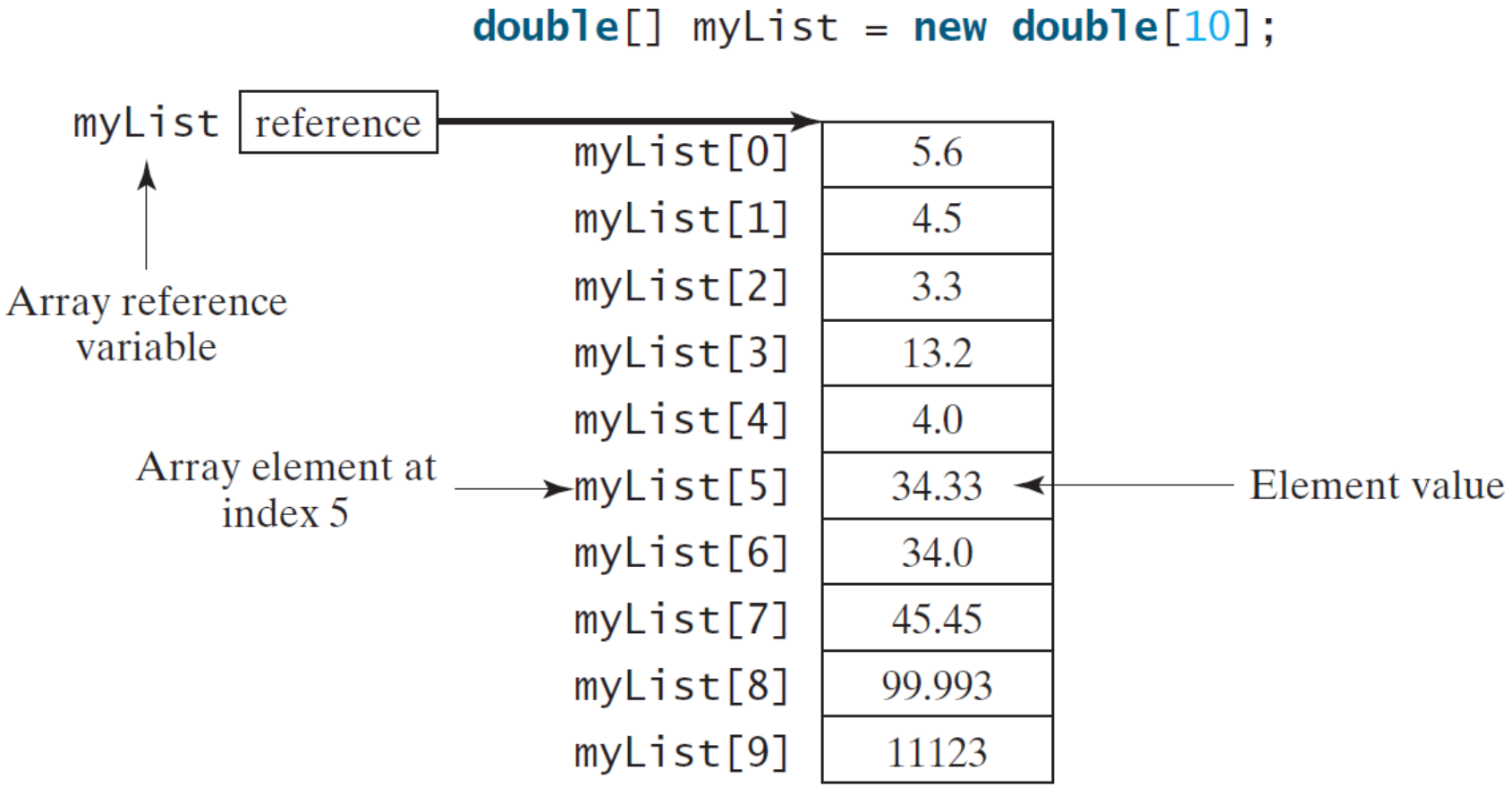
NYU

Lecture 5

Arrays

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:
  - 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
  - The index starts from 0 to `arrayRefVar.length-1`.
  - In the example `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.

- Example: add 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

- `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
  - Initialize
- the array all in one statement.
- Splitting it would cause a syntax error.

- Example (wrong):

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

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

0	0
1	0
2	0
3	0
4	0

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];
  }
}
```

0	0
1	0
2	0
3	0
4	0

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];
  }
}
```

0	0
1	0
2	0
3	0
4	0

Trace Program with Arrays

After this line is executed, `value[1]` is 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];
  }
}
```

0	0
1	1
2	0
3	0
4	0

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];
  }
}
```

0	0
1	1
2	0
3	0
4	0

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

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

After this line is executed, `value[2]` is 3 = (2 + 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];
  }
}
```

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

After `i++`, `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];
  }
}
```

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

i (=3) 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];
  }
}
```

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

After this line is executed, `value[3]` is 6 = (3 + 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];
  }
}
```

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

After `i++`, `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];
  }
}
```

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

i (=4) 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];
  }
}
```

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

After this line is executed, `value[4]` is 10 = (4 + 6)

```
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];
  }
}
```

0	0
1	1
2	3
3	6
4	10

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];
  }
}
```

0	0
1	1
2	3
3	6
4	10

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];
  }
}
```

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

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

```
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];
  }
}
```

0	11
1	1
2	3
3	6
4	10

Examples

- Initializing arrays with input values
- Initializing arrays with random values

- Printing arrays
- Summing all elements

- Finding the largest element
- Finding the smallest index of the largest element

- Random shuffling
- Shifting elements

Initializing arrays with input values

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

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

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


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

Printing arrays

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


...


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

Summing all elements

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


...


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

Finding the largest element

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


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

Random shuffling

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

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

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

Shifting Elements

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

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

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

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

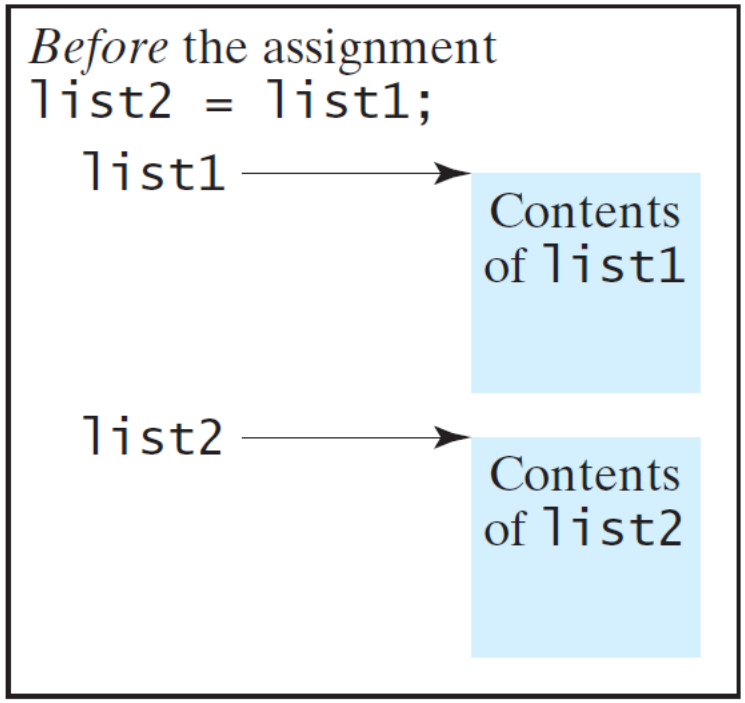
- In general, the syntax is

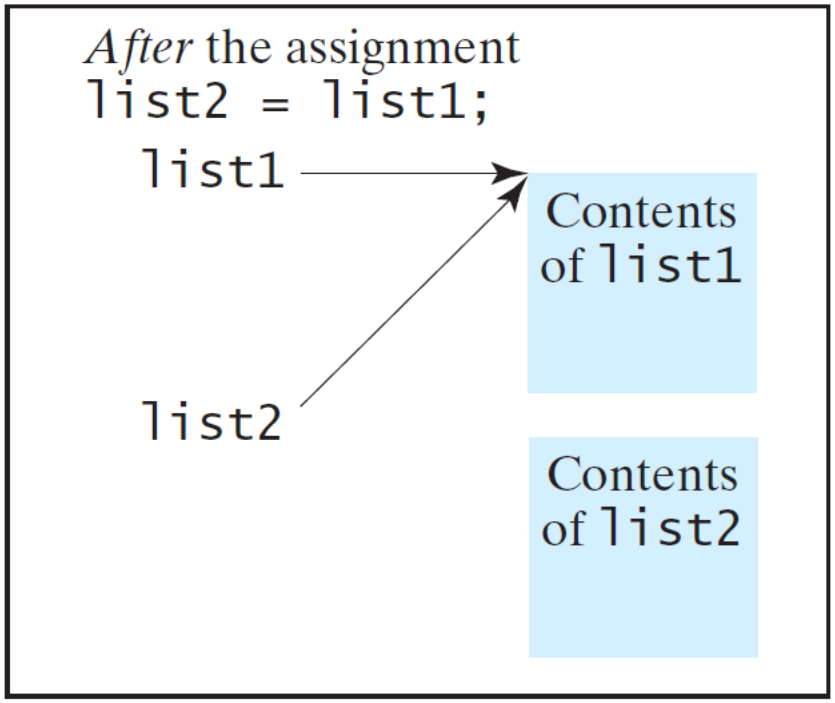
```
for (elementType value: arrayRefVar) {
  // Process the value
}
```

- 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.

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 (=)
  - `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

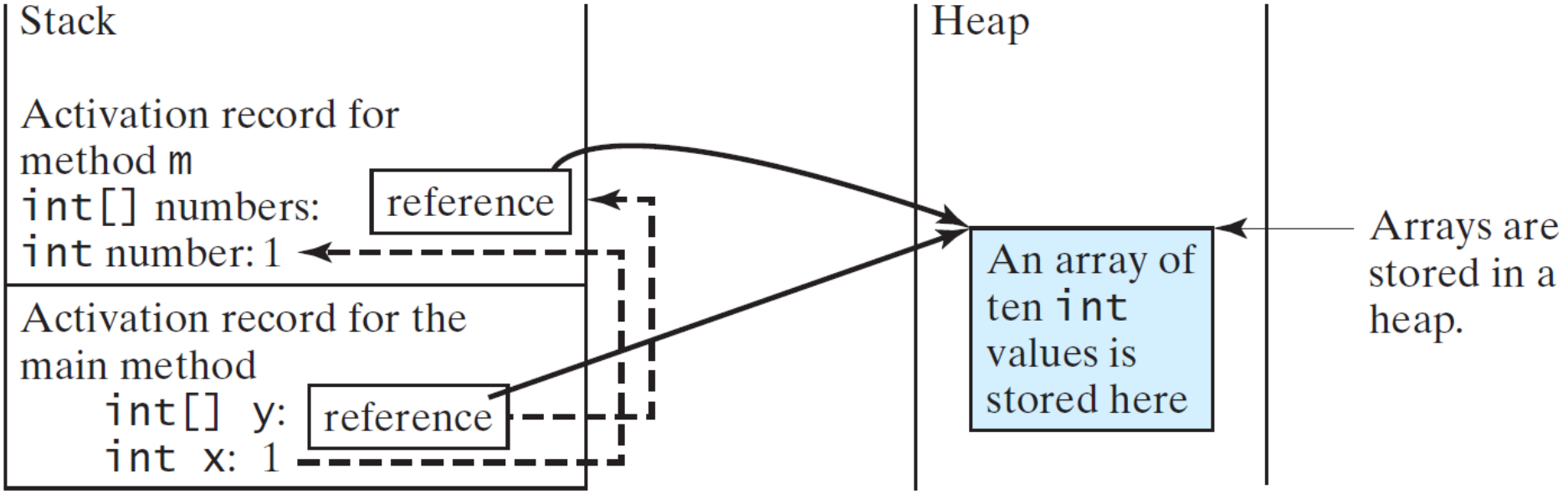
- Java uses pass by value to pass arguments to a method.
- There are important differences between passing a value of variables of **primitive data** types and passing **arrays**.
- For a parameter of a **primitive type value**, the actual value is passed. Changing the value of the local parameter inside 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]
  }
}
```

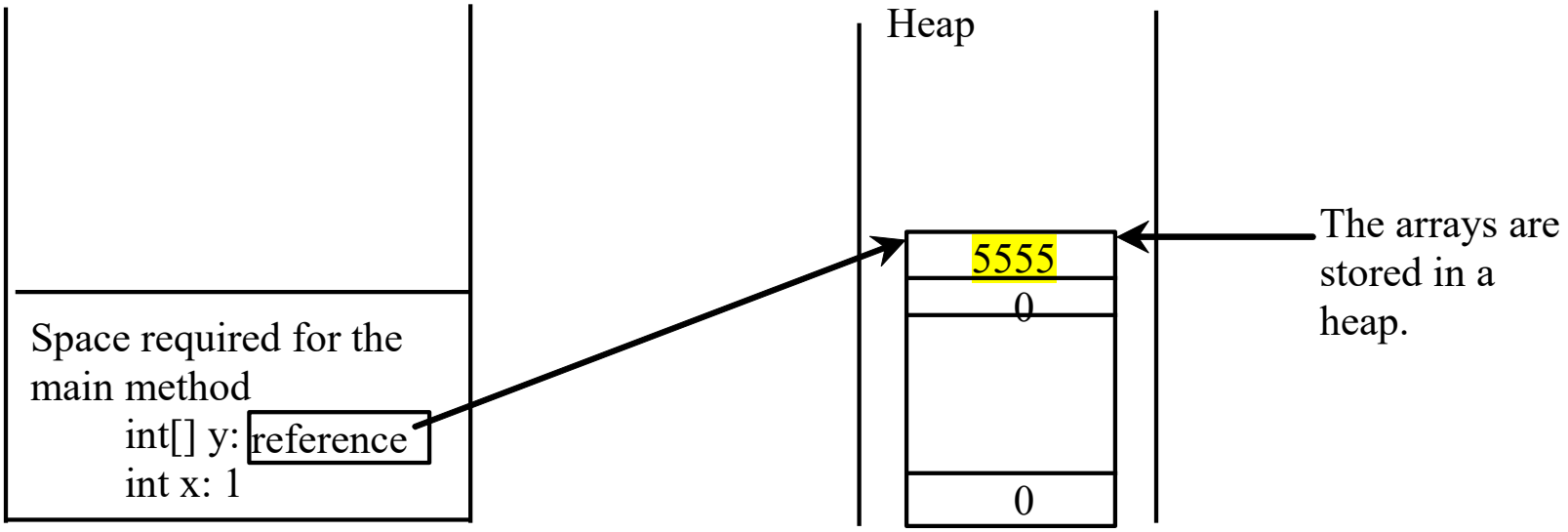
Call Stack

- 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.

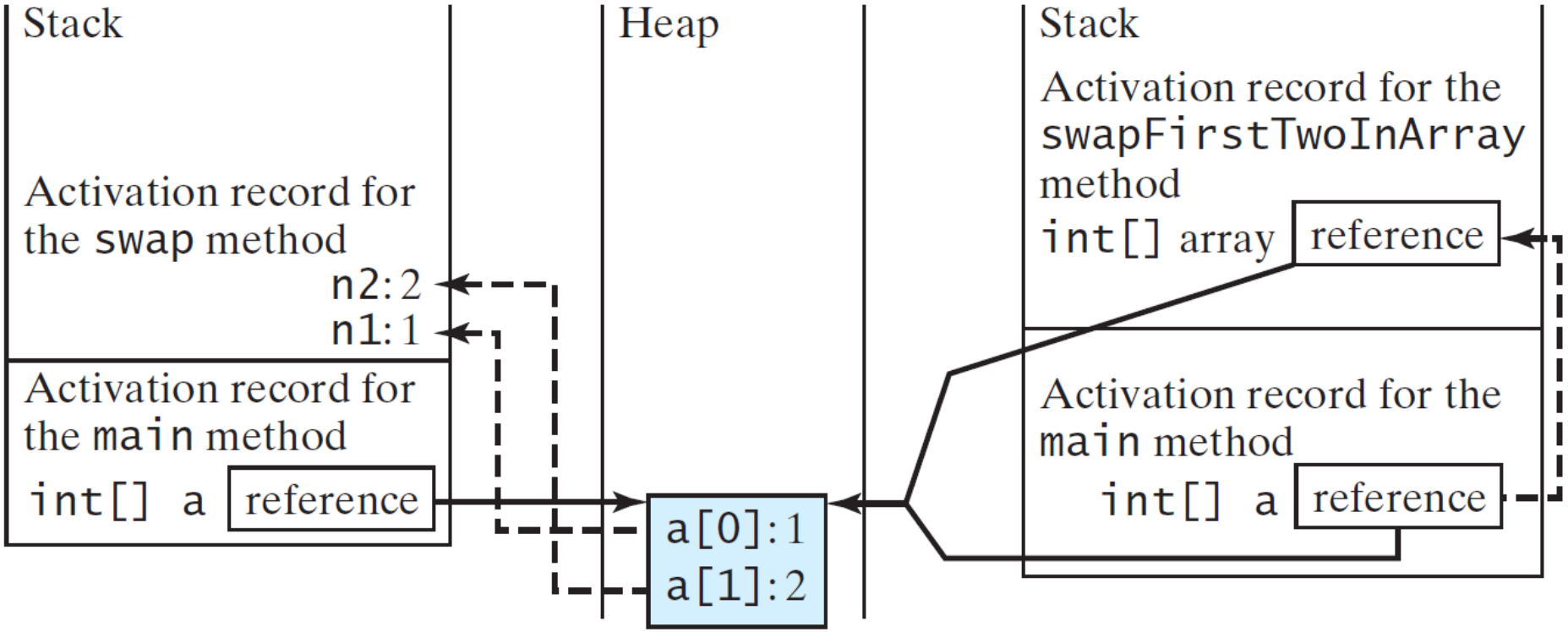


Heap

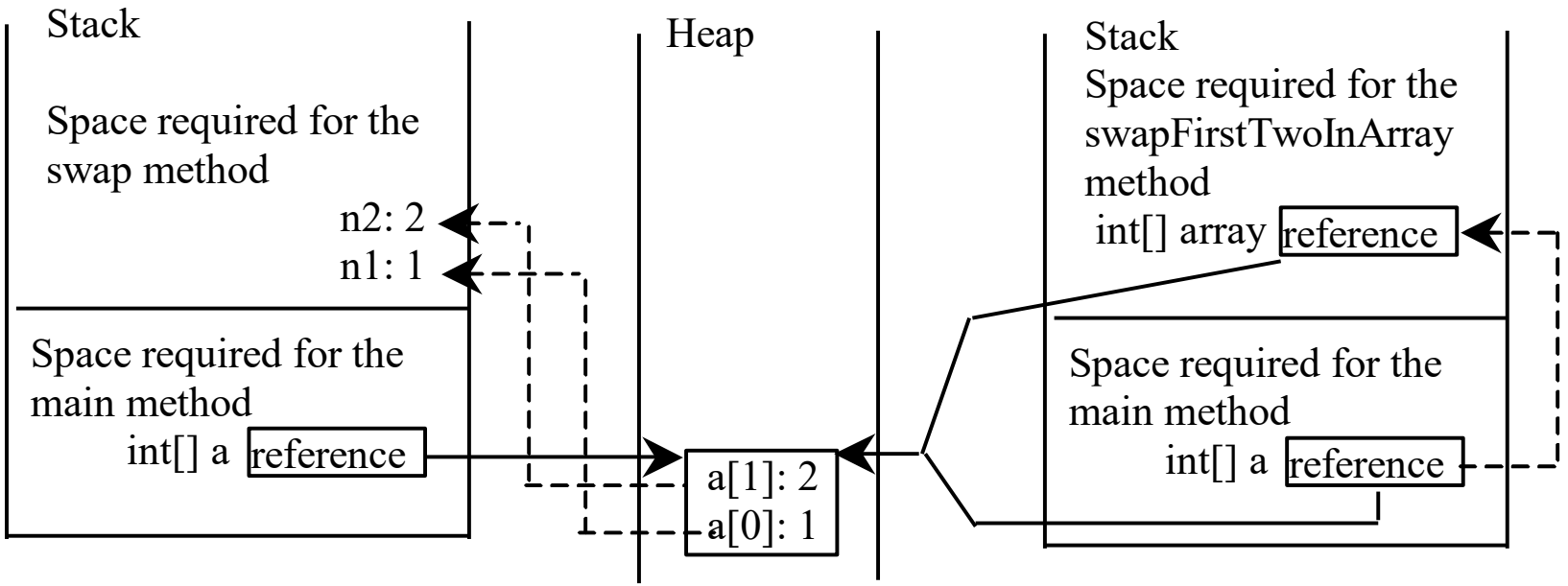
- The JVM stores the array in an area of memory, called heap, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.



Call Stack


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

Example



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.

Variable-Length Arguments

- You can pass a variable number of arguments of the same type to a method by using an array.

- Example:

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

    return average / numbers.length;
}
```

Returning an Array from a Method

```
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;
}
```

Array is declared and created

list1

1	2	3	4	5	6

Returning an Array from a Method

```
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;
}
```

The **reference** of `list1` is passed to the method reverse

list1

1	2	3	4	5	6

Returning an Array from a Method

```
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;
}
```

Declare `result` and create array

list "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	0

Returning an Array from a Method

```
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

list "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	0

Returning an Array from a Method

```
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) is less than 6

list "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	0

Returning an Array from a Method

```
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]

list "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	1

Returning an Array from a Method

```
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 "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	1

Returning an Array from a Method

```
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 "= list1"

1	2	3	4	5	6

result

0	0	0	0	0	1

Returning an Array from a Method

```
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]`

`list` "= `list1`"

1	2	3	4	5	6

`result`

0	0	0	0	2	1

Returning an Array from a Method

```
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 "= list1"

1	2	3	4	5	6

result

0	0	0	0	2	1

Returning an Array from a Method

```
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 less than 6

list "= list1"

1	2	3	4	5	6

result

0	0	0	0	2	1

Returning an Array from a Method

```
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]

list "= list1"

1	2	3	4	5	6

result

0	0	0	3	2	1

Returning an Array from a Method

```
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

list "= list1"

1	2	3	4	5	6

result

0	0	0	3	2	1

Returning an Array from a Method

```
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 less than 6

list "= list1"

1	2	3	4	5	6

result

0	0	0	3	2	1

Returning an Array from a Method

```
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]`

`list "= list1"`

1	2	3	4	5	6

`result`

0	0	4	3	2	1

Returning an Array from a Method

```
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

list "= list1"

1	2	3	4	5	6

result

0	0	4	3	2	1

Returning an Array from a Method

```
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 less than 6

list "= list1"

1	2	3	4	5	6

result

0	0	4	3	2	1

Returning an Array from a Method

```
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]`

`list "= list1"`

1	2	3	4	5	6

`result`

0	5	4	3	2	1

Returning an Array from a Method

```
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

list "= list1"

1	2	3	4	5	6

result

0	5	4	3	2	1

Returning an Array from a Method

```
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 less than 6

list "= list1"

1	2	3	4	5	6

result

0	5	4	3	2	1

Returning an Array from a Method

```
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]`

`list "= list1"`

1	2	3	4	5	6

`result`

6	5	4	3	2	1

Returning an Array from a Method

```
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

list "= list1"

1	2	3	4	5	6

result

6	5	4	3	2	1

Returning an Array from a Method

```
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.

`list "= list1"`

1	2	3	4	5	6

`result`

6	5	4	3	2	1

Returning an Array from a Method

```
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`

list "= list1"

1	2	3	4	5	6

result

6	5	4	3	2	1

Returning an Array from a Method

```
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;
}
```

The reference of "result" is copied to list2

list1

1	2	3	4	5	6

list2 = "result"

6	5	4	3	2	1

Searching Arrays

- Searching is the process of looking for a specific element in an array
- 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.

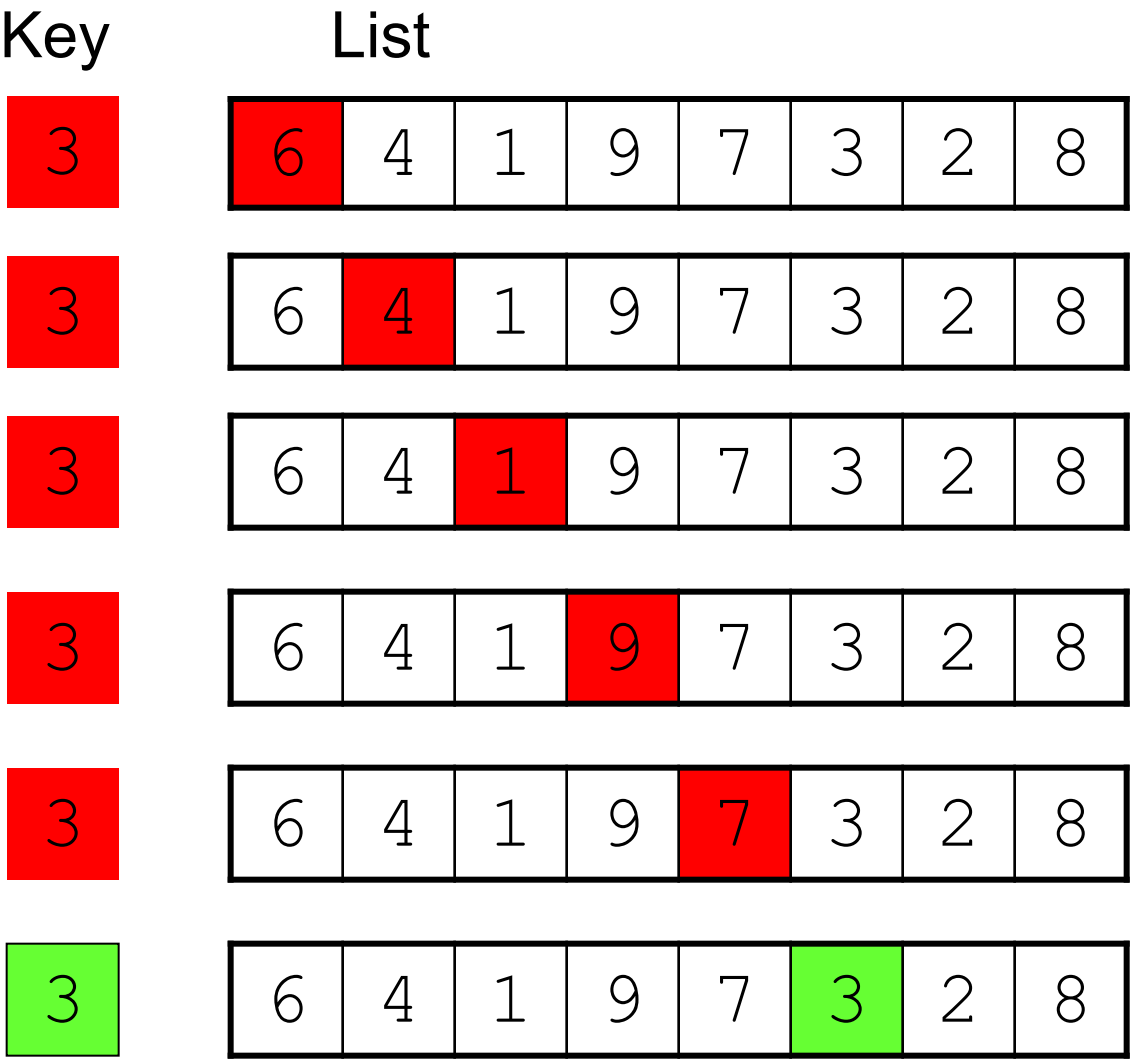
Linear Search

- 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

```
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;
  }
}

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
```

Linear Search



Linear 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;
  }
}

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
```

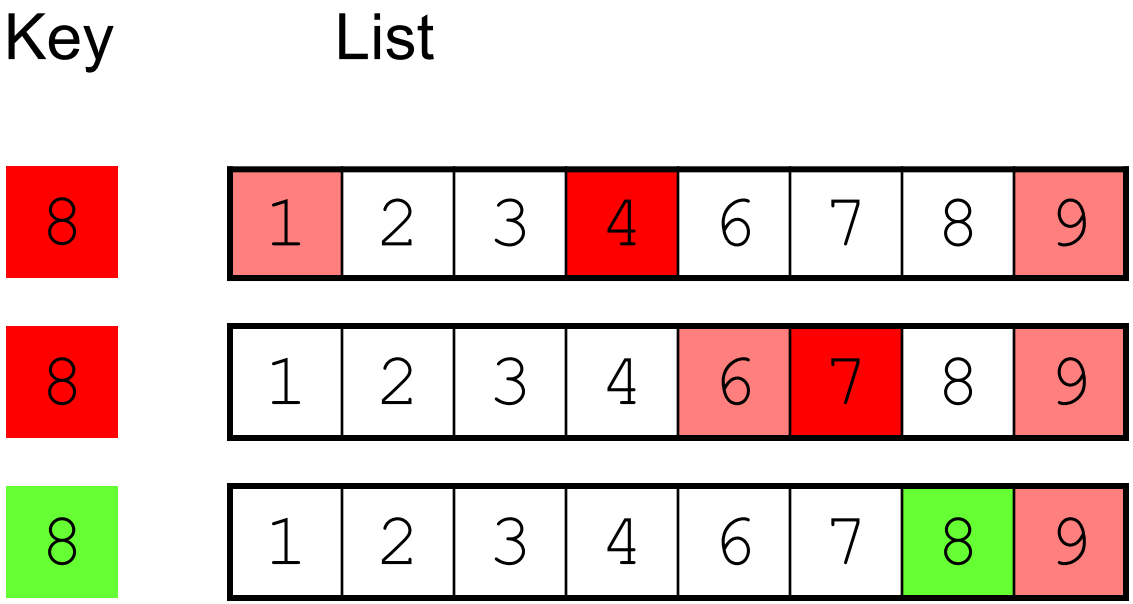
Binary Search

- 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.
  - Example: 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.

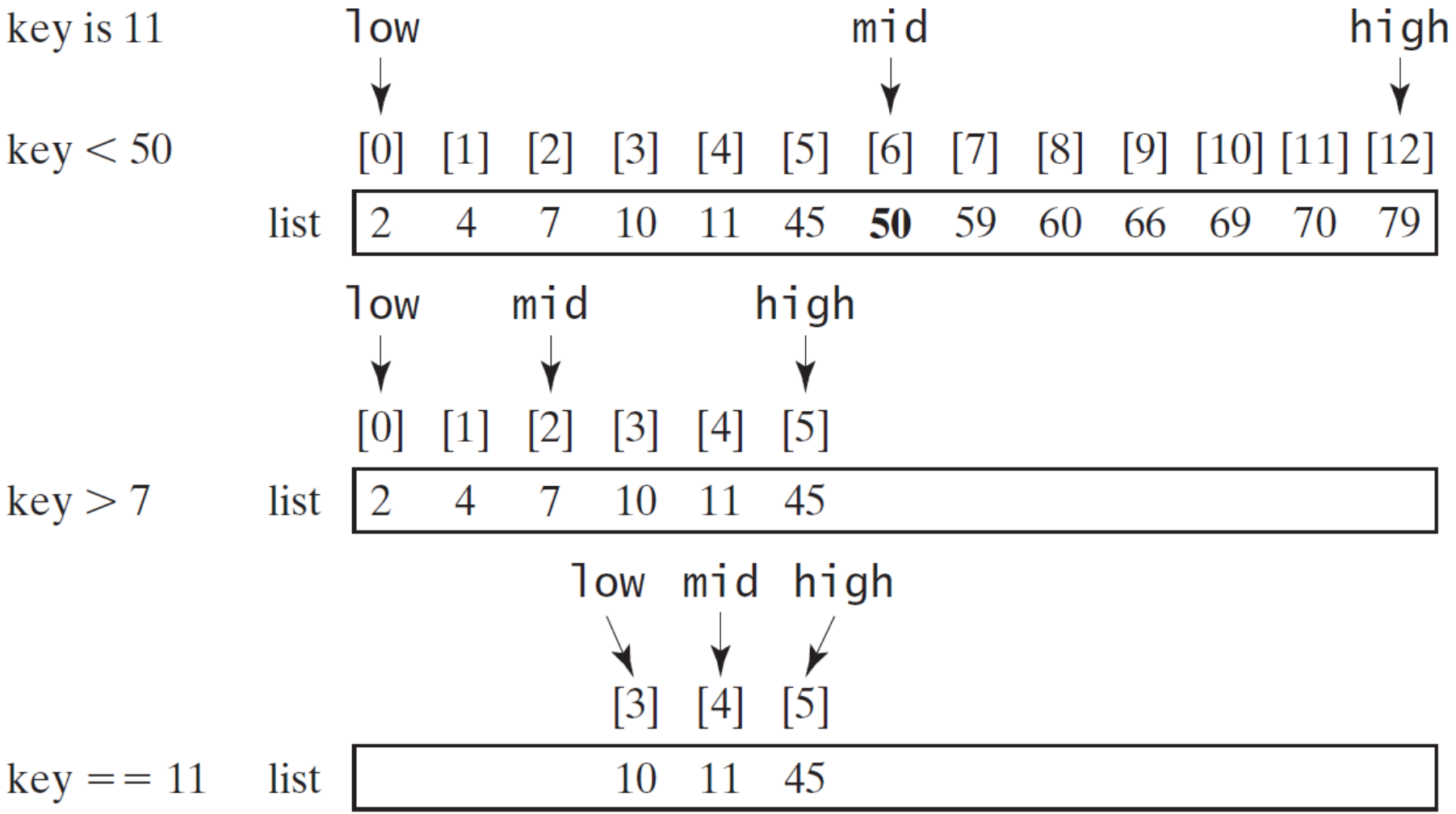
Binary Search, cont.

- Consider the following three cases:
  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.

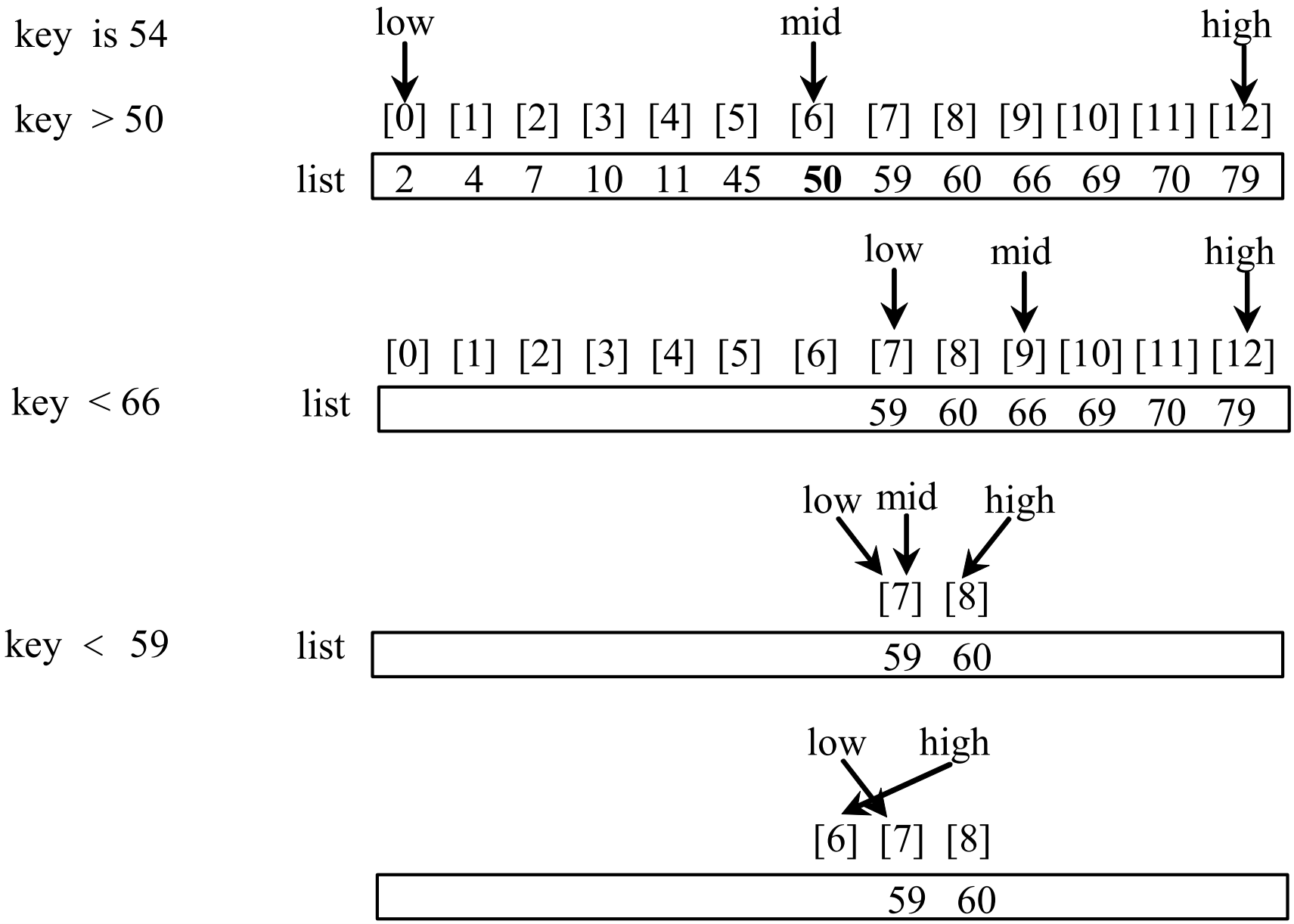
Binary Search



Binary Search



Binary Search



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

- 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 `java.util.Arrays`.

- Example:

```
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'));
```

- 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.
  - Insertion sort
  - **Selection sort**
  - Merge sort
  - Quicksort
  - Bubble sort
  - Counting sort
  - …

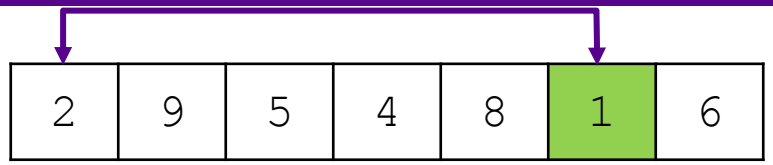
Selection Sort

- Selection sort finds the smallest number in the list and places it first.

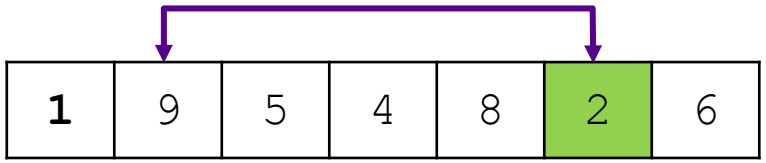
- It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Selection Sort

Select 1 (the smallest) and swap it with 2 (the first)

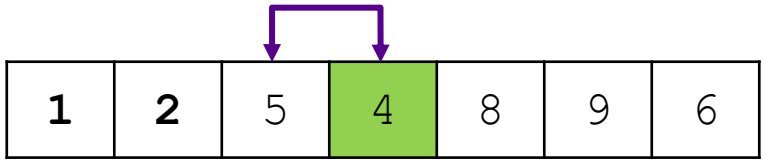


1 is in the correct position, and it is not considered



Select 2 (the smallest) and swap it with 9 (the first)

2 is in the correct position, and it is not considered

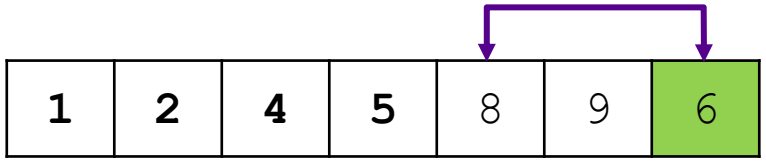


Select 4 (the smallest) and swap it with 5 (the first)

4 is in the correct position, and it is not considered

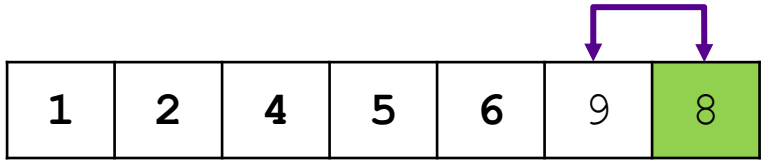
5 is the smallest and in the correct position. No swap

5 is in the correct position, and it is not considered



Select 6 (the smallest) and swap it with 8 (the first)

6 is in the correct position, and it is not considered



Select 8 (the smallest) and swap it with 9 (the first)

8 is in the correct position, and it is not considered

Only one element left. The sort is completed

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+1..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]
```

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+1..listSize-1]
}
```

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

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+1..listSize-1]
}
```

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

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+1..listSize-1]
}
```

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

Wrap it in a Method

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

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.
- Example:

```
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.

The Arrays.toString(list) Method

- The `Arrays.toString(list)` method can be used to return a string representation for the list.

- Example:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
String s = java.util.Arrays.toString(numbers);
System.out.println(s);
```

Main Is Just a Regular Method

- You can call a regular method by passing actual parameters.

- Can you pass arguments to main?
- Of course, yes.

- Example:
  - the main method in class B is invoked by a method in A

```
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
```

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