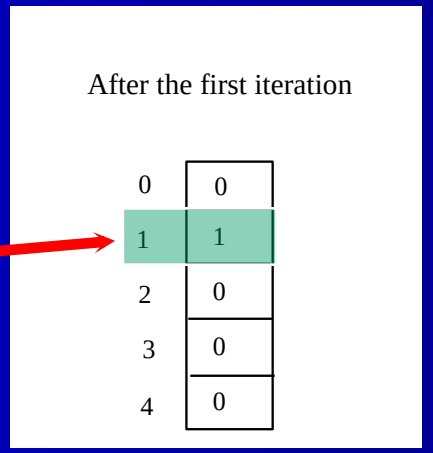


animation

Trace Program with Arrays

After this line is executed, value[1] is 1

```
public class Test {
  public static void main(Strin  gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



animation

Trace Program with Arrays

After i++, i becomes 2

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = values[i] + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

animation

Trace Program with Arrays

i (= 2) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = values[i] + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

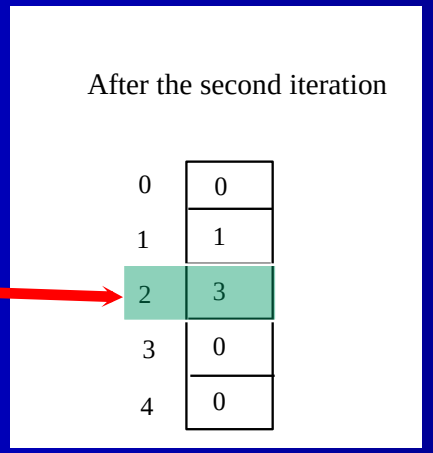
0	0
1	1
2	0
3	0
4	0

animation

Trace Program with Arrays

After this line is executed, values[2] is 3 (2 + 1)

```
public class Test {
  public static void main(Strin    gs) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



animation

Trace Program with Arrays

After this, i becomes 3.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

animation

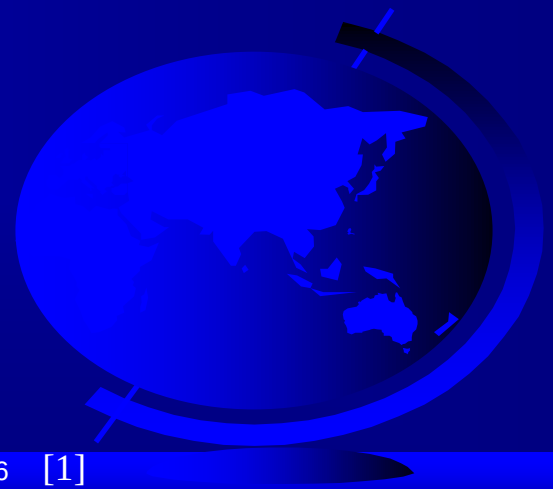
Trace Program with Arrays

i (=3) is still less than 5.

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

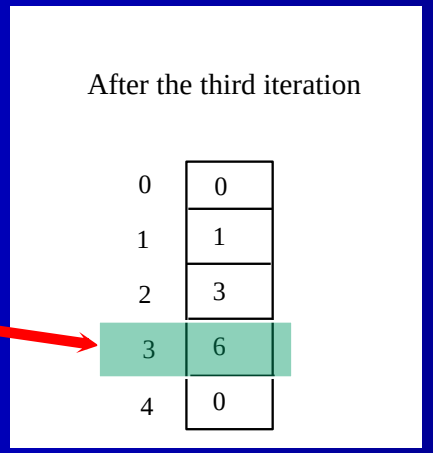


animation

Trace Program with Arrays

After this line, values[3] becomes 6 (3 + 3)

```
public class Test {
  public static void main(String[] arg
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



animation

Trace Program with Arrays

After this, i becomes 4

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

animation

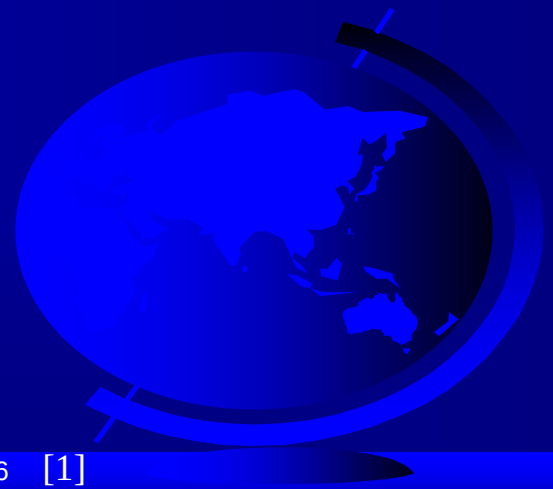
Trace Program with Arrays

i (=4) is still less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

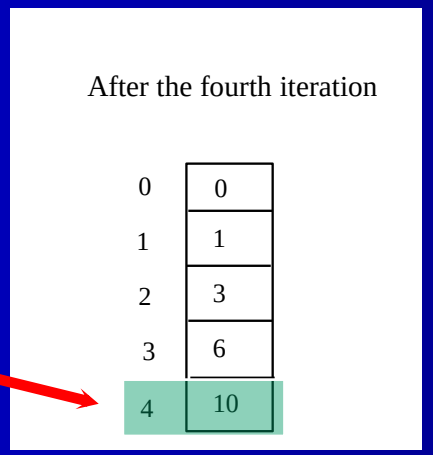


animation

Trace Program with Arrays

After this, values[4] becomes 10 (4 + 6)

```
public class Test {
  public static void main(Stri    args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
     values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



animation

Trace Program with Arrays

After i++, i becomes 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

i (=5) < 5 is false. Exit the loop

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the fourth iteration

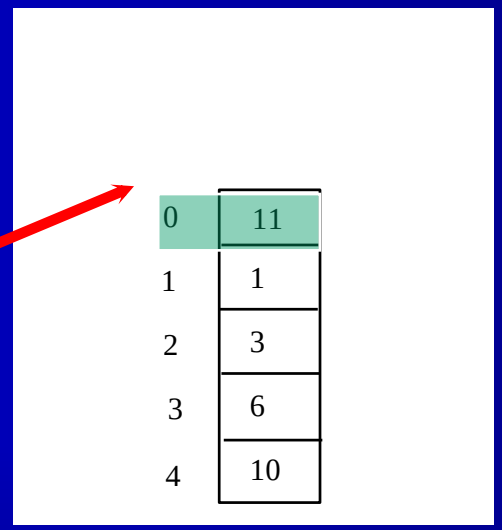
0	0
1	1
2	3
3	6
4	10

animation

Trace Program with Arrays

After this line, values[0] is 11 (1 + 10)

```
public class Test {
  public static void m      tring[] args) {
    int[] values = ne      [5];
    for (int i = 1; i      ; i++) {
      values[i] = i      values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```



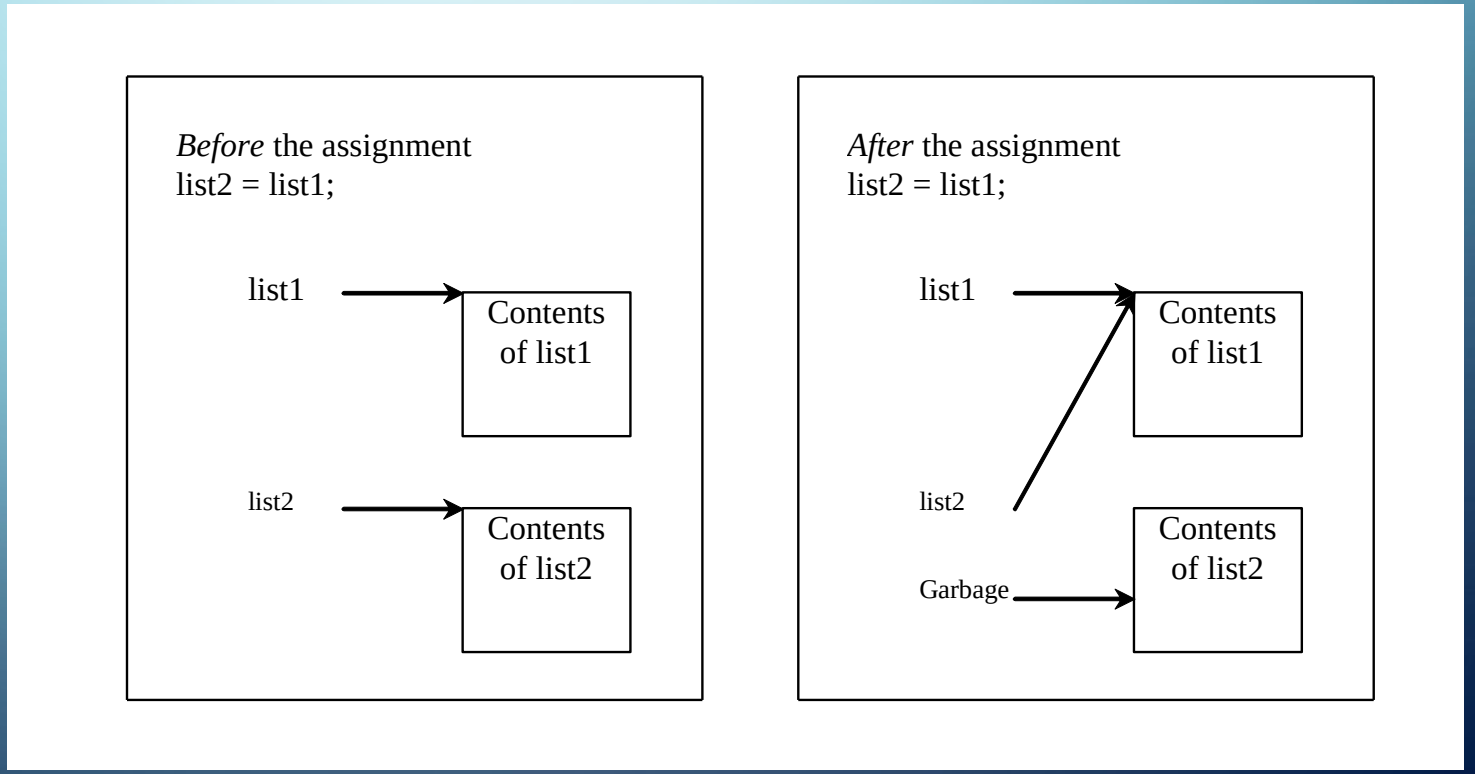
PROCESSING ARRAYS

1. Initializing arrays
2. Printing arrays
3. Summing all elements
4. Finding the largest element
5. Finding the smallest index of the largest element

COPYING ARRAYS

- To duplicate an array or a part of an array.
- Use the assignment statement (=), as follows:

```
list2 = list1;
```



COPYING ARRAYS

1. 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];
```

COPYING ARRAYS

2. Using `arraycopy` Utility :

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

Example:

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

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

PASSING ARRAYS TO METHODS BY REFERENCE

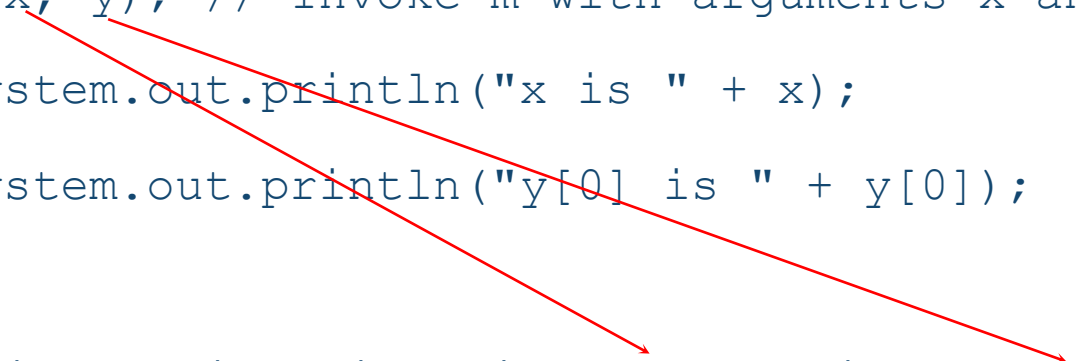
Java uses *pass by value* to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

- 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.

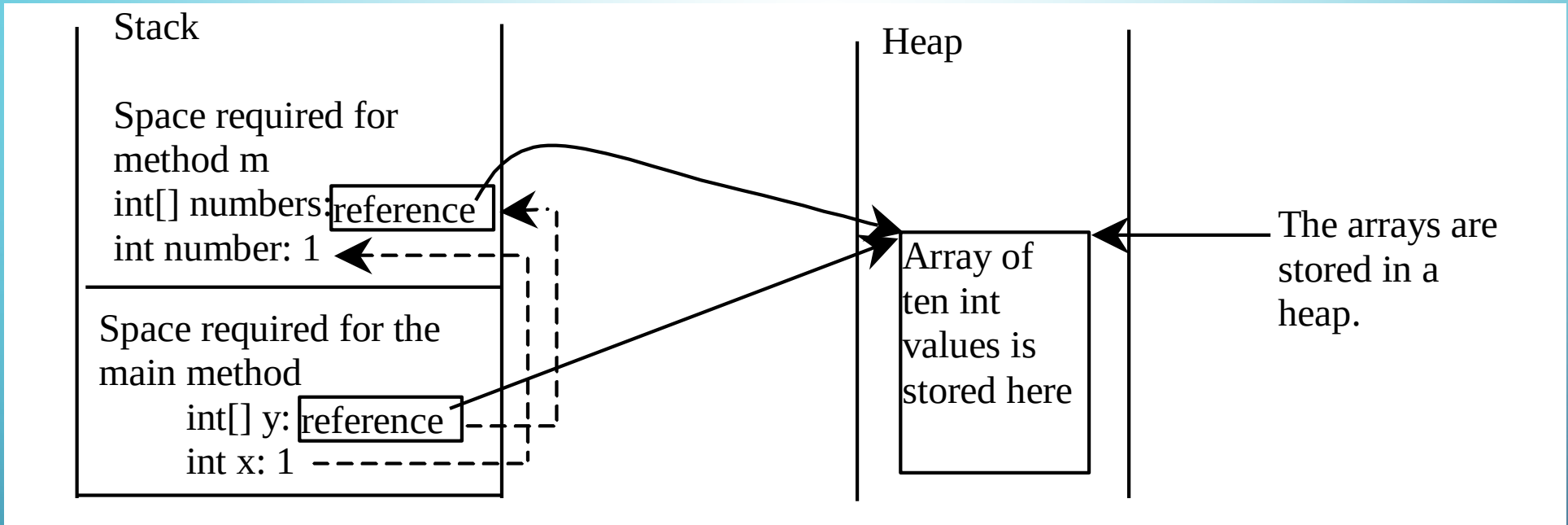
- For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

PASSING ARRAYS TO METHODS BY REFERENCE: 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]
  }
}
```

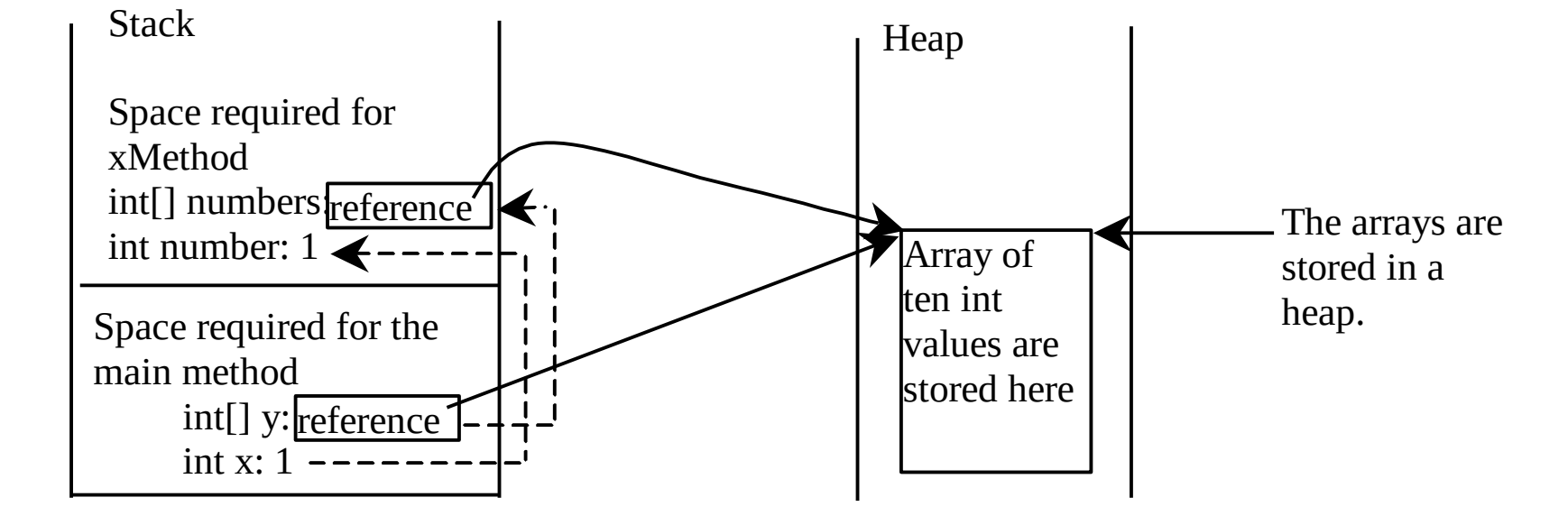


PASSING ARRAYS TO METHODS BY REFERENCE: STACK



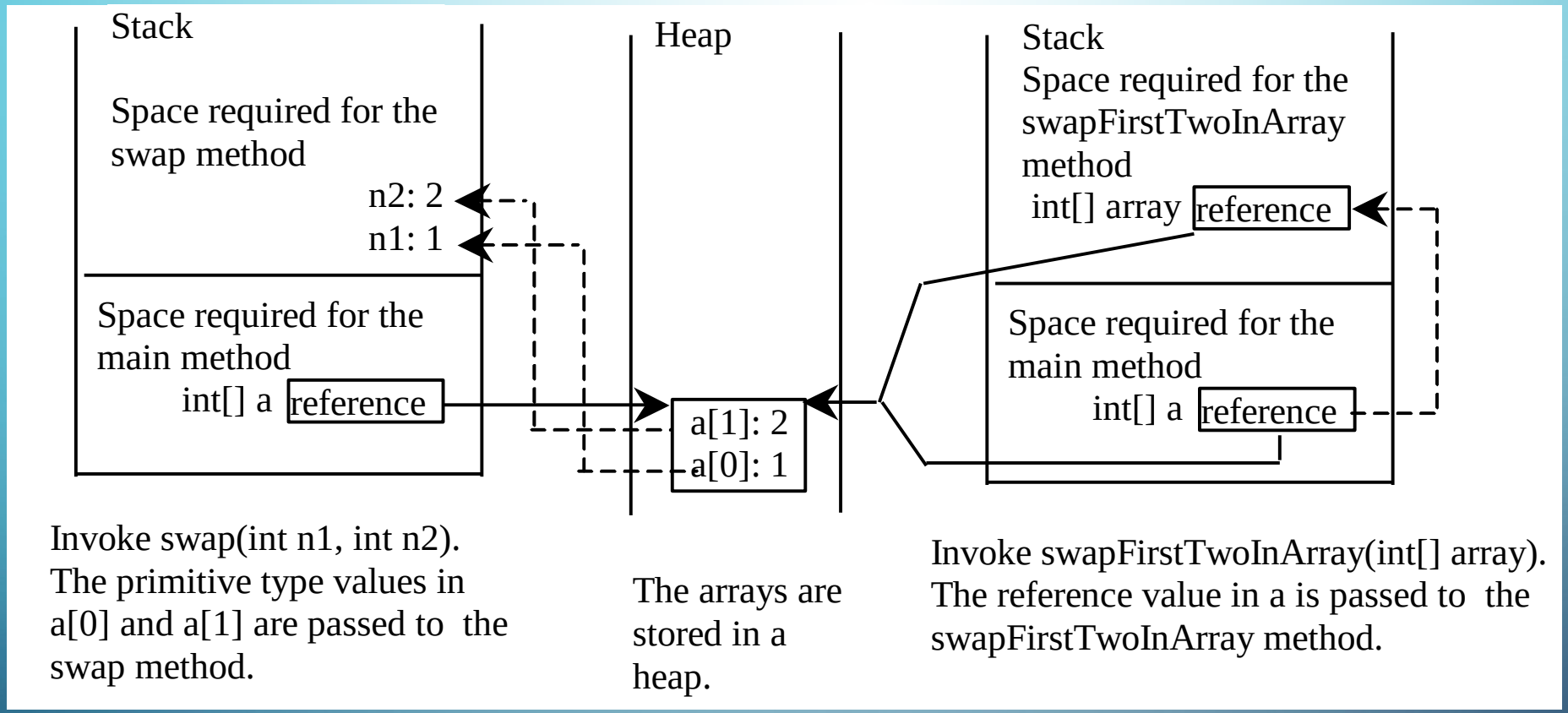
When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

PASSING ARRAYS TO METHODS BY REFERENCE: HEAP



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 TO METHODS BY REFERENCE: HEAP



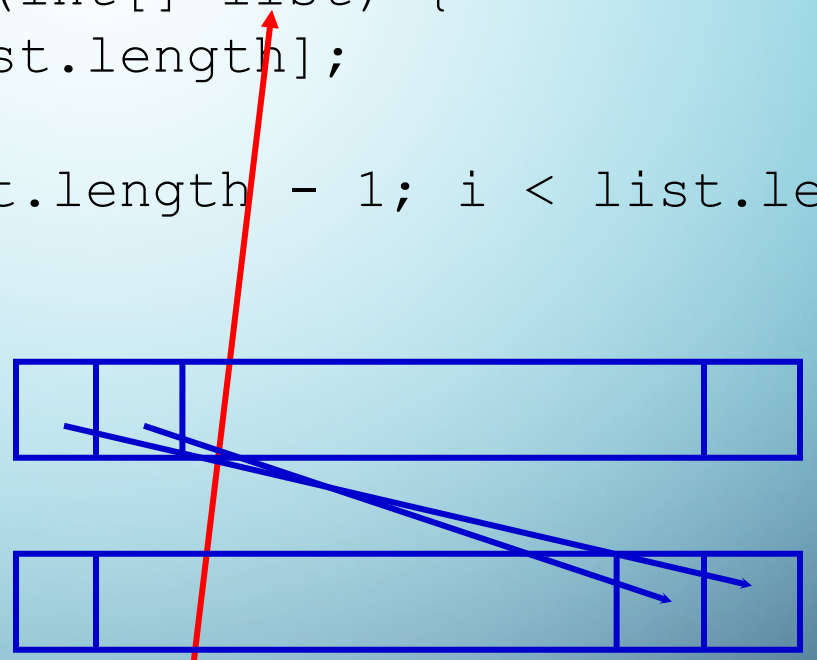
Invoke swap(int n1, int n2). The primitive type values in a[0] and a[1] are passed to the swap method.

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

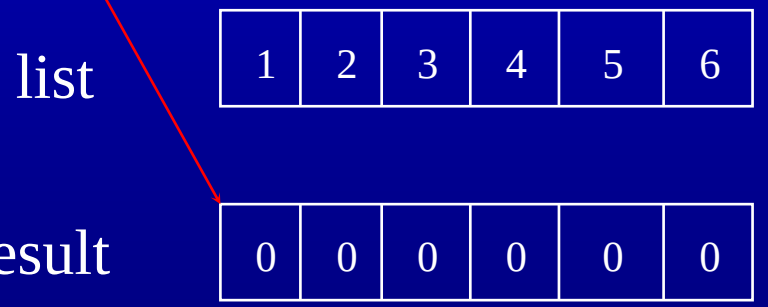
animation

Trace the reverse Method

```
int[] list1 = new int[]{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;
}
```



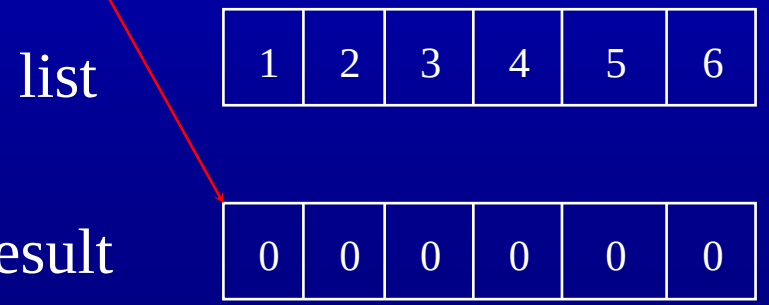
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```



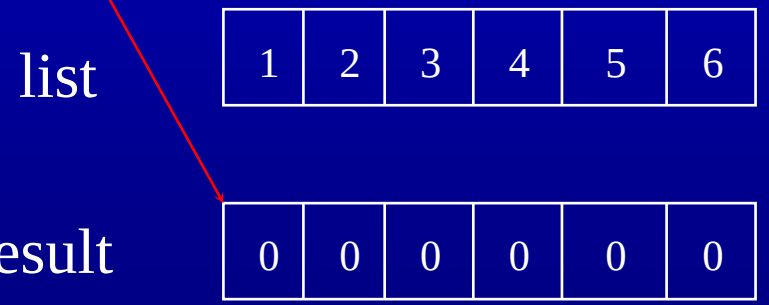
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```



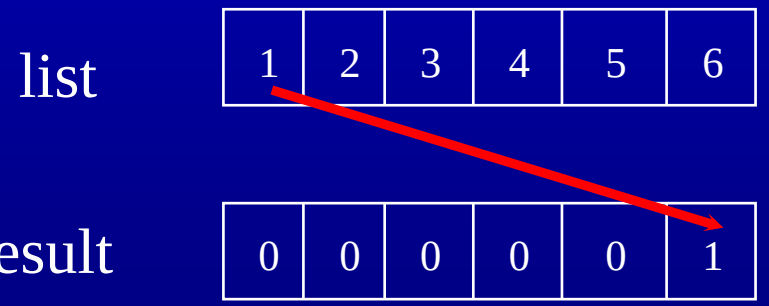
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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 1 and j becomes 4

list	1	2	3	4	5	6

result	0	0	0	0	0	1

animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	0	0	0	0	1

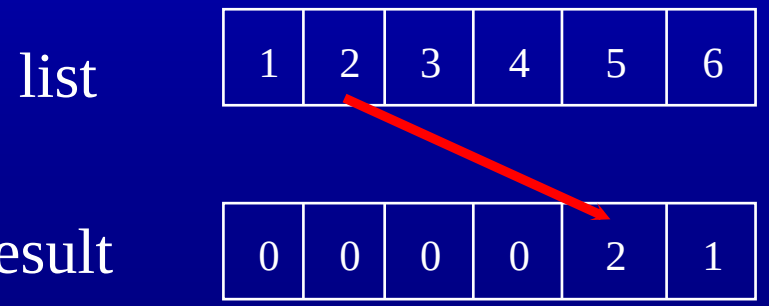
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	0	0	0	2	1

animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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) is still less than 6

list	1	2	3	4	5	6

result	0	0	0	0	2	1

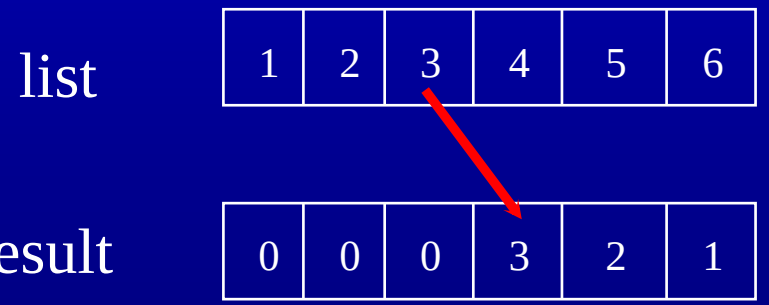
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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 3 and j becomes 2

list	1	2	3	4	5	6

result	0	0	0	3	2	1

animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	0	0	3	2	1

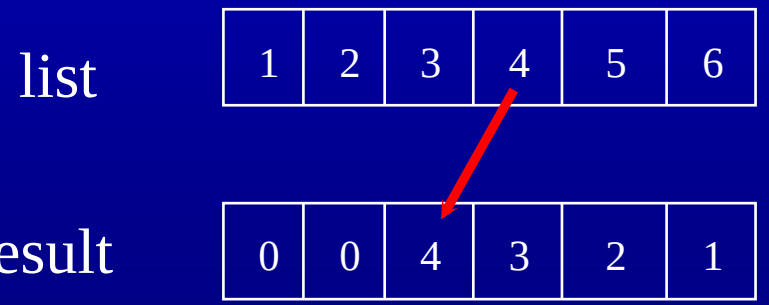
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	0	4	3	2	1

animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	0	4	3	2	1

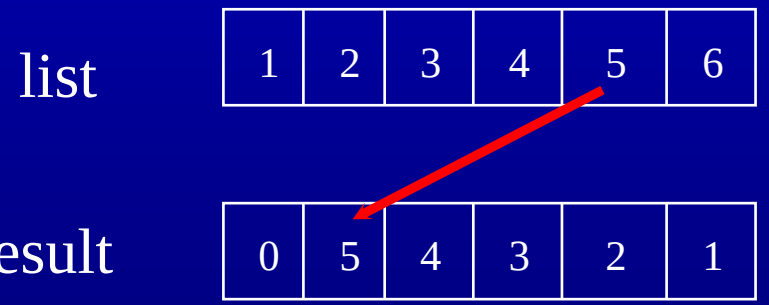
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



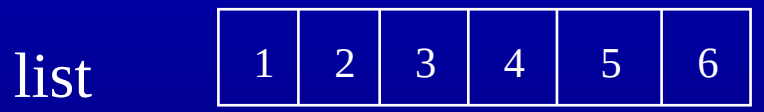
animation

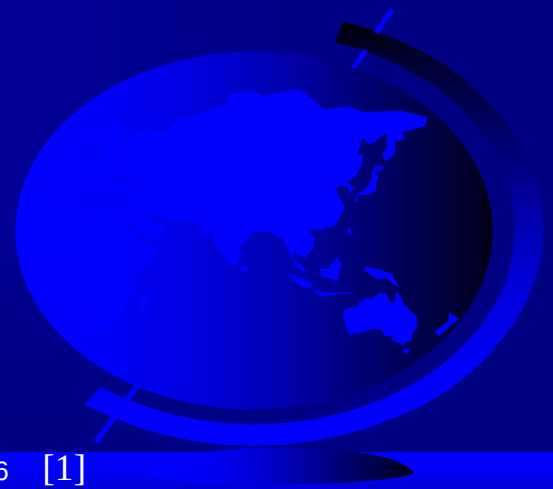
Trace the reverse Method, cont.

```
int[] list1 = new int[]{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





animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	0	5	4	3	2	1

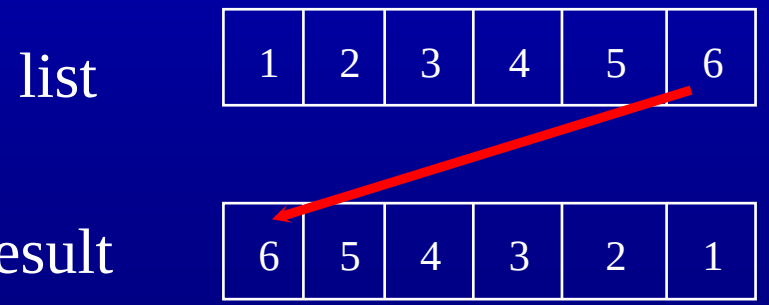
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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]



animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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

list	1	2	3	4	5	6

result	6	5	4	3	2	1

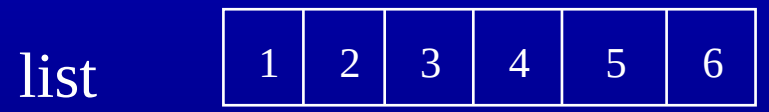
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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 (=6) < 6 is false. So exit the loop.



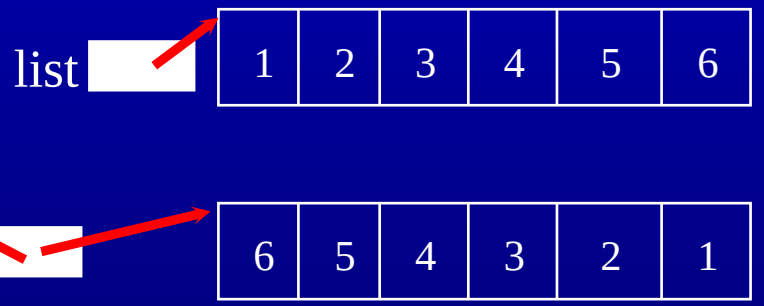
animation

Trace the reverse Method, cont.

```
int[] list1 = new int[]{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;
}
```

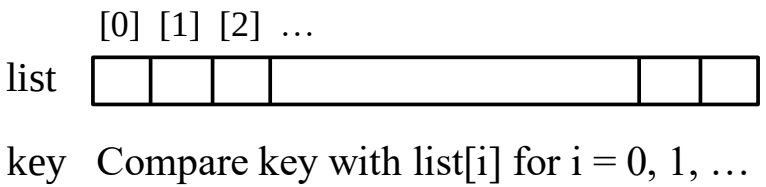
Return result



SEARCHING ARRAY

- 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.
- Two commonly used approaches are ***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.

animation

Linear Search Animation

LINEAR SEARCH ALGORITHM

```
/** 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
```

BINARY SEARCH

- 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.

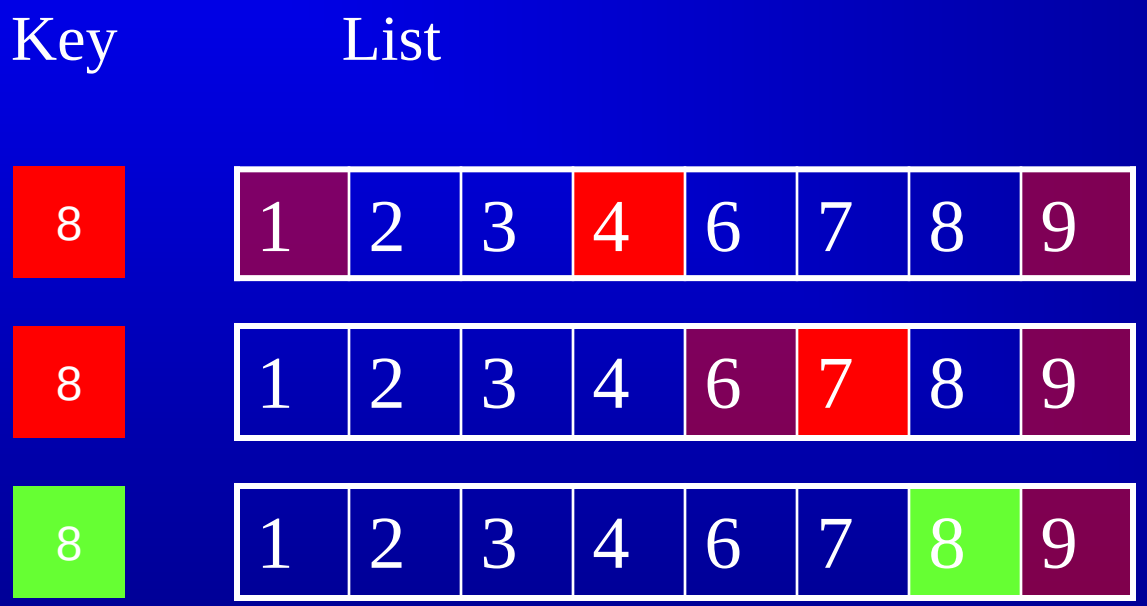
BINARY SEARCH

- Consider the following three cases:

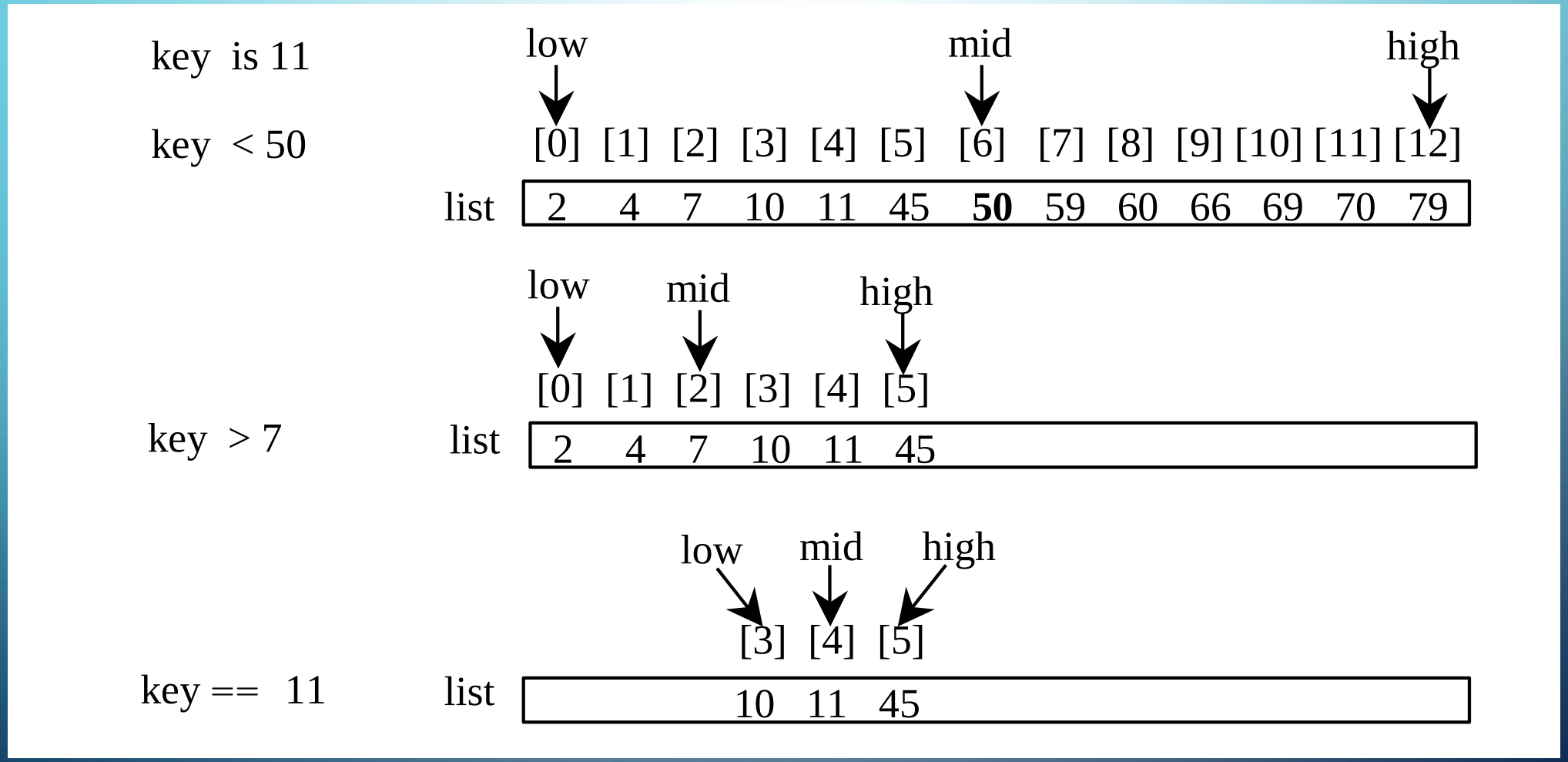
1. If the key is less than the middle element, you only need to search the key in the first half of the array.
2. If the key is equal to the middle element, the search ends with a match.
3. If the key is greater than the middle element, you only need to search the key in the second half of the array.

animation

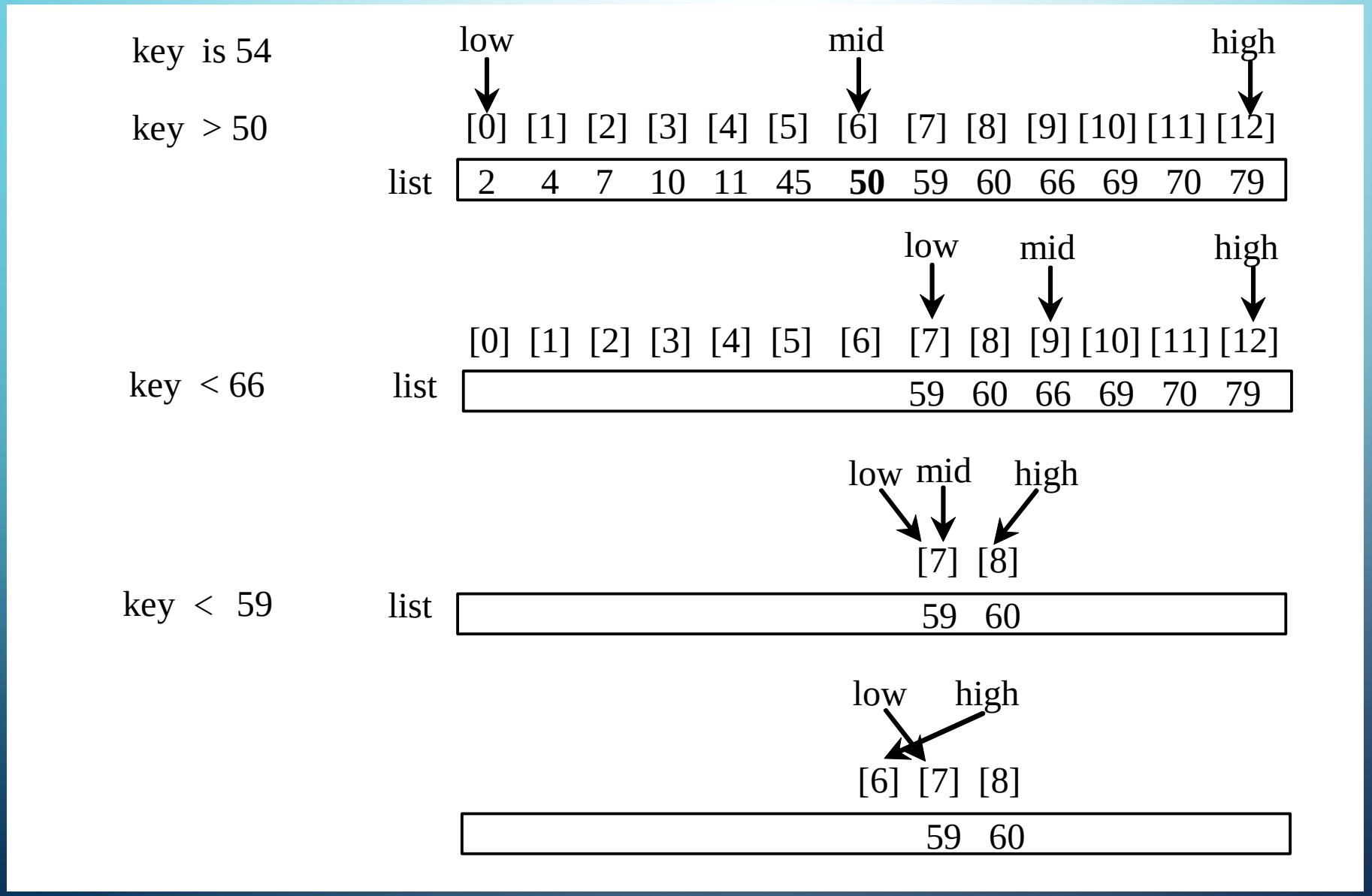
Binary Search



BINARY SEARCH



BINARY SEARCH



BINARY SEARCH

- The binarySearch method returns the index of the search key if it is contained in the list.
- Otherwise, it returns –insertion point - 1.
  - The insertion point is the point at which the key would be inserted into the list.

BINARY SEARCH ALGORITHM

```
/** Use binary search to find the key in the list */
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;
}
```

[1]

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

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

[1]

THE ARRAYS.SORT METHOD

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the **java.util.Arrays** class.

For example, the following code sorts an array of numbers and an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

TWO-DIMENSIONAL ARRAYS

```
// Declare array ref var
dataType[][] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

TWO-DIMENSIONAL DECLARATION AND CREATION

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

or

```
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);


double[][] x;
```

TWO-DIMENSIONAL DECLARING, CREATING AND INITIALIZING USING SHORTHAND NOTATION

You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

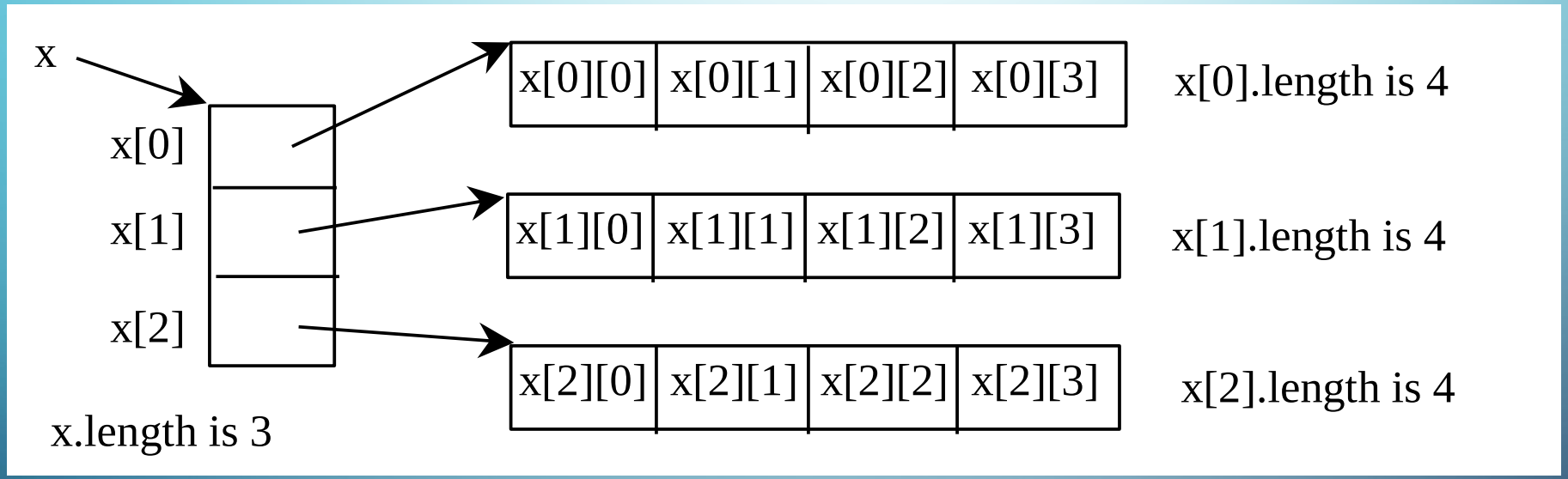
```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

LENGTHS OF TWO-DIMENSIONAL ARRAYS

```
int[][] x = new int[3][4];
```



LENGTHS OF TWO-DIMENSIONAL ARRAYS (CONT.)

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

array.length

array[0].length

array[1].length

array[2].length

array[3].length

array[4].length ArrayIndexOutOfBoundsException

MULTIDIMENSIONAL ARRAYS

- In Java, you can create n-dimensional arrays for any integer n and represent it as n-dimensional data structures.
- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3.

- For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

```
double[][][] scores = new double[10][5][2];
```

REFERENCES

[1] Liang, “Introduction to Java Programming”, 6th Edition, Pearson Education, Inc.