# Chapter 7 Single-Dimensional Arrays

# Opening Problem

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

# Objectives

To describe why arrays are necessary in programming (§7.1).

To declare array reference variables and create arrays (§§7.2.1–7.2.2).

To obtain array size using **arrayRefVar.length** and know default values in an array (§7.2.3).

To access array elements using indexes (§7.2.4).

To declare, create, and initialize an array using an array initializer (§7.2.5).

To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements, random shuffling, and shifting elements) (§7.2.6).

To simplify programming using the foreach loops (§7.2.7).

To apply arrays in application development (**AnalyzeNumbers**, **DeckOfCards**) (§§7.3–7.4).

To copy contents from one array to another (§7.5).

To develop and invoke methods with array arguments and return values (§§7.6–7.8).

To define a method with a variable-length argument list (§7.9).

To search elements using the linear (§7.10.1) or binary (§7.10.2) search algorithm.

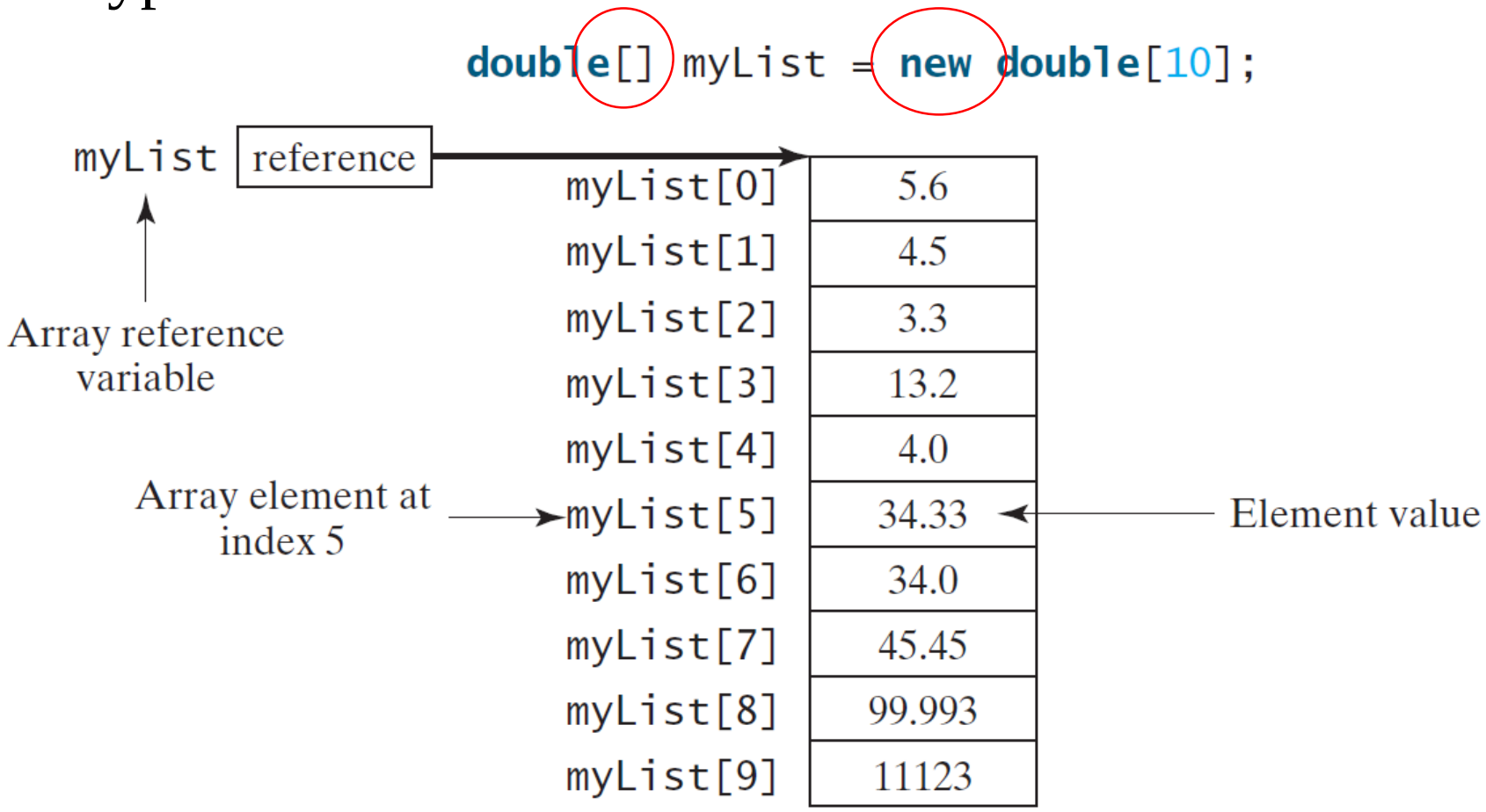
To sort an array using the selection sort approach (§7.11).

To use the methods in the **java.util.Arrays** class (§7.12).

To pass arguments to the main method from the command line (§7.13).

# Introducing Arrays

Array is a data structure that represents a collection of the same types of data.



# Declaring Array Variables

- `datatype[] arrayRefVar;`

  Example:

  **`double[] myList;`**

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

  Example:

  `double myList[];`

# Creating Arrays

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

Example:

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

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the 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];
```

# The Length of an Array

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. In the example in Figure 6.1, myList holds ten double values and the indices are from 0 to 9.

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;
```

# CAUTION

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};
```

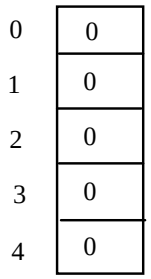
animation

# Trace Program with Arrays

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

```
public class Test {
  public static void main(Str[] 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 array is created



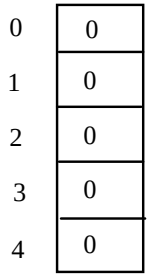
animation

# 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] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



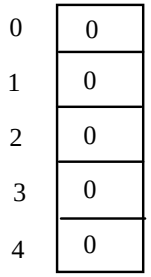

animation

# 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] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created



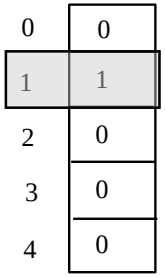
animation

# 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];
  }
}
```

After the first iteration



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] = 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

```
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];
  }
}
```

i (= 2) is less than 5

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(Stri      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];
  }
}
```

After the second iteration

| | |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 3 |
| 3 | 0 |
| 4 | 0 |

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 |

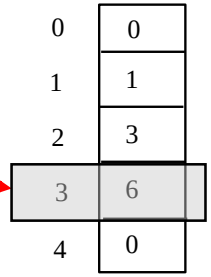
animation

# 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];
  }
}
```

After the third iteration



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 |

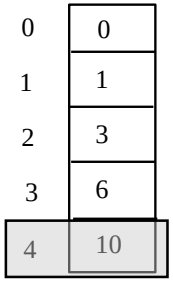
animation

# 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];
  }
}
```

After the fourth iteration



animation

# Trace Program with Arrays

After i++, i becomes 5

```
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];
  }
}
```

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];
  }
}
```

| | |
|---|---|
| 0 | 11 |
| 1 | 1 |
| 2 | 3 |
| 3 | 6 |
| 4 | 10 |

# Processing Arrays

See the examples in the text.

1. (Initializing arrays with input values)
2. (Initializing arrays with random values)
3. (Printing arrays)
4. (Summing all elements)
5. (Finding the largest element)
6. (Finding the smallest index of the largest element)
7. (*Random shuffling*)
8. (*Shifting elements*)

# Initializing arrays with input values

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

# Initializing arrays with random values

```
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
```

# Printing arrays

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

# Summing all elements

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
```

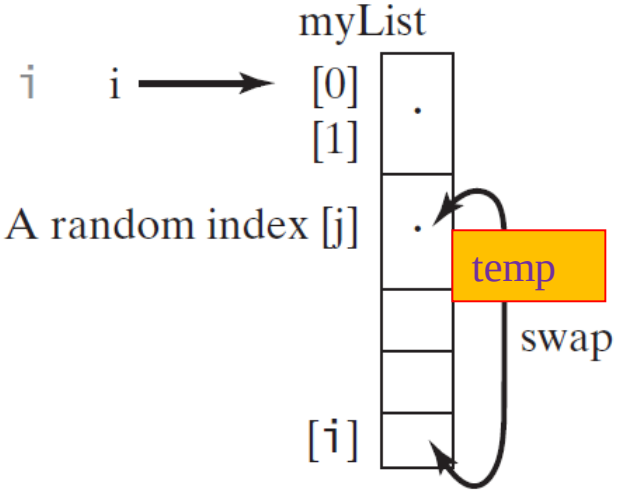
# Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max) max = myList[i];
}
```

# Random shuffling

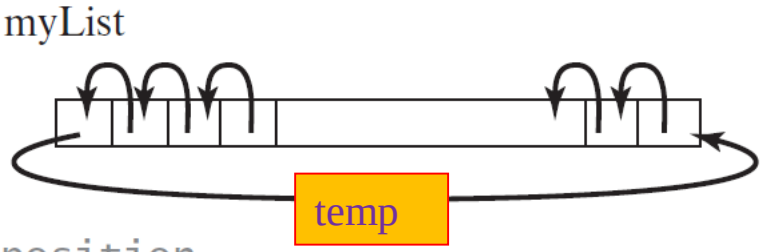
```
for (int i = myList.length - 1; i > 0; i--) {
  // Generate an index j randomly with 0 <= j <= i
  int j = (int)(Math.random()
    * (i + 1));

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```

range decreased



# Shifting Elements

```
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
  myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```




# 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:

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

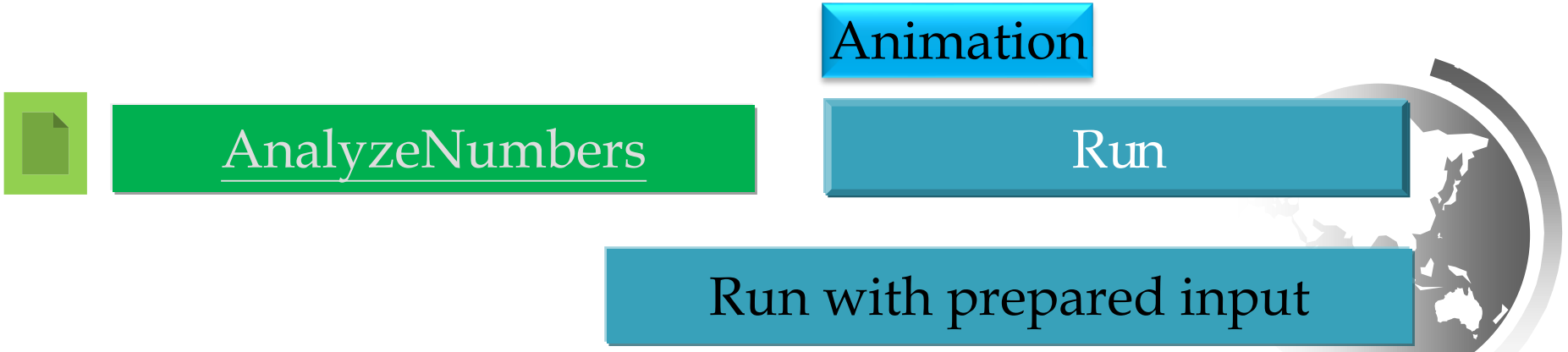
In general, the syntax is

```
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.

# Opening Problem

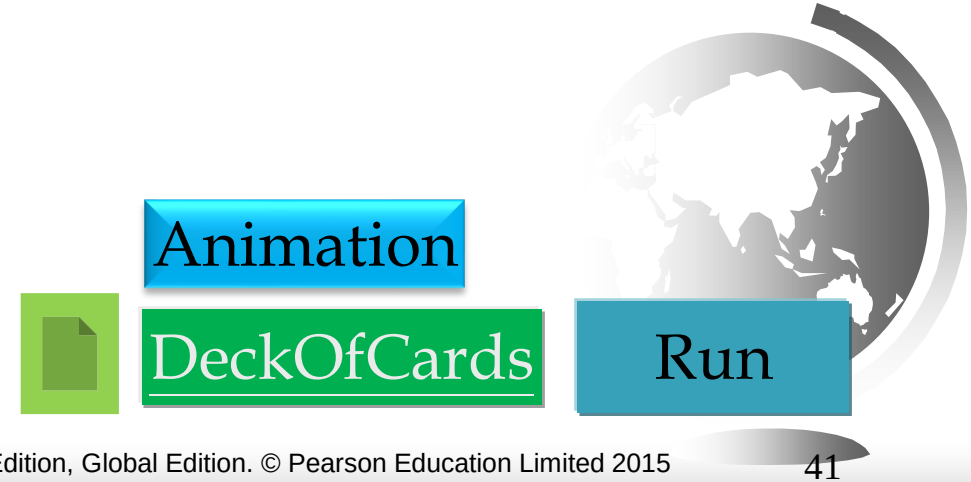
Read one hundred numbers, compute their average, and find out how many numbers are above the average.



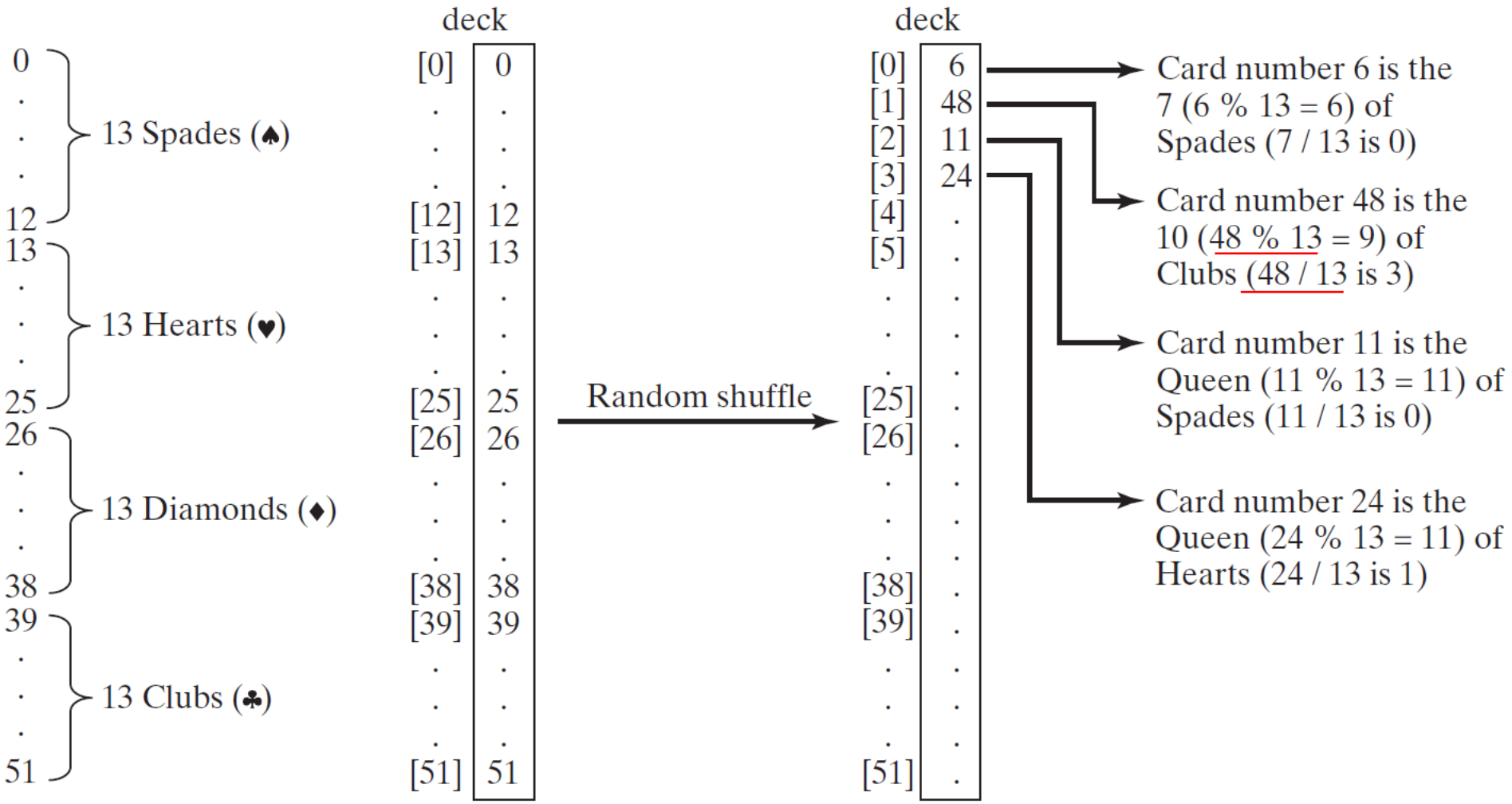
# Problem: Deck of Cards

The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows:

```
int[] deck = new int[52];
// Initialize cards
for (int i = 0; i < deck.length; i++)
  deck[i] = i;
```



# Problem: Deck of Cards, cont.



# Problem: Deck of Cards, cont.

cardNumber / 13 =
- 0 → Spades
- 1 → Hearts
- 2 → Diamonds
- 3 → Clubs

cardNumber % 13 =
- 0 → Ace
- 1 → 2
- .
- .
- 10 → Jack
- 11 → Queen
- 12 → King

GUI Demo (picking four cards)

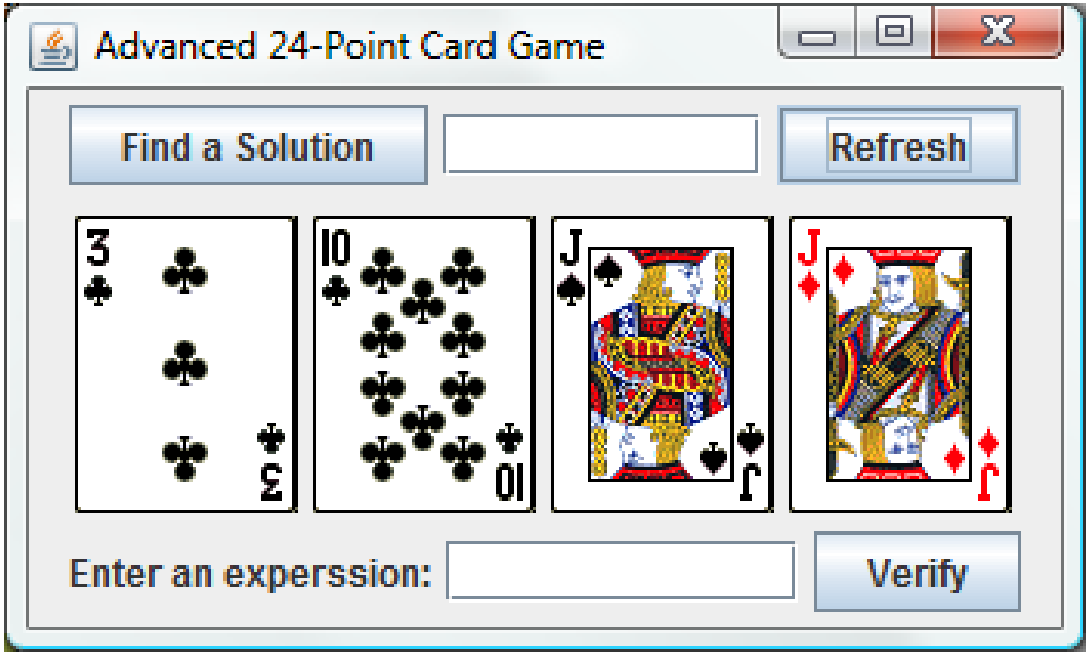
DeckOfCards

Run

# Problem: Deck of Cards

This problem builds a foundation for future more interesting and realistic applications:

See Exercise 22.15.




Run 24 Point Game

Companion Website

# Problem: Lotto Numbers

Suppose you play the Pick-10 lotto. Each ticket has 10 unique numbers ranging from 1 to 99. You buy a lot of tickets. You like to have your tickets to cover all numbers from 1 to 99. Write a program that reads the ticket numbers from a file and checks whether all numbers are covered. Assume the last number in the file is 0.

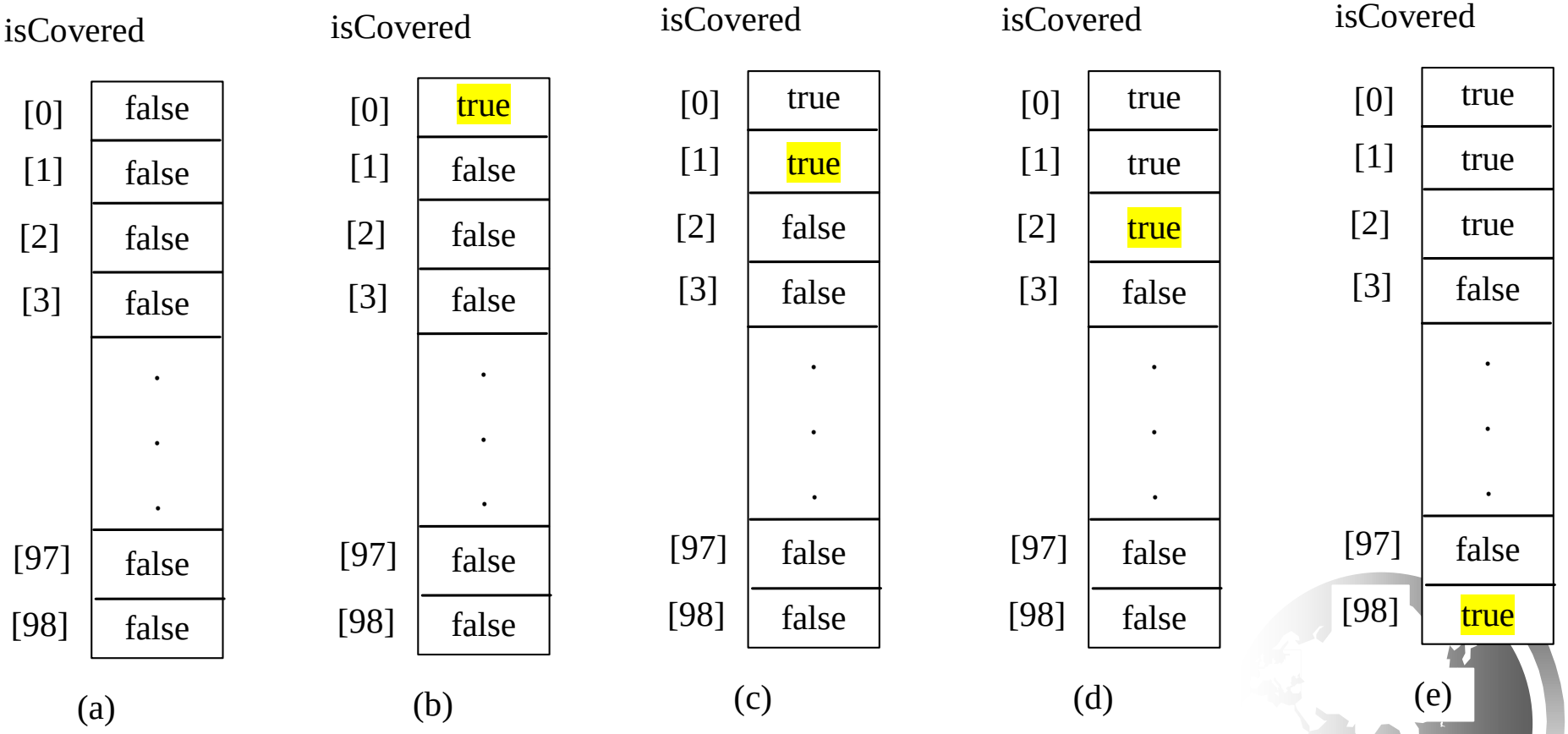
Lotto Numbers Sample Data

LottoNumbers

Run

Companion Website

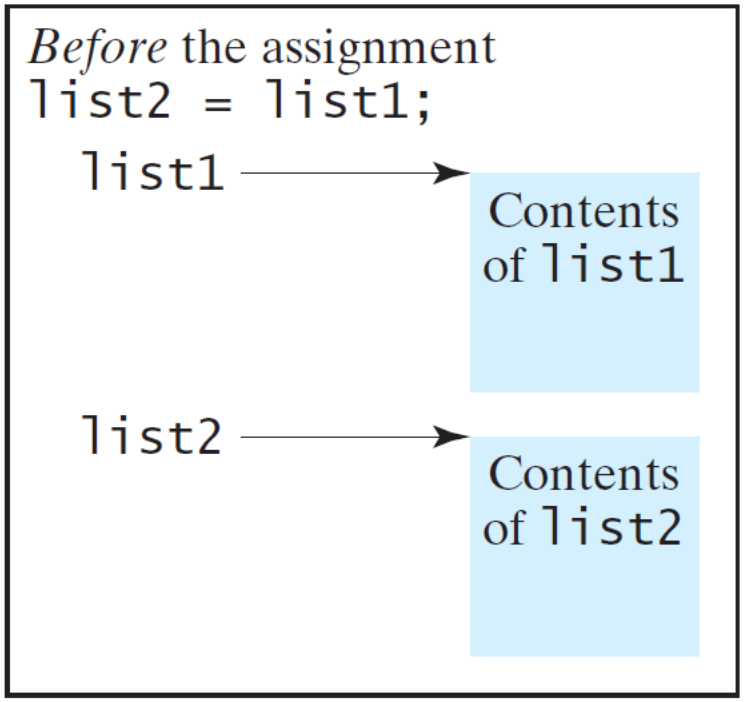
# Problem: Lotto Numbers

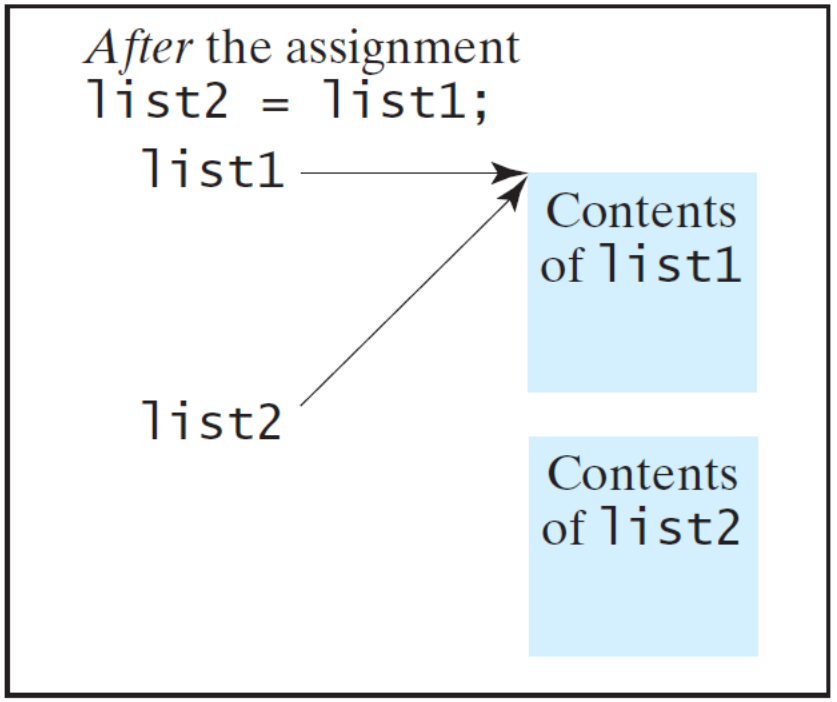


# Copying Arrays

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 follows:

```
list2 = list1;
```



# 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];
```

# The `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

# 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.

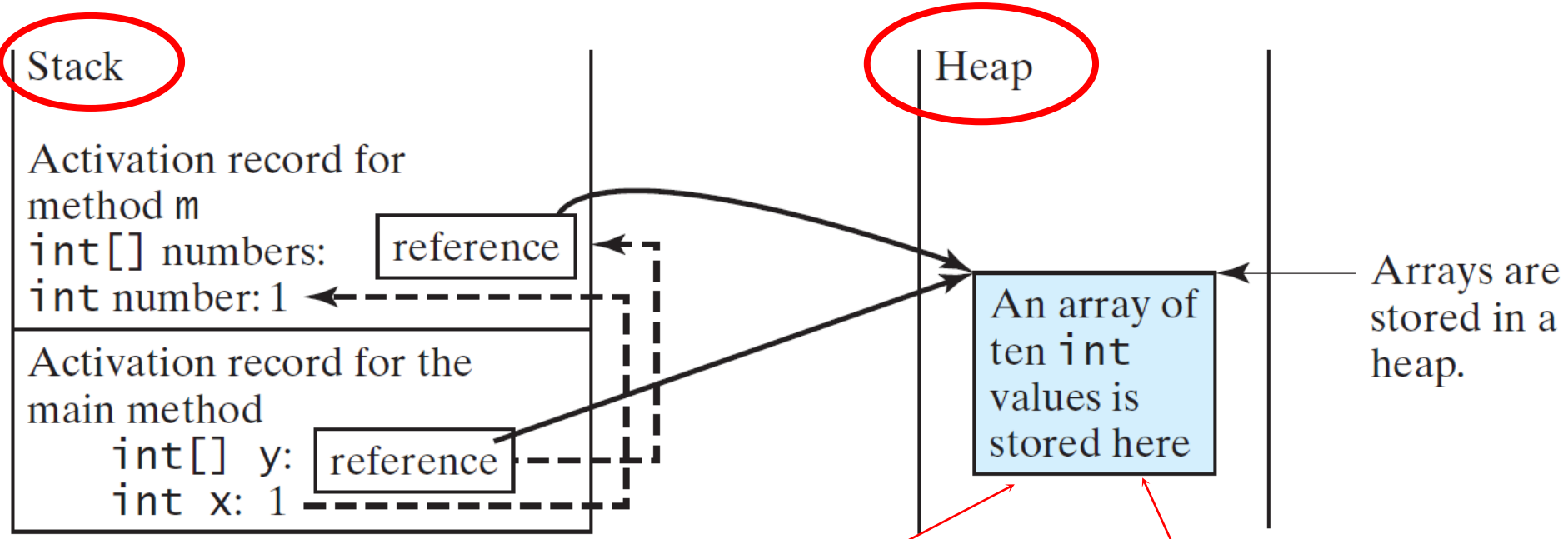
# Simple Example

```
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]
  }
}
```

1, not 1001

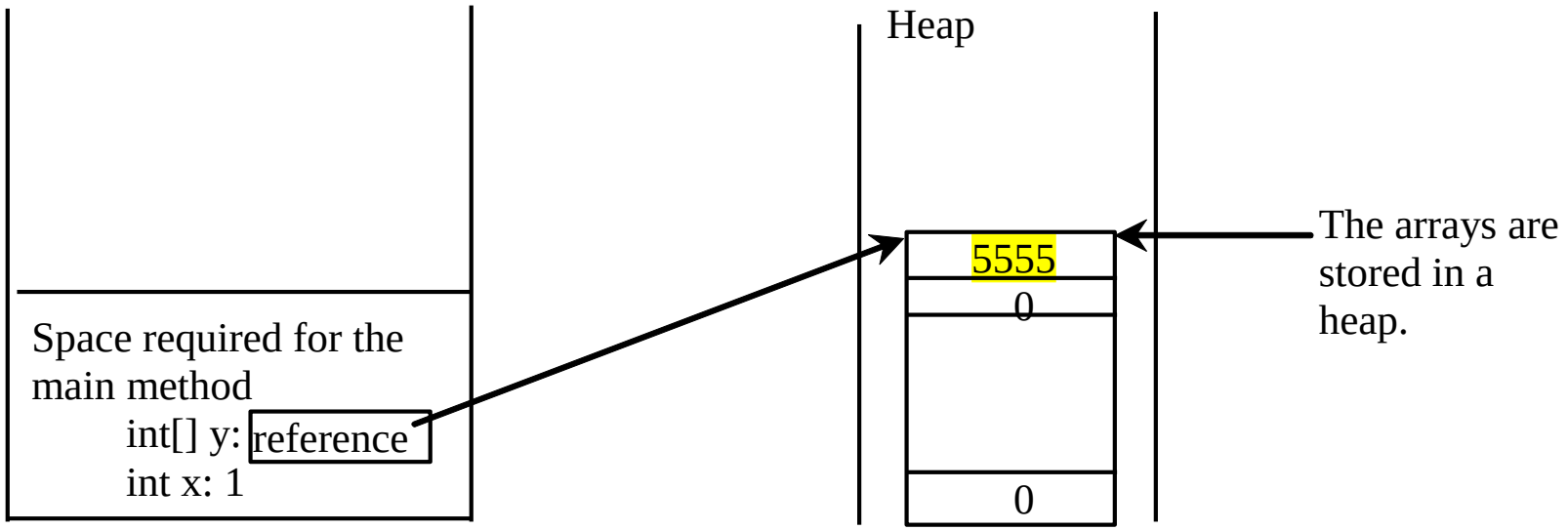
5555

# Call Stack and Heap




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




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.

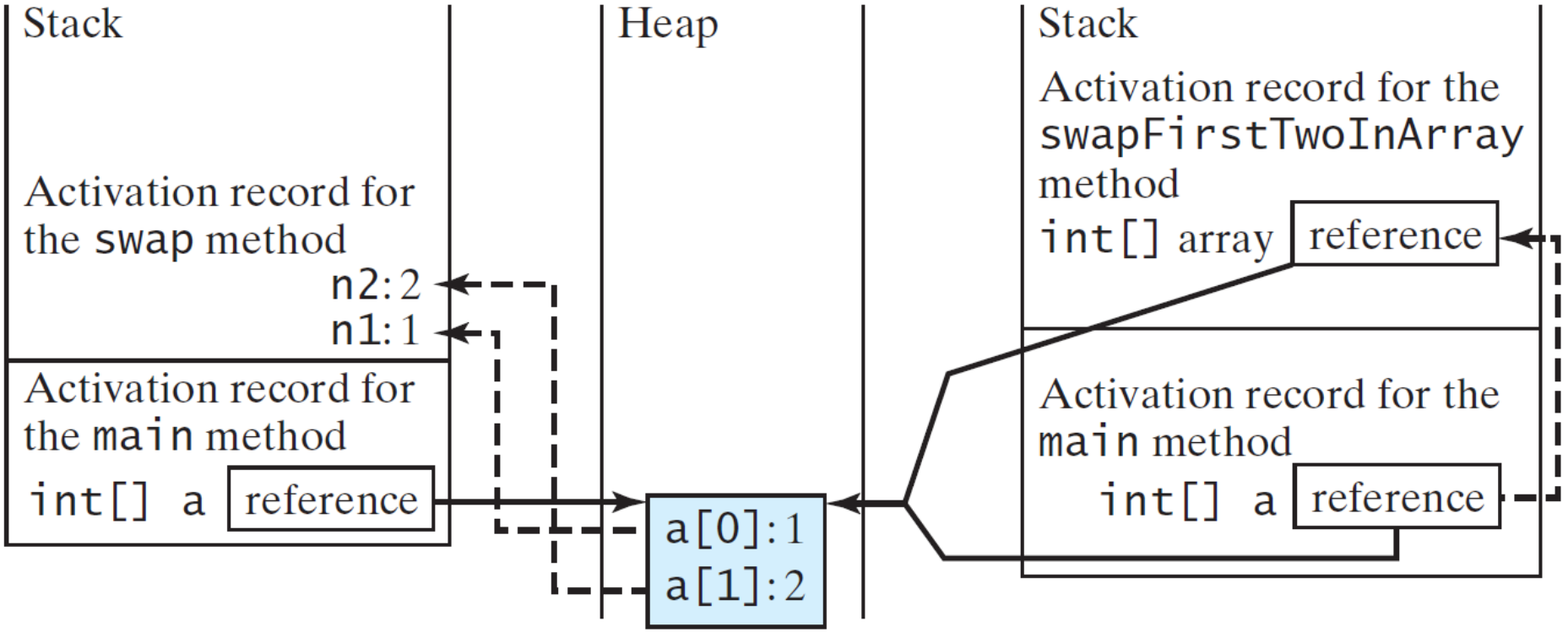
# Passing Arrays as Arguments

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

TestPassArray

Run

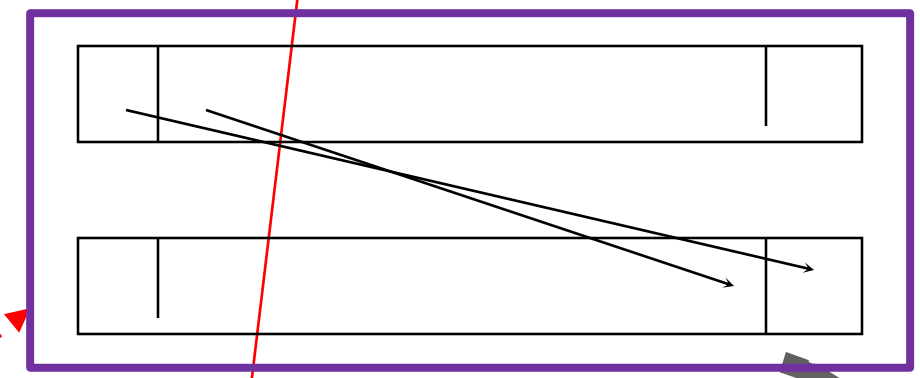
# Call Stack




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.

# 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);
```

**Data in ”result” still exist in “heap” after calling reverse()**

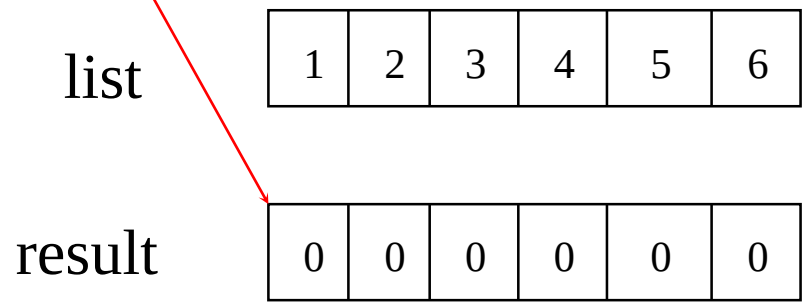
animation

# Trace the reverse Method

```
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;
}
```



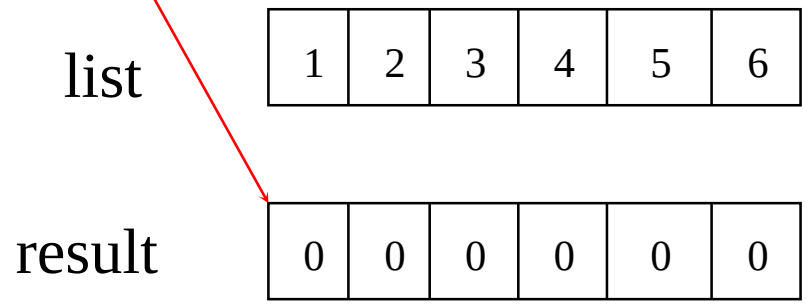
animation

# 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



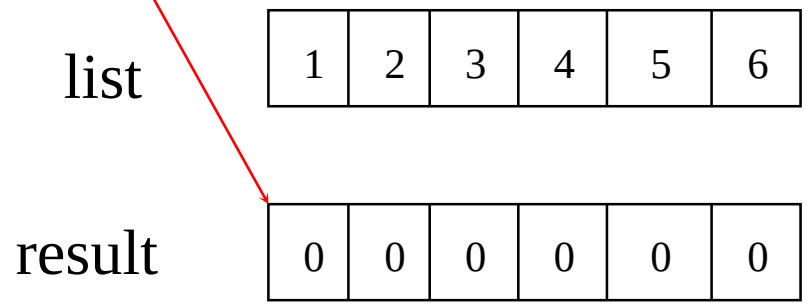
animation

# 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;
}
```



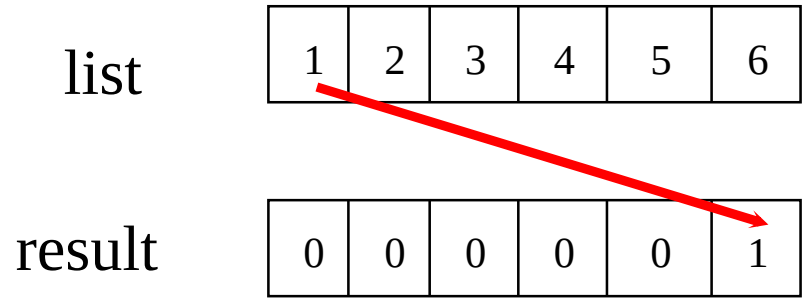
animation

# 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]




animation

# 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 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 = {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 |
|---|---|---|---|---|---|---|

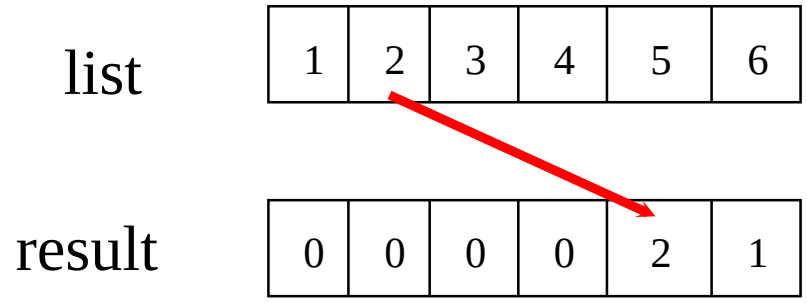
animation

# Trace the reverse Method, cont.

```
int[] list1 = {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 = {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 |
|---|---|---|---|---|---|---|

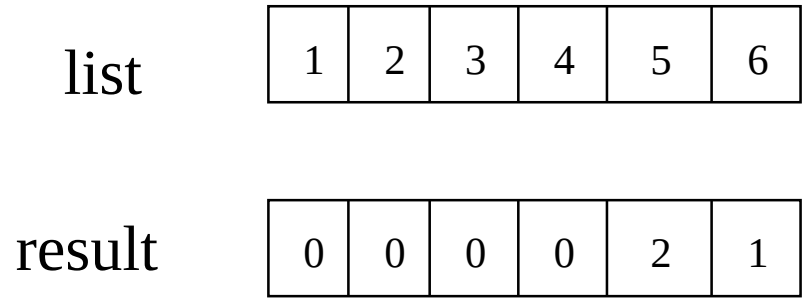
animation

# Trace the reverse Method, cont.

```
int[] list1 = {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



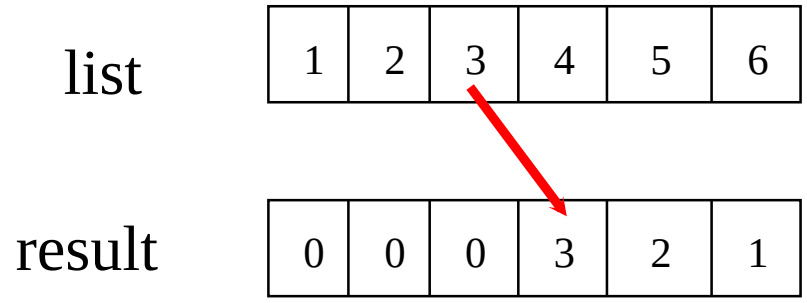

animation

# 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]



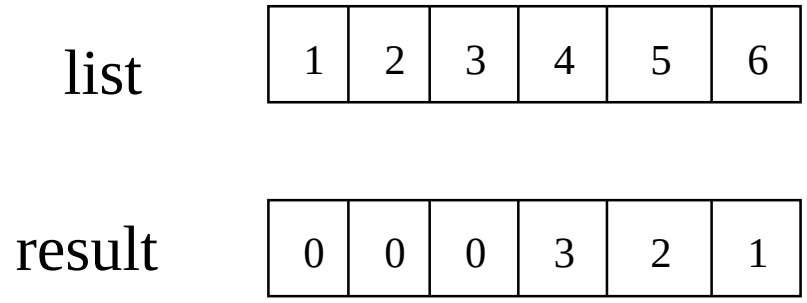
animation

# 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 3 and j becomes 2



animation

# 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

| list | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|

| result | 0 | 0 | 0 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|

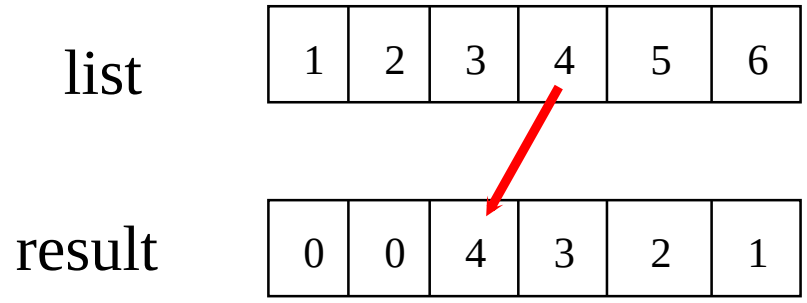
animation

# 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]



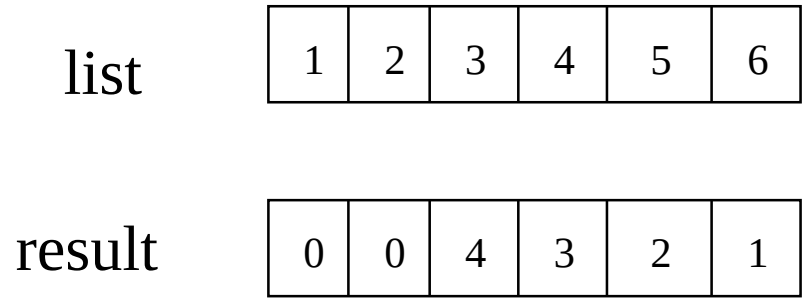
animation

# 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



animation

# 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

| list | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|

| result | 0 | 0 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|

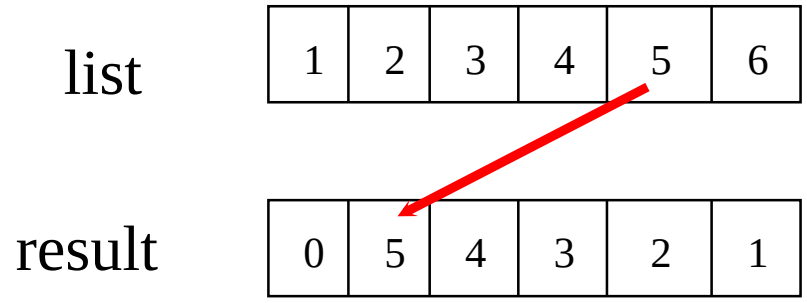
animation

# 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]



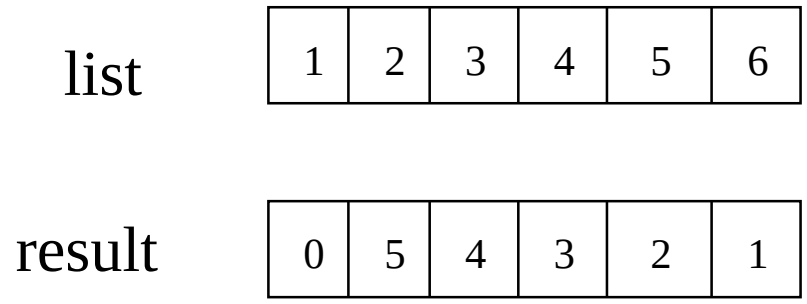
animation

# 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



animation

# 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

| list | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|

| result | 0 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|

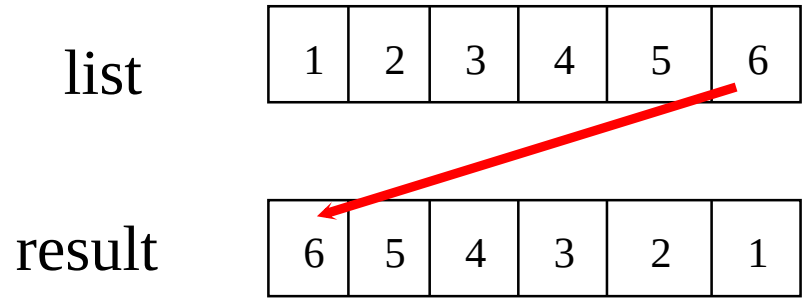
animation

# 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]



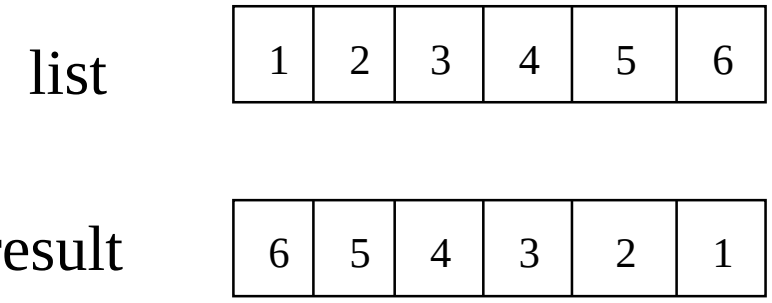
animation

# 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



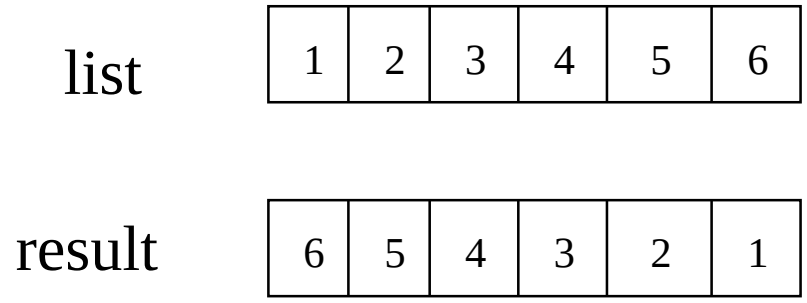

animation

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



animation

# Trace the reverse Method, cont.

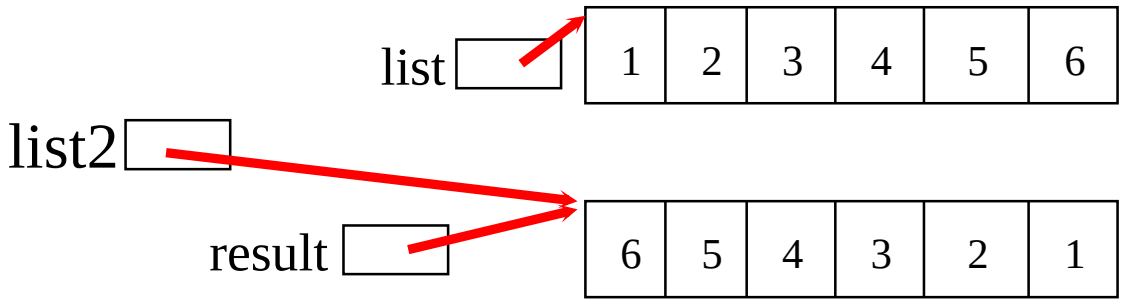
```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
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

list1 and list2 are in main program.

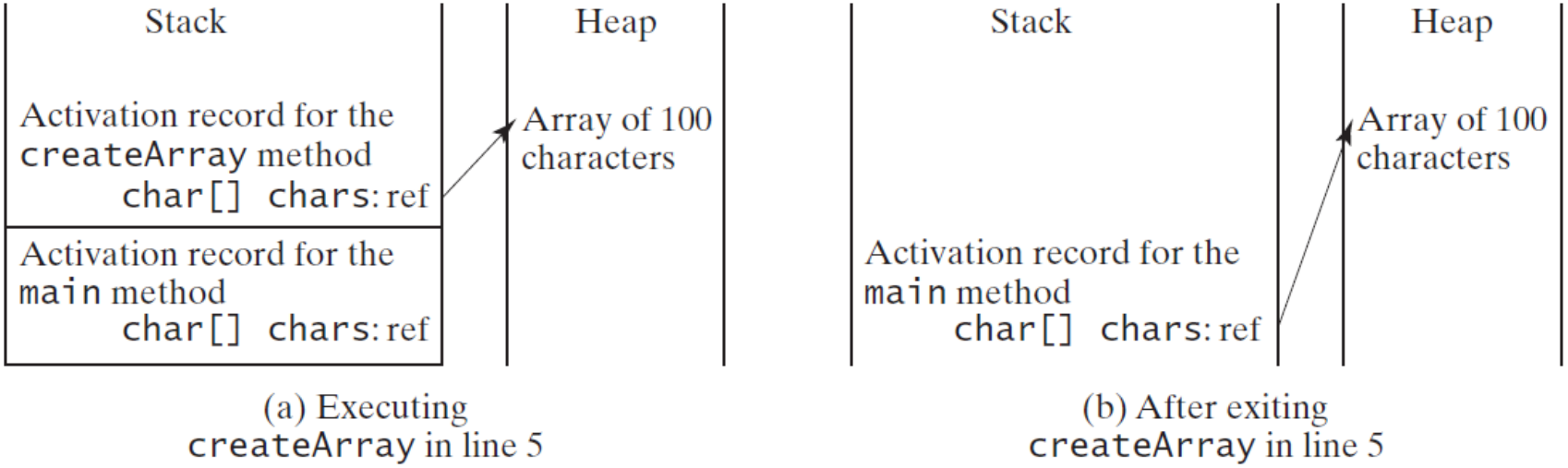
list and result are in reverse().




# Problem: Counting Occurrence of Each Letter

Generate 100 lowercase letters randomly and assign to an array of characters.

Count the occurrence of each letter in the array.



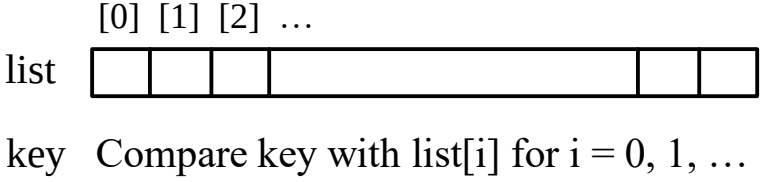

CountLettersInArray

Run

# 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;
  }
}
```

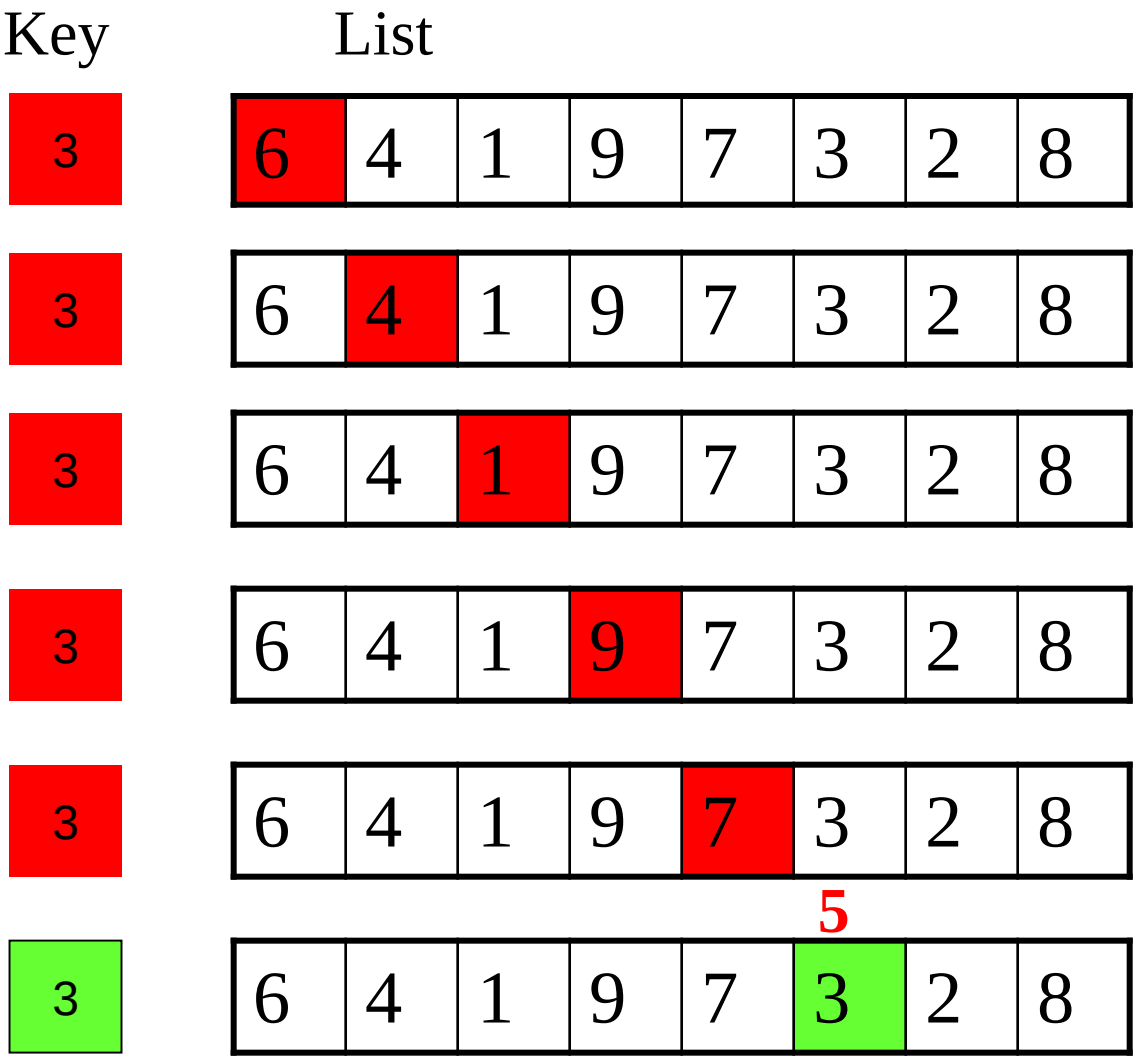


# Linear Search

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.

animation

# 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
```

# Binary Search

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

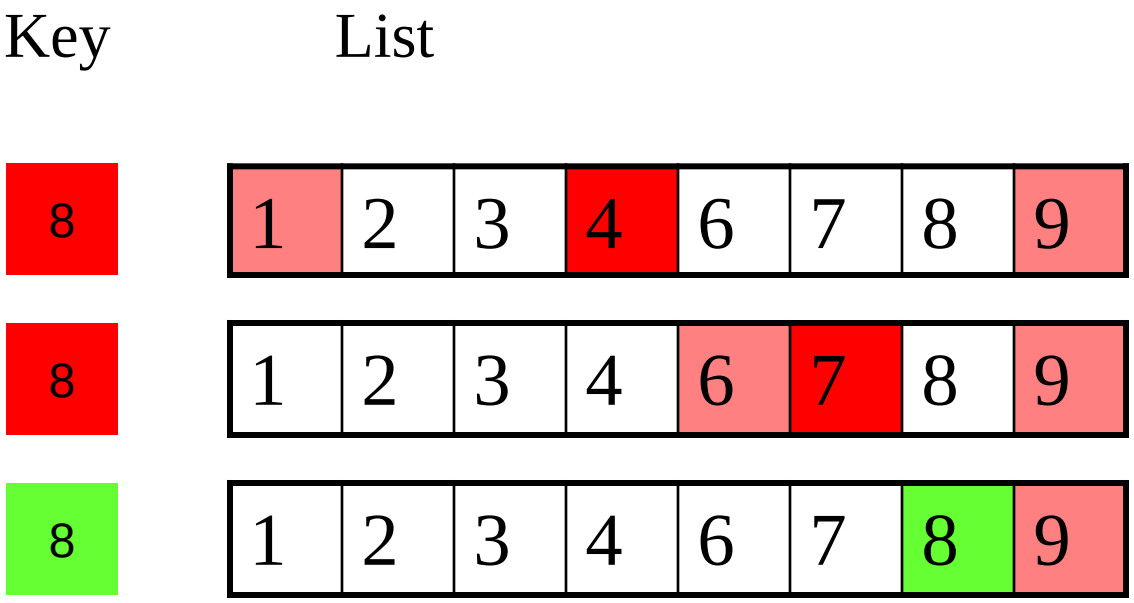
The binary search first compares the key with the element in the middle of the array.

# Binary Search, cont.

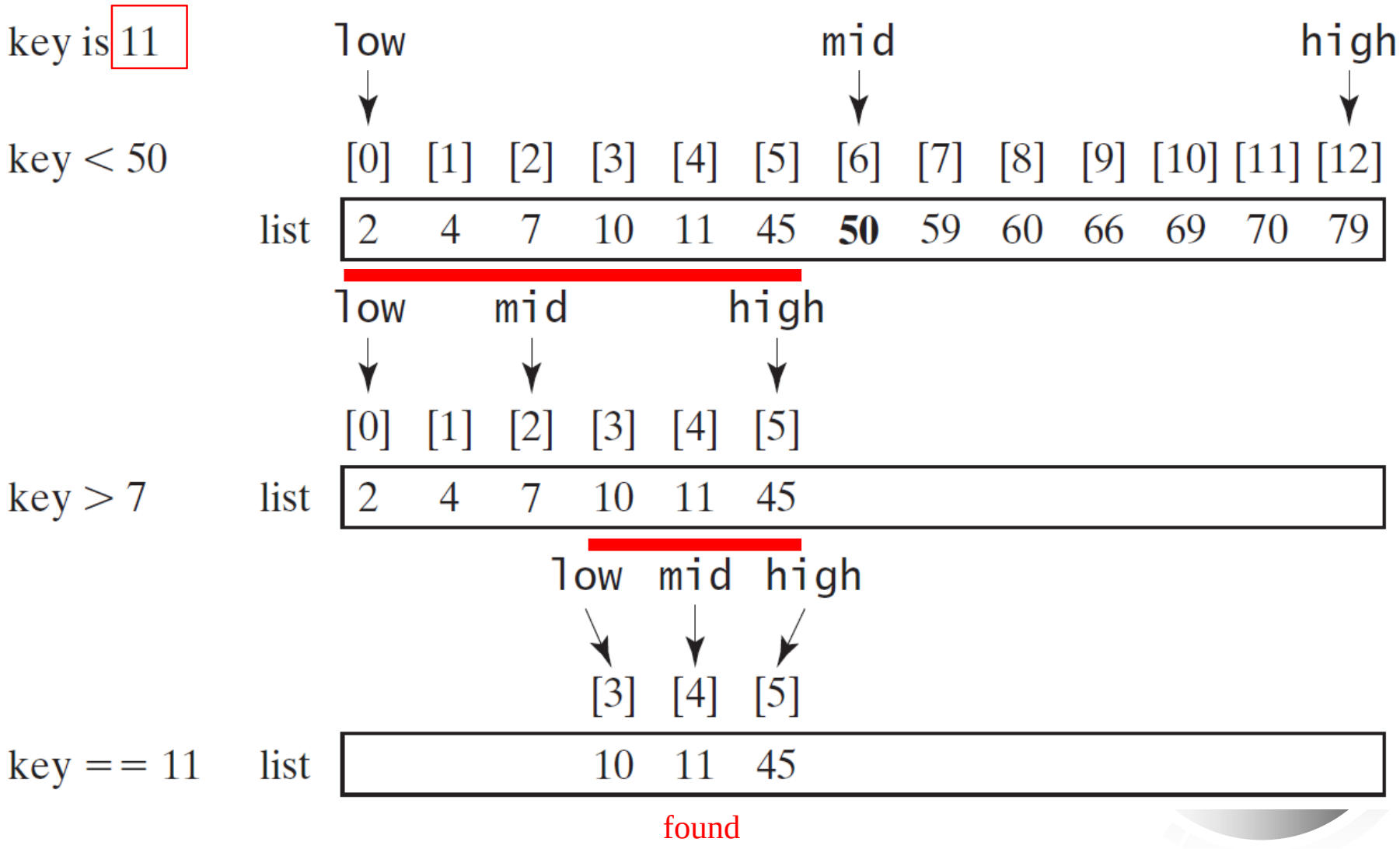
Consider the following three cases:

- 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.

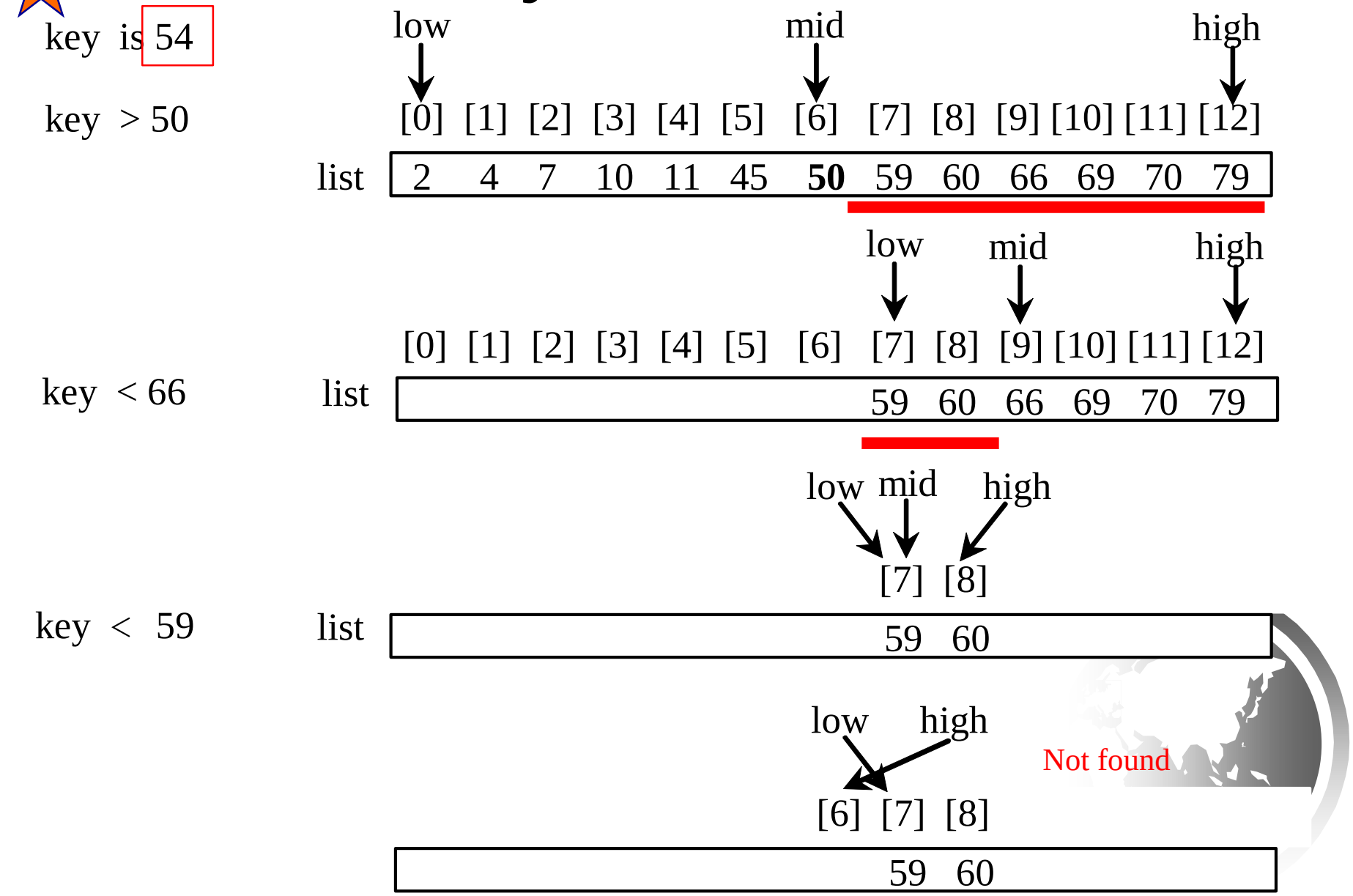

# Binary Search




# Binary Search, cont.



# Binary Search, cont.



# 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.

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

# From Idea to Soluton

```
/** 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;
}
```

# 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

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

Return is –4 (insertion point is 3, so return is -3-1)

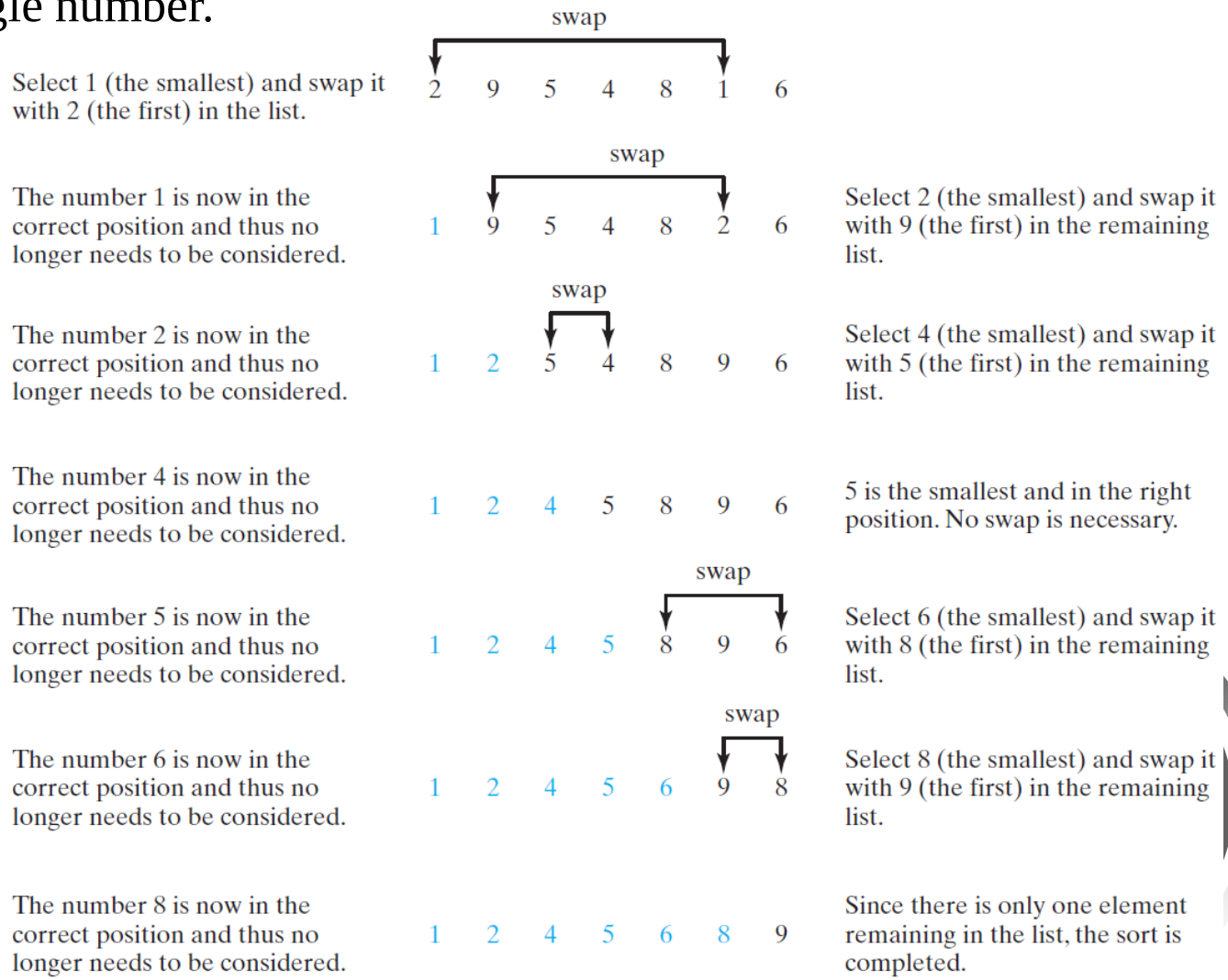
For the binarySearch method to work, the array must be pre-sorted in increasing order.

# Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithms: *selection sort*.

# Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



# From Idea to Solution

```
for (int i = 0; i < list.length; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

```
list[0] list[1] list[2] list[3] ...                 list[10]

list[0] list[1] list[2] list[3] ...                 list[10]

list[0] list[1] list[2] list[3] ...                 list[10]

list[0] list[1] list[2] list[3] ...                 list[10]

list[0] list[1] list[2] list[3] ...                 list[10]

                                ...

list[0] list[1] list[2] list[3] ...                 list[10]
```

```
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

# Expand

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

```
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

Expand

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

```
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

Expand

```
if (currentMinIndex != i) {
  list[currentMinIndex] = list[i];
  list[i] = currentMin;
}
```

# Wrap it in a Method

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length; i++) {
    // Find the minimum in the list[i..list.length-1]
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) { //new min is found
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Invoke it:

selectionSort(yourList)

# 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);
```

Java 8 now provides Arrays.**parallelSort**(list) that utilizes the multicore for fast sorting.

# Pass Arguments to Invoke the Main Method

# Main Method Is Just a Regular Method

You can call a regular method by passing actual parameters. **Can you pass arguments to main?** Of course, yes. For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

# Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
    ...
  }
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

# Processing Command-Line Parameters

In the main method, get the arguments from `args[0]`, `args[1]`, `...`, `args[n]`, which corresponds to `arg0`, `arg1`, `...`, `argn` in the command line.

# Problem: Calculator

Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

Calculator

Run

java Calculator 2 + 3

java Calculator 2 - 3

java Calculator 2 / 3

java Calculator 2 . 3