

JAVATM

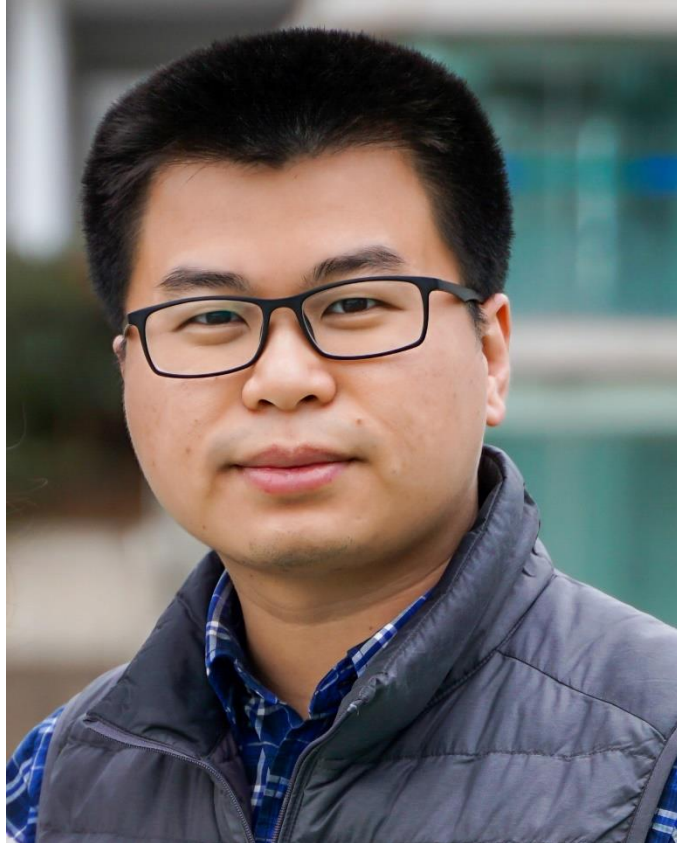


MẢNG MỘT CHIỀU

GIẢNG VIÊN

TS. Hà Ngọc Long

THÔNG TIN GIẢNG VIÊN



TS. Hà Ngọc Long

Email: hnlng@hueuni.edu.vn

- Thông tin Giảng viên
 - Tiến sĩ ngành Hệ thống Thông tin và Truyền thông, Hàn Quốc.
 - Thạc sĩ ngành Kỹ thuật Quản trị Công nghiệp, Hàn Quốc.
 - Cử nhân ngành Hệ thống Thông tin Quản lý, Đại học Kinh tế Huế
- Hướng nghiên cứu:
 - Quản trị quy trình nghiệp vụ (Business Process Management).
 - Chuyển đổi số.
 - FinTech, RegTech
 - Tiền điện tử, hợp đồng thông minh, và Blockchain.
 - Khai phá quy trình nghiệp vụ (Process Mining)

MỤC TIÊU CỦA HỌC PHẦN

“Giới thiệu về ngôn ngữ lập trình hướng đối tượng Java. Tập trung giới thiệu cho sinh viên kỹ năng, kỹ thuật lập trình theo phương pháp hướng đối tượng”

TIÊU CHUẨN ĐÁNH GIÁ SINH VIÊN

Hình thức Đánh giá	% Điểm
Chuyên cần	5%
Phát Biểu & Thảo Luận	5%
Bài Tập Về Nhà	5%
Thực Hành & Thực Tế	15%
Kiểm tra giữa kỳ	15%
Bài tập cá nhân	10%
<i>Thi kết thúc học phần</i>	<i>45%</i>
Tổng cộng:	100%

NỘI DUNG MÔN HỌC

1

- Java cơ bản

2

- Nhập môn lập trình HĐT

3

- Lập trình HĐT với Java

4

- Lập trình đồ họa và sự kiện trong Java

5

- Cấu trúc dữ liệu nâng cao trong Java

TÀI LIỆU HỌC TẬP

- *Y. Daniel Liang* (2019) Introduction to Java Programming and Data Structures - Comprehensive Version (12th edition), *Pearson*
- *C Thomas Wu*. (2004). An introduction to object oriented programming with Java. *McGraw-Hill – Boston*.

Tài liệu hướng dẫn và phần mềm thực hành:

- Apache Netbeans IDE 12 (or higher)
- Java SE 17 (or higher)
- Videos bài giảng và hướng dẫn thực hành



Chapter 7 Single-Dimensional Arrays

Opening Problem

- Nhập vào một trăm số nguyên, sau đó (1) tính trung bình cộng các số đó, và (2) tìm xem thử bao nhiêu số trong 100 số đó có giá trị lớn hơn trung bình

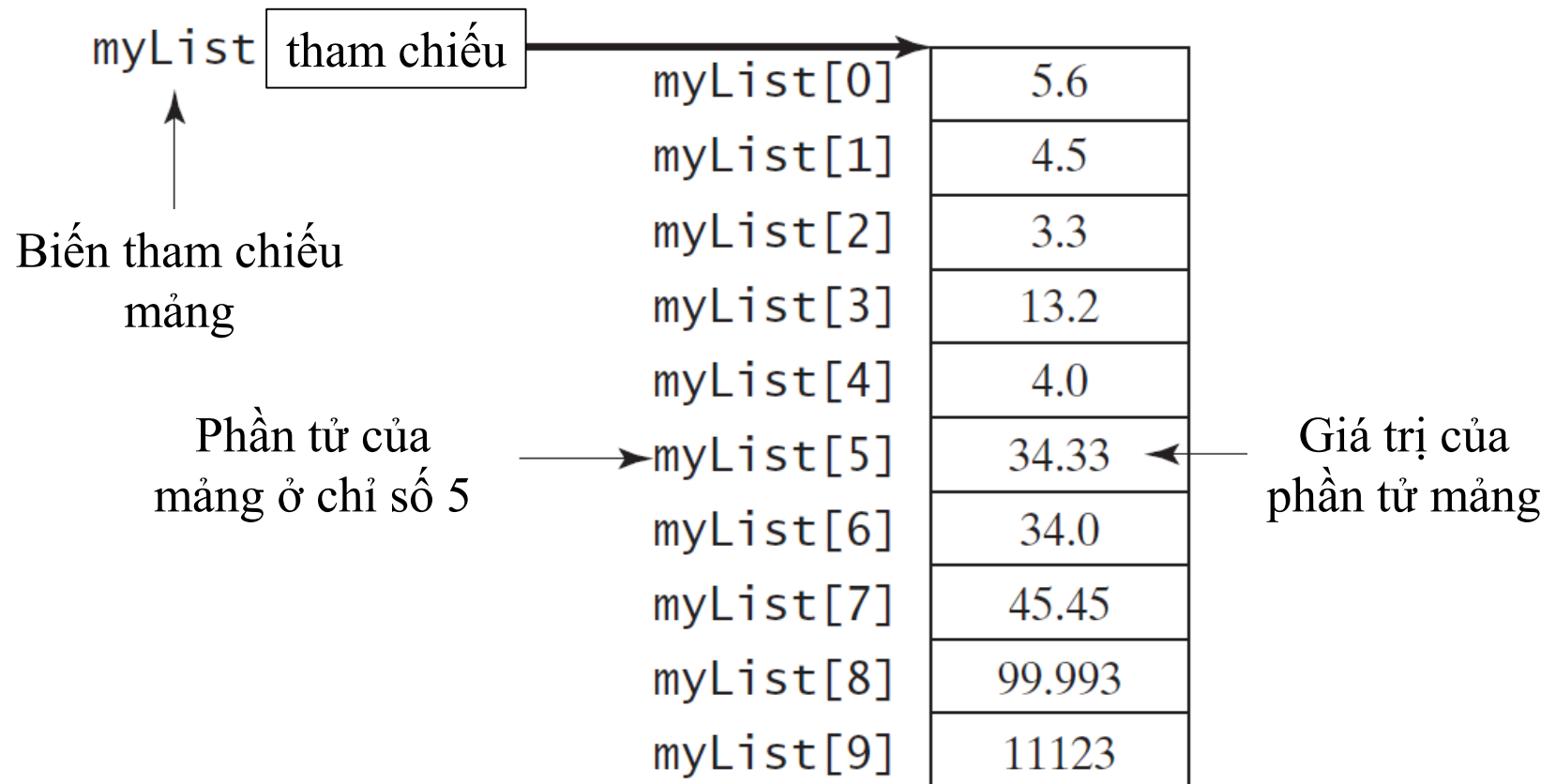
Objectives

- Tìm hiểu tại sao mảng là cần thiết trong lập trình (§7.1).
- Khởi tạo biến tham chiếu và mảng (§§7.2.1–7.2.2).
- Lấy ra giá trị kích thước của mảng với **arrayRefVar.length** (§7.2.3).
- Truy xuất mảng sử dụng các chỉ số (indexes) (§7.2.4).
- Sử dụng các chức năng của mảng (hiển thị, truy cập, tìm giá trị nhỏ nhất, lớn nhất trong mảng (§7.2.6).
- Đơn giản hóa việc lập trình với câu lệnh lặp foreach (§7.2.7).
- Sử dụng mảng để phát triển ứng dụng (**AnalyzeNumbers**, **DeckOfCards**) (§§7.3–7.4).
- Sao chép nội dung giữa các mảng (§7.5).
- Triển khai một số giải thuật tìm kiếm trong mảng (§7.10.1 - §7.10.2).
- Sắp xếp một mảng sử dụng giải thuật sắp xếp lựa chọn (§7.11).
- Làm quen với các phương thức trong lớp **java.util.Arrays** (§7.12).

Introducing Arrays

- Mảng là một cấu trúc dữ liệu biểu diễn một tập hợp của dữ liệu có cùng kiểu
- Mảng có thể được khai báo với các kiểu dữ liệu có sẵn trong Java: *Int*, *Boolean*, *Double* ...

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



Declaring Array Variables

- `datatype[] arrayRefVar;`

Example:

```
double[] myList;
```

- `datatype arrayRefVar[];` // Cách này OK nhưng không khuyến khích

Example:

```
double myList[];
```

Creating Arrays

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

Example:

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

```
myList[0] // Tham chiếu tới phần tử đầu của mảng.
```

```
myList[9] // Tham chiếu tới phần tử cuối của mảng.
```

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

- Khi một mảng được khởi tạo, kích thước của nó là cố định. Chúng ta không thể thay đổi kích thước mảng. Ta có thể kiểm tra kích thước mảng sử dụng phương thức `length`.

`arrayRefVar.length`

- For example,

`myList.length // trả về giá trị 10`

Default Values

- Khi một mảng được khởi tạo, những phần tử của nó được gán một giá trị mặc định, tùy theo kiểu dữ liệu mà mảng đó sử dụng.

0 cho các kiểu dữ liệu dạng số nguyên thủy,

'\u0000' cho kiểu char, và

false cho kiểu boolean.

Indexed Variables

- Các phần tử mảng có thể được truy cập thông qua các chỉ số mảng.
- Chỉ số mảng bắt đầu bằng số 0 và được đếm tới giá trị `arrayRefVar.length-1`.

- Mỗi phần tử trong mảng được biểu diễn dưới cú pháp như sau:

`arrayRefVar[index];`

Cú pháp trên được gọi là một *biến chỉ số (indexed variable)*

Using Indexed Variables

- Sau khi một mảng được khởi tạo, *biến chỉ số* có thể được sử dụng như các chúng ta sử dụng các biến thông thường trong Java.
- Ví dụ, đoạn mã dưới đây tính tổng các giá trị của `myList[0]` và `myList[1]` sau đó lưu vào `myList[2]`.

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

Array Initializers

- Khai báo, tạo, và khởi tạo giá trị trong một bước duy nhất:

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

Cú pháp viết tắt (*shorthand*) này phải nằm trong một câu lệnh!

Using the Shorthand Notation

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

Ký hiệu viết tắt này tương đương với các câu lệnh sau:

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

```
myList[0] = 1.9;
```

```
myList[1] = 2.9;
```

```
myList[2] = 3.4;
```

```
myList[3] = 3.5;
```

CAUTION

- Khi sử dụng ký hiệu viết tắt, chúng ta phải khai báo, tạo, và khởi tạo mảng trong một câu lệnh duy nhất.
- Việc phân tách câu lệnh sẽ gây nên lỗi cú pháp.
- Ví dụ sau minh họa cho lỗi đó

```
double[] myList;
```


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

Trace Program with Arrays

Khai báo các giá trị biến mảng, tạo một mảng và gán tham chiếu của nó cho các giá trị

```
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];  
    }  
}
```

Sau khi mảng được khởi tạo



0	0
1	0
2	0
3	0
4	0

Trace Program with Arrays

i giờ có giá trị là 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];  
    }  
}
```

Sau khi mảng được khởi tạo

0	0
1	0
2	0
3	0
4	0

Trace Program with Arrays

i (=1) bé hơn 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];  
    }  
}
```

Sau khi mảng được khởi tạo

0	0
1	0
2	0
3	0
4	0

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];  
    }  
}
```

Sau khi hoàn thành câu lệnh,
value[1] bằng 1

Sau lần lặp thứ nhất

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

Sau lệnh `i++`, `i` có giá trị 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];  
    }  
}
```

Sau lần lặp thứ nhất

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

i (=2) bé hơn 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];  
    }  
}
```

Sau lần lặp thứ nhất

0	0
1	1
2	0
3	0
4	0

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];  
    }  
}
```

Sau khi hoàn thành câu lệnh,
value[2] bằng 3 (2+1)

Sau lần lặp thứ hai

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

Sau lệnh `i++`, `i` có giá trị 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];  
    }  
}
```

Sau lần lặp thứ hai

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

i (=3) bé hơn 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];  
    }  
}
```

Sau lần lặp thứ hai

0	0
1	1
2	3
3	0
4	0

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];  
    }  
}
```

Sau khi hoàn thành câu lệnh,
value[3] bằng 6 (3+3)

Sau lần lặp thứ ba

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

Sau lệnh `i++`, `i` có giá trị 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];  
    }  
}
```

Sau lần lặp thứ ba

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

i (=4) bé hơn 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];  
    }  
}
```

Sau lần lặp thứ ba

0	0
1	1
2	3
3	6
4	0

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];  
    }  
}
```

Sau khi hoàn thành câu lệnh,
value[4] bằng 10 (4+6)

Sau lần lặp thứ tư

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

Sau lệnh `i++`, `i` có giá trị 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];  
    }  
}
```

Sau lần lặp thứ tư

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

i (=5) bé hơn 5 trở thành false.
Thoát khỏi vòng lặp

```
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];  
    }  
}
```

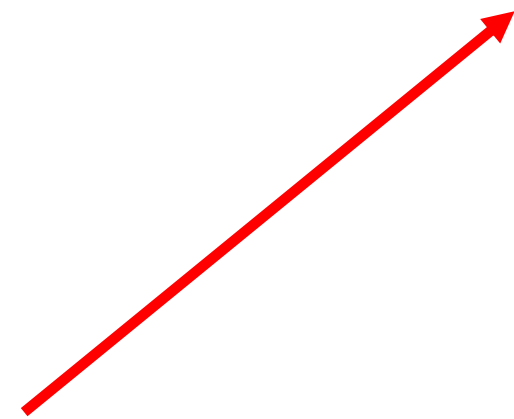
Sau lần lặp thứ tư

0	0
1	1
2	3
3	6
4	10

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];  
    }  
}
```

Sau khi hoàn thành câu lệnh,
value[0] bằng 11 (1+10)



0	11
1	1
2	3
3	6
4	10

Processing Arrays

- ❑ Initializing arrays with input values (Khởi tạo mảng với giá trị đầu vào)
- ❑ Initializing arrays with random values (Khởi tạo giá trị ngẫu nhiên)
- ❑ Printing arrays (In mảng)
- ❑ Summing all elements (Tính tổng các phần tử mảng)
- ❑ Finding the largest element (Tìm phần tử lớn nhất trong mảng)
- ❑ Finding the smallest index of the largest element
- ❑ Random shuffling (Đổi chỗ ngẫu nhiên các phần tử mảng)
- ❑ Shifting elements (Dịch chuyển các phần tử mảng)

Initializing arrays with input values

// Khởi tạo mảng với giá trị nhập vào từ bàn phím

```
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

```
// Khởi tạo mảng với giá trị ngẫu nhiên  
for (int i = 0; i < myList.length; i++) {  
    myList[i] = Math.random() * 100;  
}
```

Printing arrays

```
// In ra màn hình các giá trị của mảng  
for (int i = 0; i < myList.length; i++) {  
    System.out.print(myList[i] + " ");  
}
```


Summing all elements

// Tính tổng của các phần tử trong mảng

```
double total = 0;
```

```
for (int i = 0; i < myList.length; i++) {
```

```
    total += myList[i];
```

```
}
```

Finding the largest element

// Tìm phần tử có giá trị lớn nhất trong mảng

```
double max = myList[0];
```

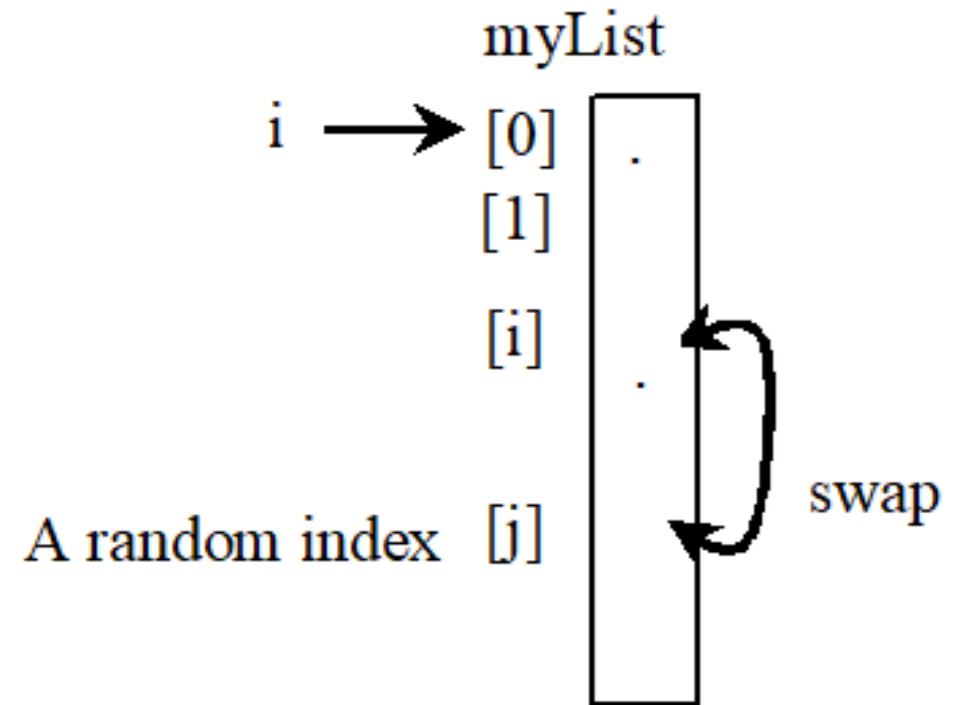
```
for (int i = 1; i < myList.length; i++) {
```

```
    if (myList[i] > max) max = myList[i];
```

```
}
```

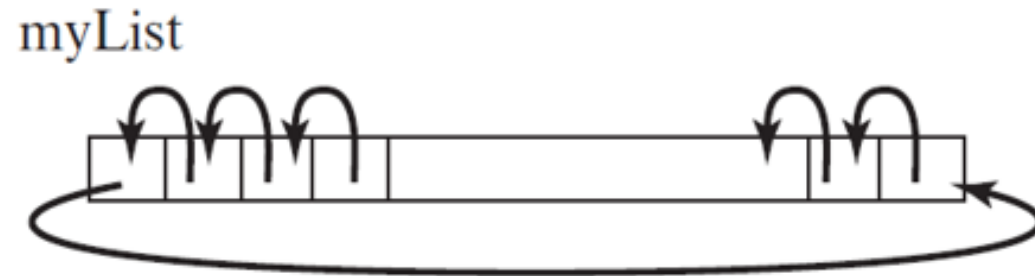
Random shuffling

```
for (int i = 0; i < myList.length - 1; i++) { // Trộn ngẫu nhiên
    // Tạo chỉ số j một cách ngẫu nhiên
    int j = (int)(Math.random()
        * myList.length);
    // Hoán đổi myList[i] với myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```



Shifting Elements

```
double temp = myList[0]; // Giữ lại giá trị phần tử đầu  
// Dịch chuyển các phần tử sang trái  
for (int i = 1; i < myList.length; i++) {  
    myList[i - 1] = myList[i];  
}  
// Di chuyển phần tử đầu để lấp đầy vị trí cuối cùng  
myList[myList.length - 1] = temp;
```



Enhanced for Loop (for-each loop)

JDK 1.5 giới thiệu vòng lặp **for** mới cho phép chúng ta duyệt qua các phần tử mảng mà không cần sử dụng các biến chỉ số. Ví dụ, đoạn code sau xuất ra tất cả các phần tử của mảng `myList`:

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

Cú pháp tổng quát là:

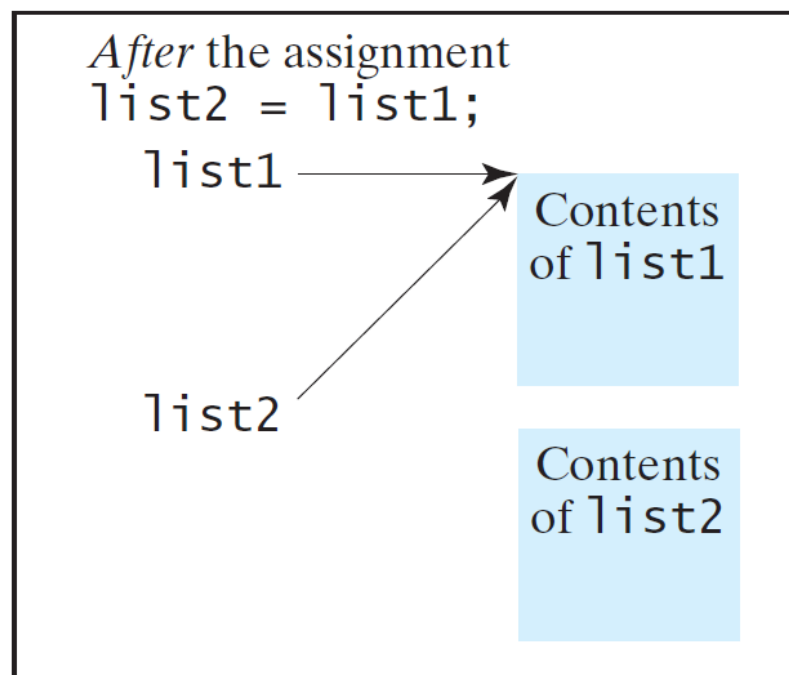
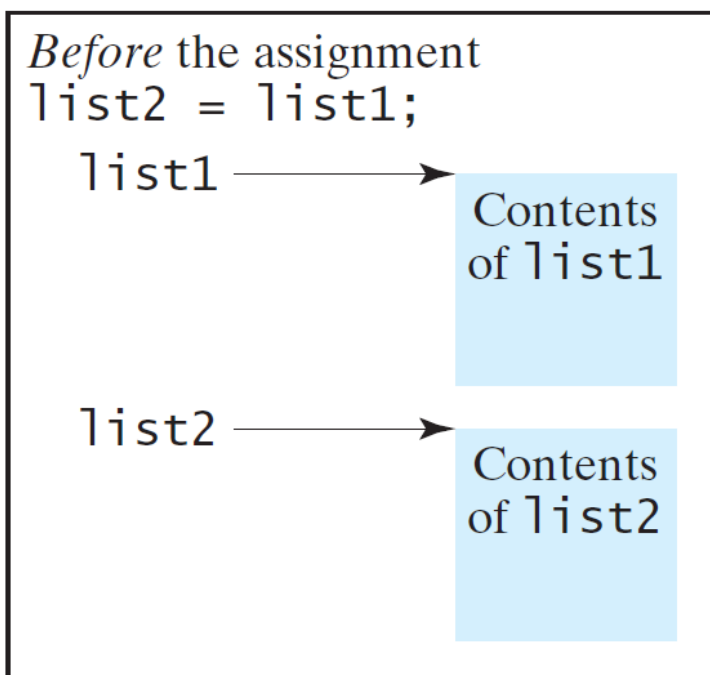
```
for (elementType value: arrayRefVar) {
    // Process the value
}
```

Exercise 1: Analyze Numbers

- Nhập vào một trăm số nguyên, sau đó (1) tính trung bình cộng các số đó, và (2) tìm xem thử bao nhiêu số trong 100 số đó có giá trị lớn hơn trung bình.

Copying Arrays (1/2)

- ❑ Khi lập trình, chúng ta thường có nhu cầu tạo ra bản sao của một mảng hoặc một phần của một mảng. Trong những trường hợp này, chúng ta có thể sử dụng lệnh gán (=): `list2 = list1;`



Copying Arrays (2/2)

❑ Sử dụng một vòng lặp:

```
int[] sourceArray = {2, 3, 1, 5, 10};
```

```
int[] targetArray = new int[sourceArray.length];
```

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


The arraycopy Utility

// Sử dụng tiện ích (phương thức) arraycopy
`arraycopy`(sourceArray, src_pos, targetArray,
tar_pos, length);

Ví dụ:

`System.arraycopy`(Array_Goc, 0, Array_Moi, 0,
Array_Goc.length);

Passing Arrays to Methods

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

```
// Lời gọi phương thức  
int[] list = {3, 1, 2, 6, 4, 2};  
printArray(list);
```

```
// Lời gọi phương thức  
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Mảng ẩn danh

Anonymous Array

Câu lệnh:

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

tạo ra một mảng sử dụng cú pháp sau:

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

Không có biến tham chiếu rõ ràng cho mảng. Mảng như vậy được gọi là mảng ẩn danh (**anonymous array**).

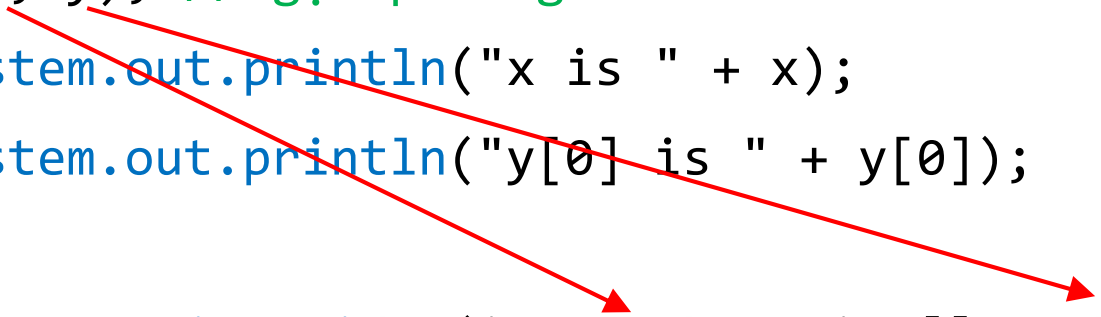
Pass By Value

Java sử dụng truyền theo giá trị để truyền các đối số cho một phương thức. Có sự khác biệt quan trọng giữa việc truyền một giá trị của các biến của kiểu dữ liệu nguyên thủy (primitive) và việc truyền các mảng.

- ❖ Đối với một tham số của một giá trị primitive, giá trị thực được truyền vào. Việc thay đổi giá trị của tham số cục bộ bên trong phương thức **không ảnh hưởng** đến giá trị của biến bên ngoài phương thức.
- ❖ Đối với một tham số kiểu mảng, giá trị của tham số chứa một *tham chiếu đến một mảng*; tham chiếu này được chuyển đến phương thức. Bất kỳ thay đổi nào đối với mảng xảy ra bên trong thân phương thức sẽ ảnh hưởng đến mảng ban đầu được truyền làm đối số.

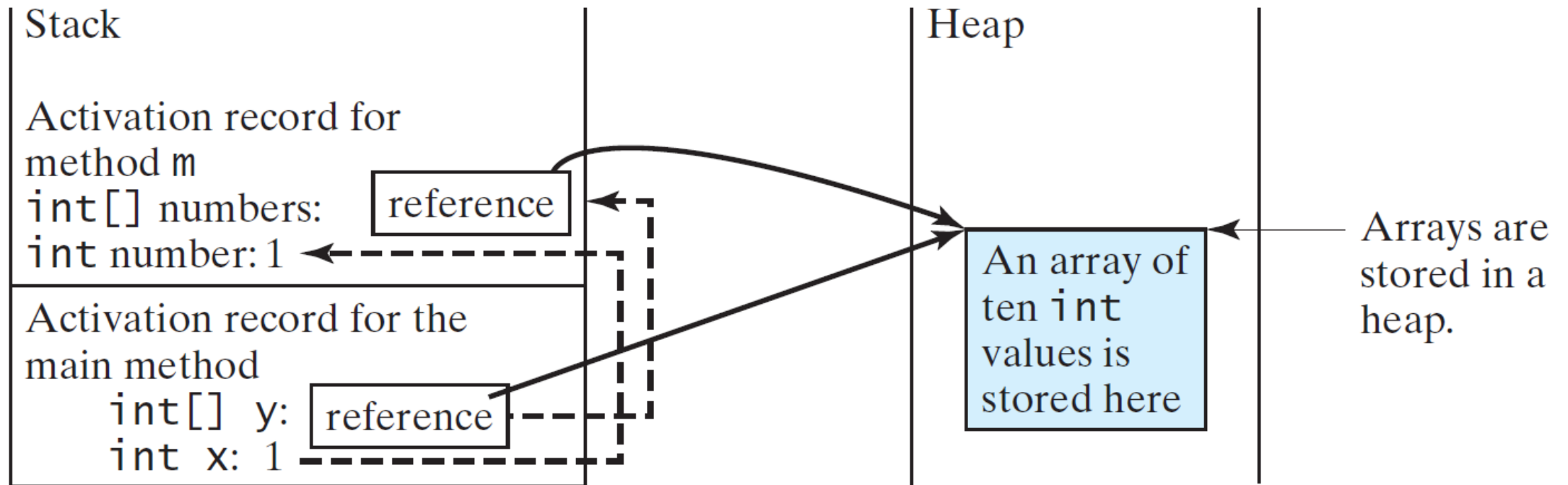
Simple Example

```
public class Test {  
    public static void main(String[] args) {  
        int x = 1; // x được biểu diễn dưới giá trị số nguyên  
        int[] y = new int[10]; // y được biểu diễn dưới dạng một mảng các số nguyên  
        m(x, y); // gọi phương thức m với hai đối số x và 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; // Gán một giá trị mới cho biến number  
        numbers[0] = 5555; // gán một giá trị mới cho biến numbers[0]  
    }  
}
```

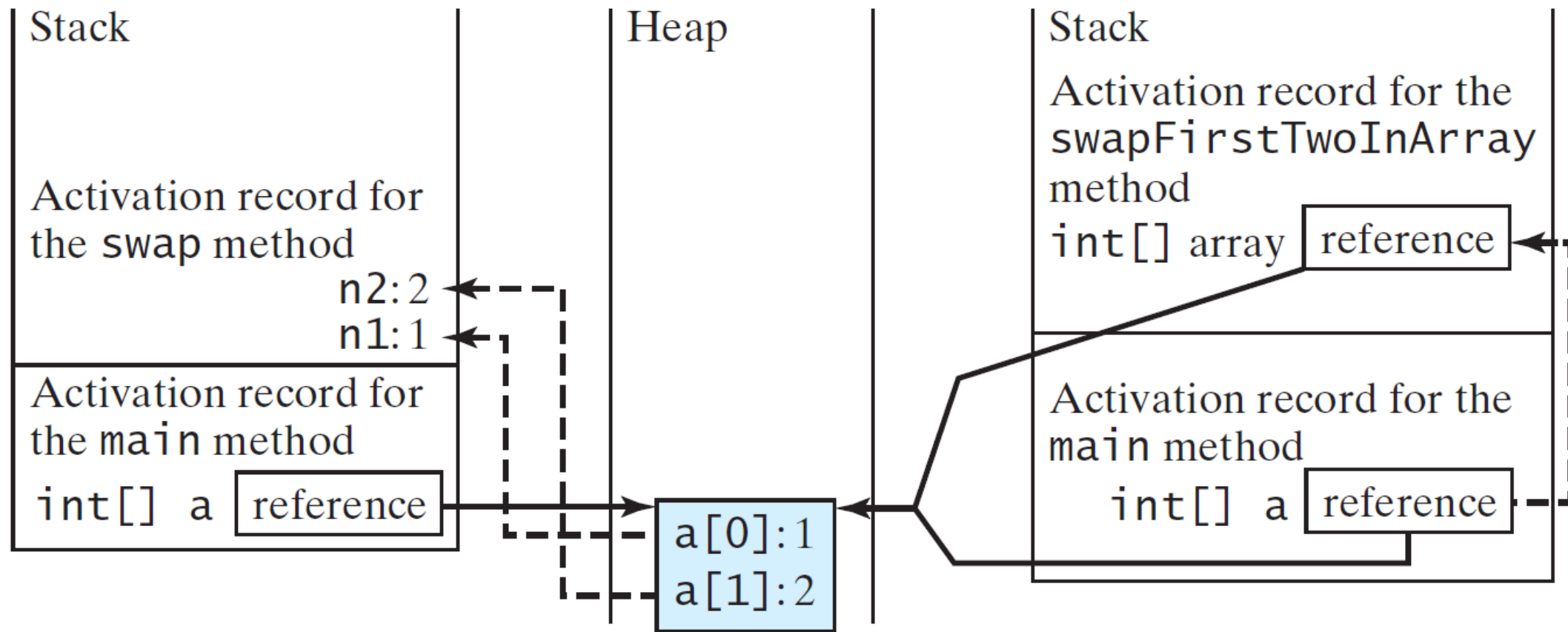
Two red arrows originate from the method calls in the main method. One arrow starts at the 'm' in 'm(x, y);' and points to the 'm' in the method definition 'public static void m(int number, int[] numbers)'. The other arrow starts at the 'y[0]' in 'System.out.println("y[0] is " + y[0]);' and points to the 'numbers' parameter in the same method definition.

Call Stack (1/3)

- Khi gọi (invoke) phương thức $m(x, y)$, giá trị của x và y được truyền cho biến **number** và **numbers**. Vì y là giá trị tham chiếu tới mảng, mảng **numbers** giờ chứa cùng tham chiếu giá trị tới cùng mảng.

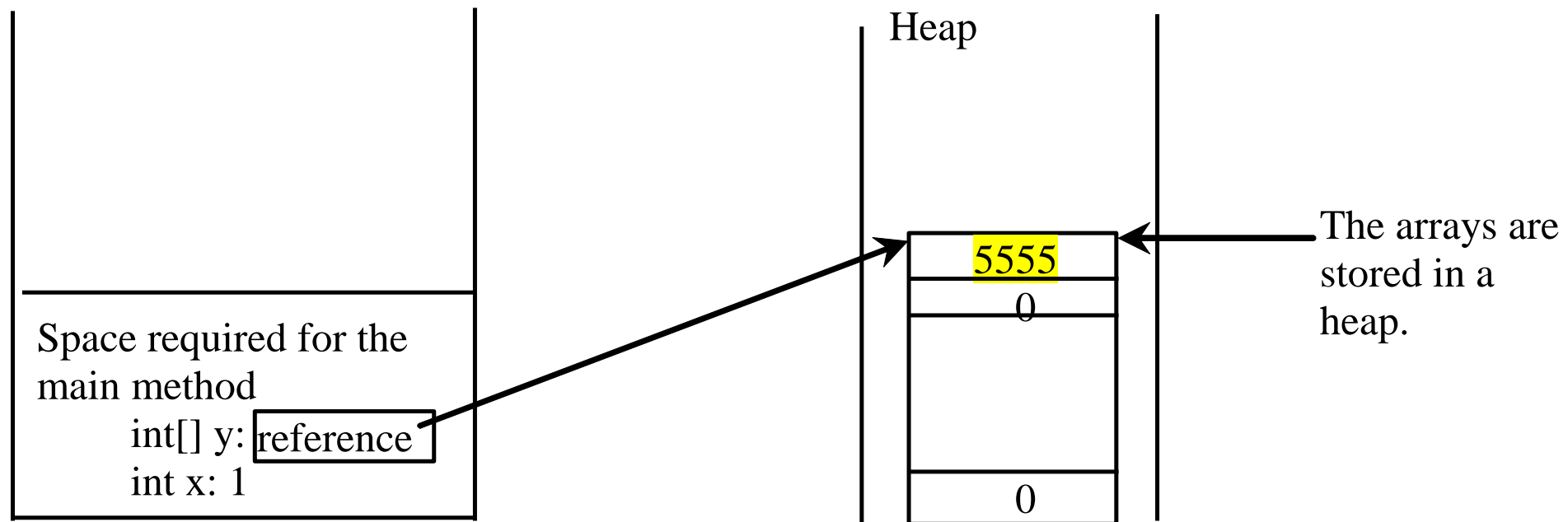


Call Stack (2/3)



Call Stack – Heap (3/3)

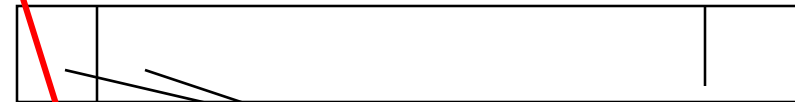
- JVM trong vùng nhớ được gọi là *heap*. Vùng nhớ này cho phép cấp phát bộ nhớ động trong đó các khối bộ nhớ được cấp phát và giải phóng theo thứ tự tùy ý.



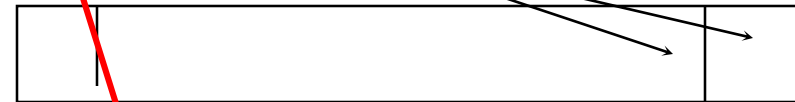
Returning an Array from a Method

```
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;  
}
```

list



result



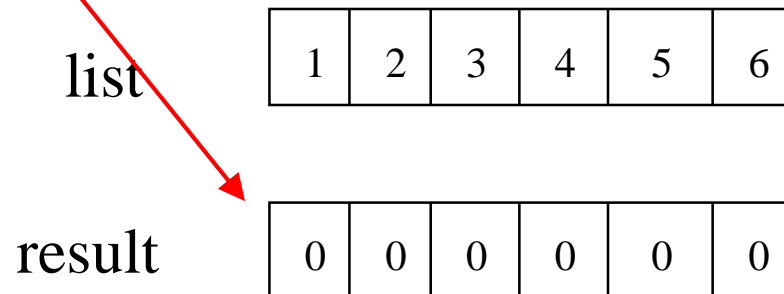
```
int[] list1 = {1, 2, 3, 4, 5, 6};  
int[] list2 = reverse(list1);
```

Trace the reverse Method

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

Khai báo biến result và tạo mảng

```
public static int[] reverse(int[] list) {  
    int[] result = new int[list.length];  
    for (int i = 0, j = result.length - 1;  
         i < list.length; i++, j--) {  
        result[j] = list[i];  
    }  
    return result;  
}
```

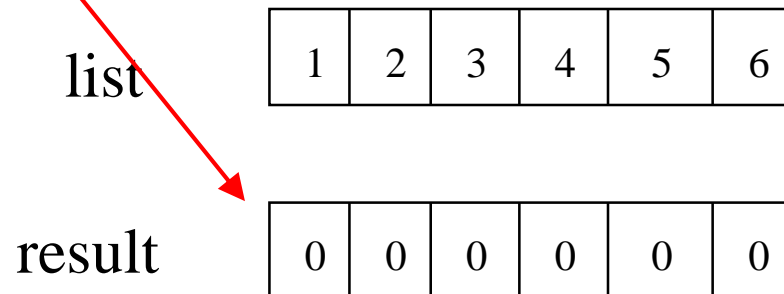


Trace the reverse Method

```
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 và j = 5

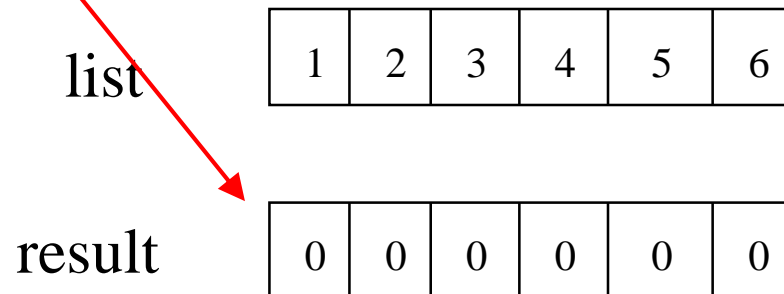


Trace the reverse Method

```
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) bé hơn 6

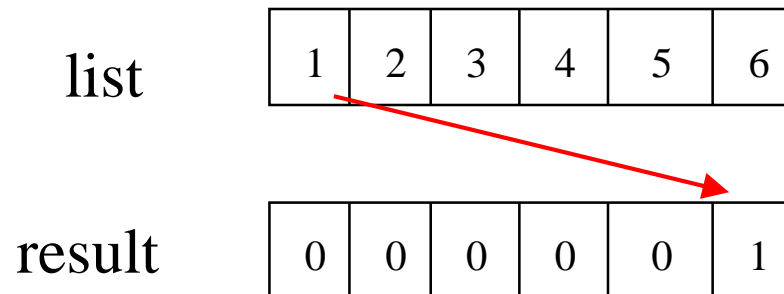


Trace the reverse Method

```
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 và j = 5
Gán list[0] vào result[5]



Trace the reverse 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;  
}
```

Sau đó, i = 1 và j = 4

list

1	2	3	4	5	6
---	---	---	---	---	---

result

0	0	0	0	0	1
---	---	---	---	---	---

Trace the reverse 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) bé hơn 6

list

1	2	3	4	5	6
---	---	---	---	---	---

result

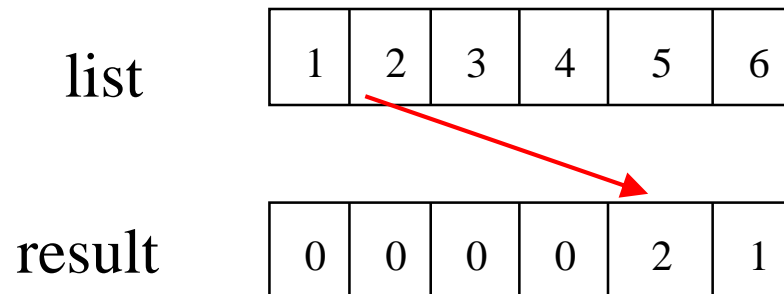
0	0	0	0	0	1
---	---	---	---	---	---

Trace the reverse 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 và j = 4
Gán list[1] vào result[4]



Trace the reverse 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;  
}
```

Sau đó, $i = 2$ và $j = 3$

list

1	2	3	4	5	6
---	---	---	---	---	---

result

0	0	0	0	2	1
---	---	---	---	---	---

Trace the reverse 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) bé hơn 6

list

1	2	3	4	5	6
---	---	---	---	---	---

result

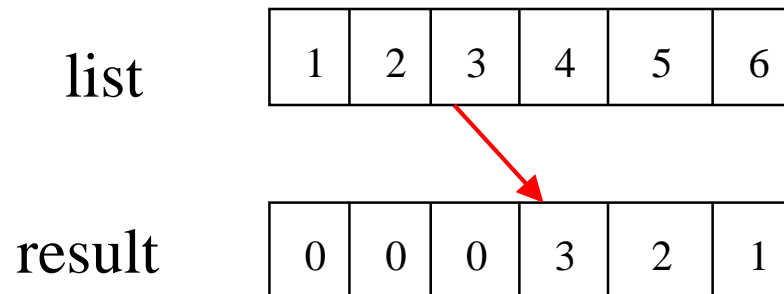
0	0	0	0	2	1
---	---	---	---	---	---

Trace the reverse 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 và j = 3
Gán list[2] vào result[3]



Trace the reverse 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;  
}
```

Sau đó, $i = 3$ và $j = 2$

list

1	2	3	4	5	6
---	---	---	---	---	---

result

0	0	0	3	2	1
---	---	---	---	---	---

Trace the reverse 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) bé hơn 6

list

1	2	3	4	5	6
---	---	---	---	---	---

result

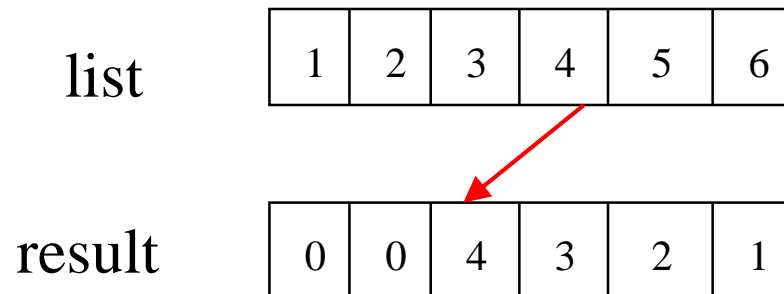
0	0	0	3	2	1
---	---	---	---	---	---

Trace the reverse 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 và j = 2
Gán list[3] vào result[2]



Trace the reverse 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;  
}
```

Sau đó, $i = 4$ và $j = 1$

list

1	2	3	4	5	6
---	---	---	---	---	---

result

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

Trace the reverse 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) bé hơn 6

list

1	2	3	4	5	6
---	---	---	---	---	---

result

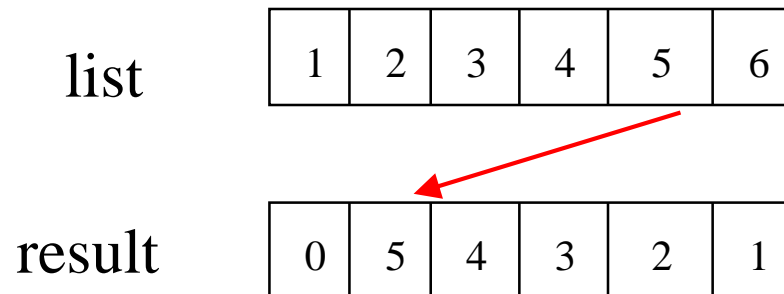
0	0	4	3	2	1
---	---	---	---	---	---

Trace the reverse 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 và j = 1
Gán list[4] vào result[1]



Trace the reverse 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;  
}
```

Sau đó, $i = 5$ và $j = 0$

list

1	2	3	4	5	6
---	---	---	---	---	---

result

0	5	4	3	2	1
---	---	---	---	---	---

Trace the reverse 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) vẫn bé hơn 6

list

1	2	3	4	5	6
---	---	---	---	---	---

result

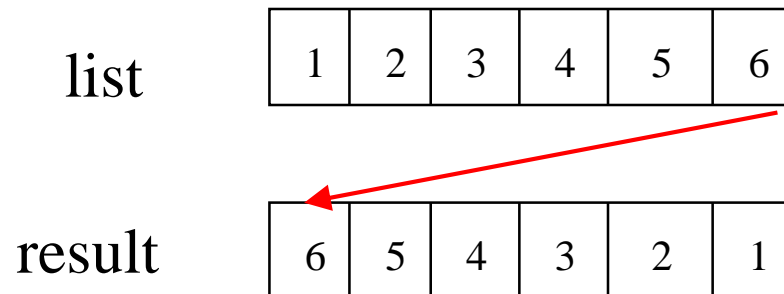
0	5	4	3	2	1
---	---	---	---	---	---

Trace the reverse 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 và j = 0
Gán list[5] vào result[0]



Trace the reverse 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;  
}
```

Sau đó, $i = 6$ và $j = -1$

list

1	2	3	4	5	6
---	---	---	---	---	---

result

6	5	4	3	2	1
---	---	---	---	---	---

Trace the reverse Method

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

Điều kiện $i (=6) < 6$ trở thành
false. Thoát khỏi vòng lặp

```
public static int[] reverse(int[] list) {  
    int[] result = new int[list.length];  
    for (int i = 0, j = result.length - 1;  
        i < list.length; i++, j--) {  
        result[j] = list[i];  
    }  
    return result;  
}
```

list

1	2	3	4	5	6
---	---	---	---	---	---

result

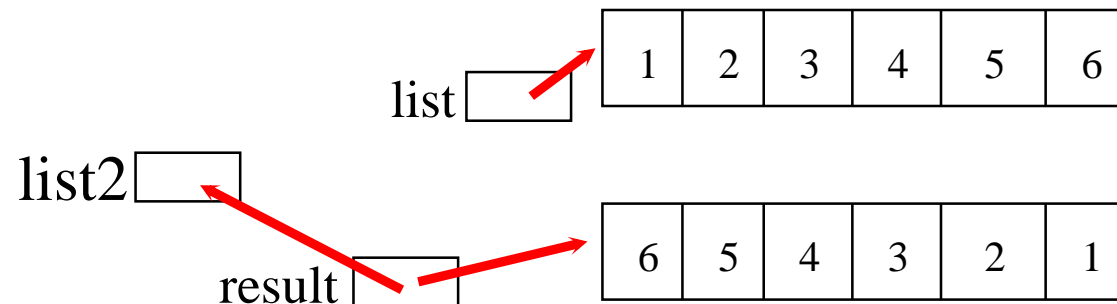
6	5	4	3	2	1
---	---	---	---	---	---

Trace the reverse 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;  
}
```

Trả về result



Searching Arrays

- Tìm kiếm là quá trình *tìm kiếm một phần tử cụ thể trong một mảng*; Tìm kiếm là một công việc phổ biến trong lập trình. Có rất nhiều thuật toán và cấu trúc dữ liệu dành cho việc tìm kiếm.
- Trong phần này, chúng ta sẽ học hai phương pháp tìm kiếm cơ bản là: *tìm kiếm tuyến tính* (linear search) và *tìm kiếm nhị phân* (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;  
    }  
}
```

[0] [1] [2] ...
list

--	--	--	--	--	--

key Compare key with list[i] for i = 0, 1, ...

Linear Search

- Phương pháp *tìm kiếm tuyến tính* so sánh tuần tự phần tử khóa **key** với từng phần tử trong danh sách mảng.
- Phương thức tiếp tục thực hiện cho đến khi khóa **key** khớp với một phần tử trong danh sách hoặc danh sách đã hết mà không tìm thấy kết quả phù hợp.
- Nếu tìm thấy phần tử, tìm kiếm tuyến tính trả về chỉ số của phần tử trong mảng khớp với khóa. Nếu không tìm thấy kết quả phù hợp nào, tìm kiếm sẽ trả về **-1**.

Linear Search Animation

Key	List
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8
3	6 4 1 9 7 3 2 8

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;  
}
```

Sử dụng phương thức ở trên bằng lời gọi (invoke)

```
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 (1/7)

- Trong tìm kiếm nhị phân, các phần tử trong mảng phải được sắp xếp theo thứ tự. Giả sử rằng mảng theo thứ tự tăng dần.

Ví dụ: **2 4 7 10 11 45 50 59 60 66 69 70 79**

- Đầu tiên, tìm kiếm nhị phân so sánh khóa với phần tử ở ***giữa*** mảng.

Binary Search (2/7)

Chúng ta xem xét ba trường hợp sau:

- ❖ Nếu khóa nhỏ hơn phần tử ở giữa, ta chỉ cần tìm kiếm khóa trong *nửa đầu* của mảng.
- ❖ Nếu khóa bằng với phần tử ở giữa, tìm kiếm kết thúc bằng một *kết quả khớp*.
- ❖ Nếu khóa lớn hơn phần tử ở giữa, bạn chỉ cần tìm kiếm khóa trong *nửa sau của mảng*.

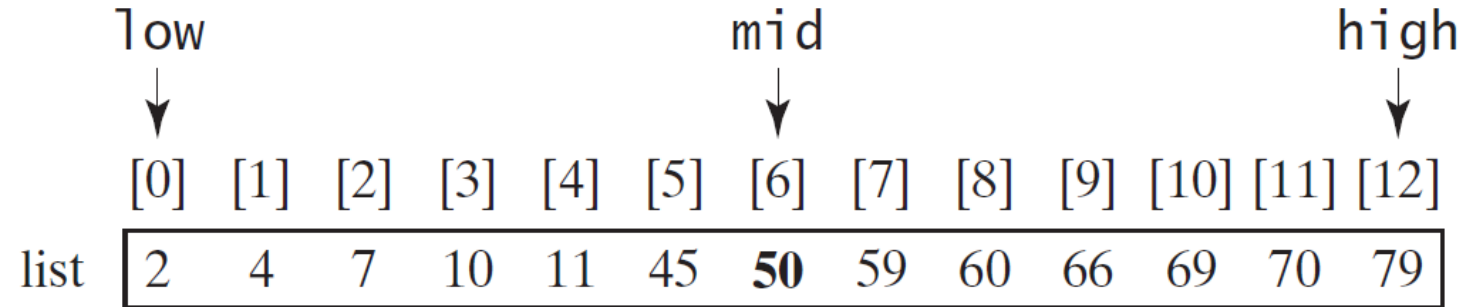
Binary Search (3/7)

Key	List								
8	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	6	7	8	9
1	2	3	4	6	7	8	9		
8	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	6	7	8	9
1	2	3	4	6	7	8	9		
8	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	6	7	8	9
1	2	3	4	6	7	8	9		

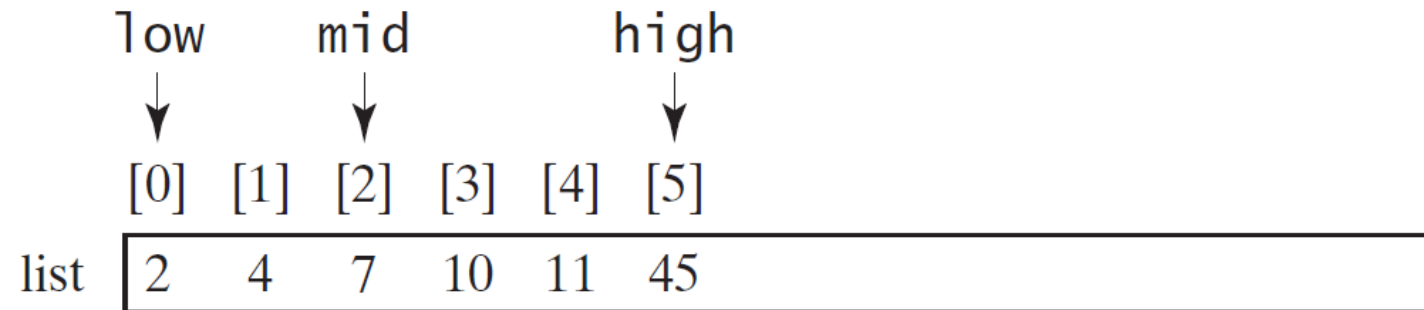
Binary Search (4/7)

key is 11

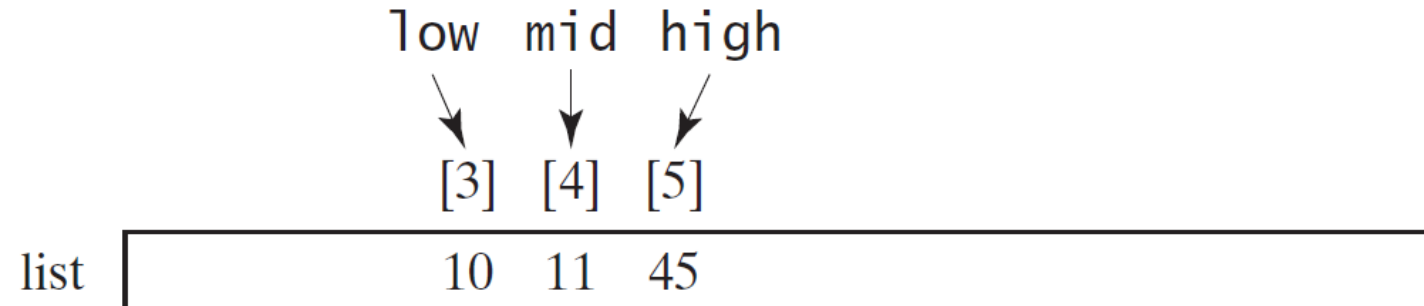
key < 50



key > 7



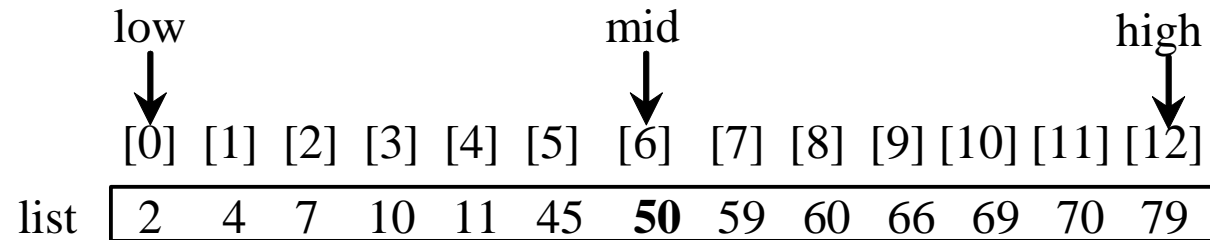
key == 11



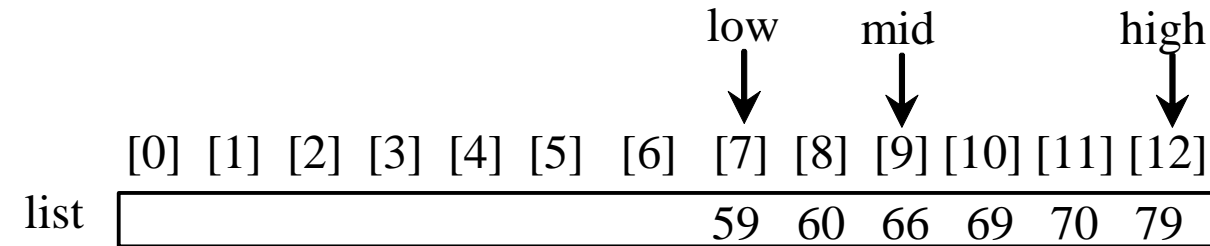
Binary Search (5/7)

key is 54

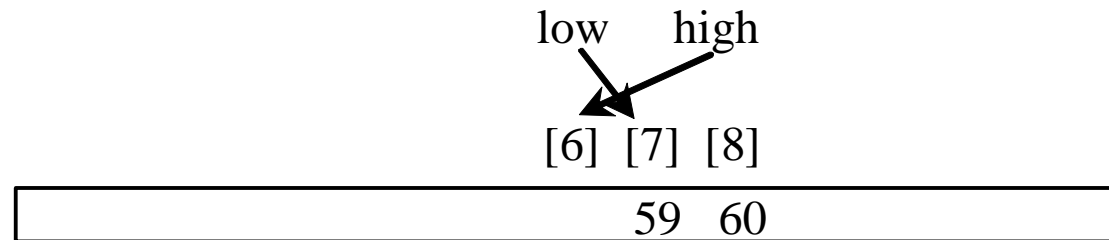
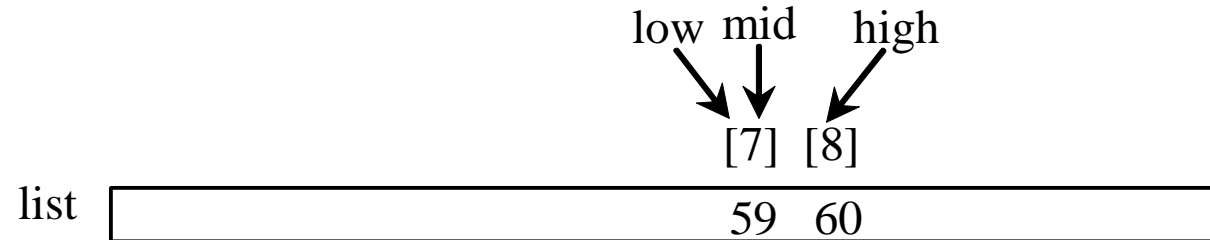
key > 50



key < 66



key < 59



Binary Search (6/7)

- Phương thức *binarySearch* trả về chỉ mục của phần tử trong danh sách khớp với khóa tìm kiếm nếu nó được chứa trong danh sách. Nếu không, nó trả về :
 - **insertion point - 1**
- *insertion point* (điểm chèn) là điểm mà khóa sẽ được chèn vào danh sách.

Binary Search (7/7)

```
/** 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

- Vì tìm kiếm nhị phân thường được sử dụng trong lập trình, Java đã cung cấp sẵn một số phương thức tìm kiếm nhị phân được nạp chồng để tìm kiếm khóa trong một mảng **int**, **double**, **char**, **short**, **long** và **float** trong lớp **java.util.Arrays**. Ví dụ, đoạn mã sau tìm kiếm các khóa trong một mảng số và một mảng ký tự.

```
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));
```

Trả về 4

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

Trả về -4 (insertion point = 3,
vì vậy trả về -3-1)

Để phương thức tìm kiếm nhị phân hoạt động, mảng phải được sắp xếp trước theo **thứ tự tăng dần**.

Sorting Arrays

- Sắp xếp, giống như tìm kiếm, cũng là một nhiệm vụ phổ biến trong lập trình máy tính.
- Nhiều thuật toán khác nhau đã được phát triển để phân loại.
- Sắp xếp (Sorting) là quá trình tổ chức lại các dữ liệu theo thứ tự giảm dần hoặc tăng dần.
- Phần này giới thiệu một thuật toán sắp xếp đơn giản, trực quan: sắp xếp lựa chọn (selection sort).

Selection Sort

Sắp xếp lựa chọn - selection sort

Thuật toán lặp gồm đúng $i = 1, 2, \dots, n - 1$ vòng lặp

- 1 Tìm phần tử nhỏ nhất đưa vào vị trí 1
- 2 Tìm phần tử nhỏ thứ hai đưa vào vị trí 2
- 3 Tìm phần tử nhỏ thứ ba đưa vào vị trí 3

- Giải thuật *selection sort* sắp xếp một danh sách các giá trị bằng cách lặp lại việc đặt một giá trị cụ thể vào đúng vị trí thích hợp cho nó trong dãy sắp xếp.

Selection Sort

❑ Minh họa thuật toán
sắp xếp với dãy khóa:
 $k = \{50, 45, 69, 12, 15, 89, 24, 11, 95, 57, 30\}$

Lượt	Khóa	50	45	69	12	15	89	24	11	95	57	30
1	11	11	45	69	12	15	89	24	50	95	57	30
2	12	-	12	69	45	15	89	24	50	95	57	30
3	15	-	-	15	45	69	89	24	50	95	57	30
4	24	-	-	-	24	69	89	45	50	95	57	30
5	30	-	-	-	-	30	89	45	50	95	57	69
6	45	-	-	-	-	-	45	89	50	95	57	69
7	50	-	-	-	-	-	-	50	89	95	57	69
8	57	-	-	-	-	-	-	-	57	95	89	69
9	69	-	-	-	-	-	-	-	-	69	89	95
10	89	-	-	-	-	-	-	-	-	-	89	95

Selection Sort

/** Phương thức sắp xếp lựa chọn */

```
public static void selectionSort(double[] list) {  
    for (int i = 0; i < list.length; i++) {  
        // Tìm số nhỏ nhất trong mảng 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;  
            }  
        }  
  
        // Đổi chỗ list[i] với list[currentMinIndex] nếu cần thiết  
        if (currentMinIndex != i) {  
            list[currentMinIndex] = list[i];  
            list[i] = currentMin;  
        }  
    }  
}
```

- Đối với phương pháp sắp xếp lựa chọn, có thể coi phép so sánh ($\text{currentMin} > \text{list}[j]$) là phép toán tích cực để đánh giá hiệu suất thuật toán về mặt thời gian.
- Vậy thuật toán sắp xếp kiểu chọn có độ phức tạp tính toán là $\Theta(n^2)$.

The Arrays.sort Method

- Vì sắp xếp thường được sử dụng trong lập trình, Java cung cấp một số phương thức sắp xếp quá tải để sắp xếp một mảng int, double, char, short, long và float trong lớp **java.util.Arrays**. Ví dụ, đoạn mã sau đây sắp xếp một mảng số và một mảng ký tự.

```
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 đã cung cấp phương thức `Arrays.parallelSort(list)` nhằm tận dụng việc xử lý đa luồng của CPU, từ đó tăng tốc việc sắp xếp.

Q&A



Bài tập ôn tập lý thuyết mảng 1 chiều

- 1) Hãy viết chương trình java cho phép nhập vào một *mảng số nguyên* có n phần tử và thực hiện các công việc sau:
 - Xuất giá trị các phần tử của mảng.
 - Tìm phần tử có giá trị lớn nhất, nhỏ nhất.
 - Đếm số phần tử là số chẵn.
 - Sắp xếp mảng tăng dần.
- 2) Viết chương trình nhập một mảng số nguyên $a_0, a_1, a_2, \dots, a_{n-1}$. Liệt kê số lần xuất hiện của các phần tử trong một mảng đã cho.