



Bài 1

Ngôn ngữ lập trình Java

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT

- Trình bày được nội dung, yêu cầu, lịch trình và kết quả của môn học BC-JAVA
- Sử dụng được cú pháp Java để thao tác với biến
- Sử dụng được cú pháp Java để thao tác với cấu trúc điều kiện
- Sử dụng được cú pháp Java để thao tác với cấu trúc lặp
- Mô tả được cú pháp khai báo và sử dụng mảng
- Sử dụng được cú pháp Java để thao tác với phương thức



Module Bootcamp Web Backend Development

Module Bootcamp-Java Web Backend Development



- **Mục đích:** Giúp học viên làm chủ các kiến thức lập trình cơ bản và tư duy giải quyết vấn đề. Hoàn thành khoá học, học viên có đủ kiến thức và kỹ năng nền tảng về lập trình để bước sang giai đoạn học lập trình chuyên sâu.
- **Thời gian:** 25 bài
- **Đánh giá:**
 - Thi thực hành và lý thuyết cuối module, điểm đạt: 75%
 - Bảng đánh giá kỹ năng theo chuẩn đầu ra
- **Yêu cầu:**
 - Phần mềm IntelliJ

Module Bootcamp Preparation



- **Tài liệu học tập:**

- CodeGymX: Bootcamp Preparation
- Source code mẫu trên kênh Github của CodeGym
- Ứng dụng CodeGym Bob gồm các bài luyện tập, bài học, bài kiểm tra.
- Các tài liệu tham chiếu bên ngoài

- **Tài liệu tham khảo:**

- Khoá học Javascript căn bản trên [Codecademy](#)
- Khoá học Javascript căn bản trên [Khanacademy](#)



Thảo luận

Ngôn ngữ lập trình Java
IntelliJ IDEA

Ngôn ngữ Lập trình Java

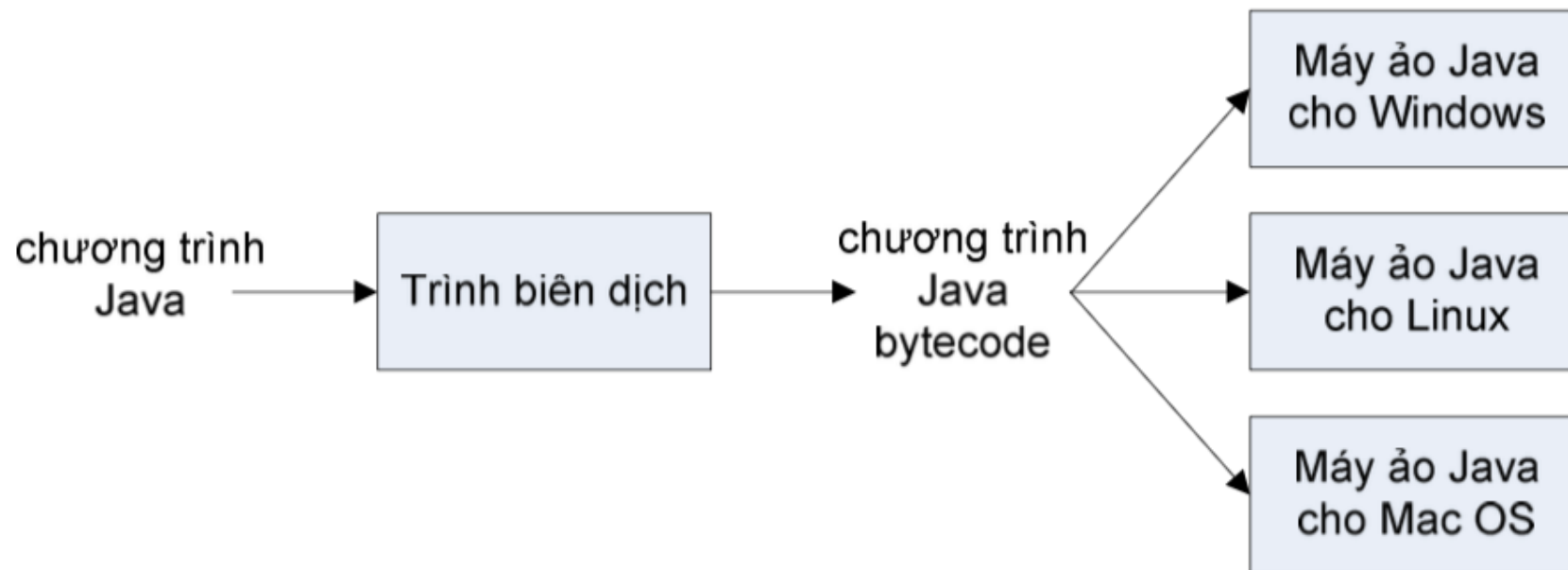


- Là một ngôn ngữ lập trình hướng đối tượng
- Có khả năng thực thi ở nhiều loại thiết bị
- Được sử dụng rộng rãi

Write One, Run anywhere



- Java có tính độc lập nền tảng (platform independent)
- Một chương trình Java có thể chạy trên các nền tảng khác nhau mà không phải biên dịch lại





Máy ảo Java (JVM) và byte code

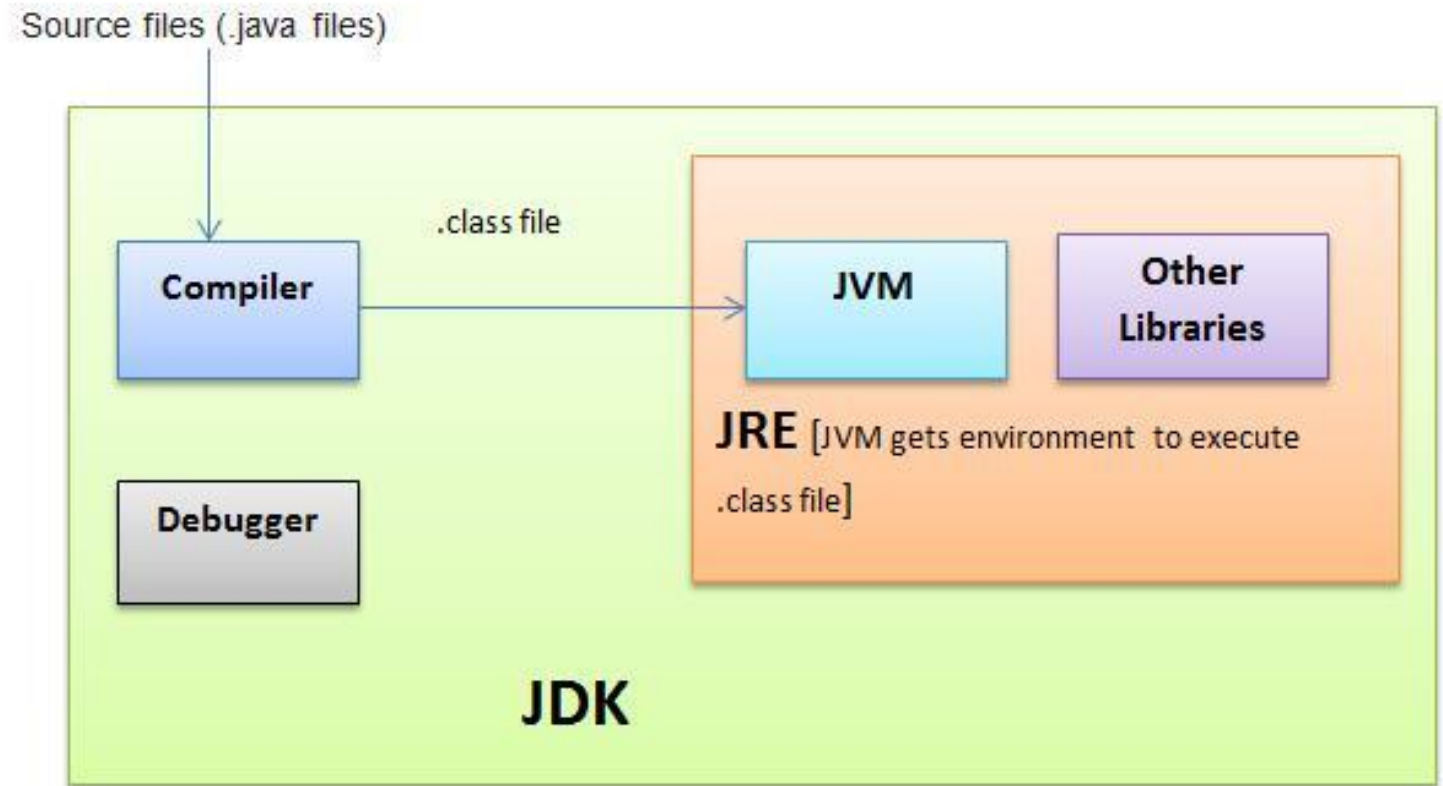
- Một chương trình viết bằng ngôn ngữ bậc cao cần được dịch sang ngôn ngữ máy trước khi có thể được chạy trên máy tính
- Mã nguồn Java không được dịch trực tiếp ra ngôn ngữ máy mà được biên dịch (*compile*) ra byte code
- byte code là ngôn ngữ được thực thi trên một máy tính ảo gọi là JVM (Java Virtual Machine)
- Khi một chương trình Java được thực thi thì JVM sẽ thông dịch (*interpret*) ra ngôn ngữ máy thực sự

JDK vs JRE

JRE giúp thực thi các chương trình Java trong JVM

JDK giúp phát triển và biên dịch mã Java thành chương trình Java

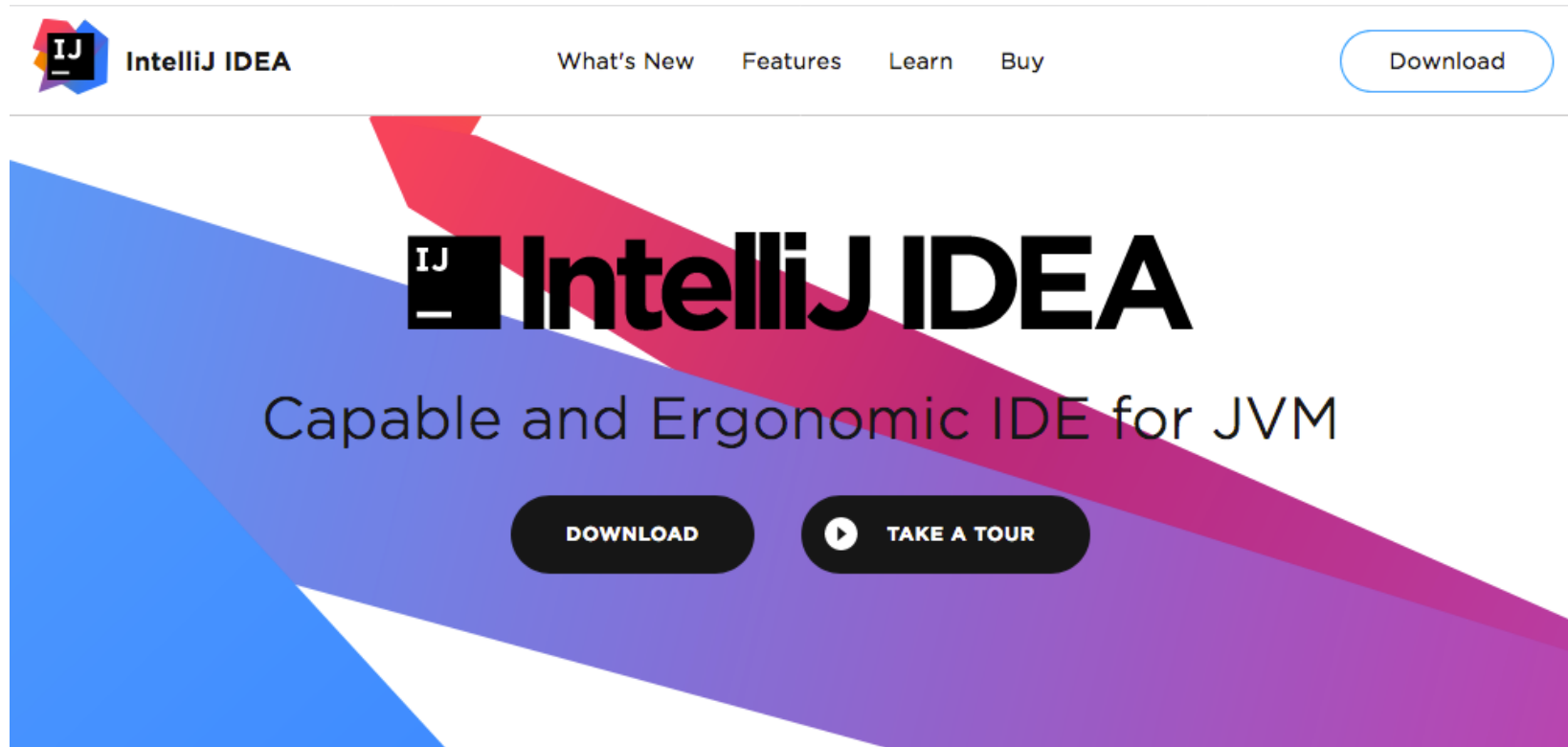
JRE cũng được kèm theo trong JDK



IntelliJ IDEA



- Tải IntelliJ IDE tại: <https://www.jetbrains.com/idea/>



Demo: Tạo ứng dụng Java - 1



- Bước 1: Tạo project mới trên IntelliJ IDEA đặt tên helloworld
- Bước 2: Tạo lớp HelloWorld với nội dung:

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello world");  
    }  
}
```

Có thể sử dụng các cách gõ tắt sau:

- psvm + TAB: viết nhanh hàm main()
- sout + TAB: viết nhanh hàm System.out.println()

Demo: Tạo ứng dụng Java - 2

- Bước 3: Chạy ứng dụng: Chọn mục "Run 'HelloWorld.main()'".



Có thể sử dụng phím tắt để chạy ứng dụng.



Demo

Tạo ứng dụng Java đầu tiên

Thảo luận

Biến và hằng

Kiểu dữ liệu

Toán tử

Khai báo biến



- Java yêu cầu phải khai báo biến trước khi sử dụng
- Cú pháp:

datatype variableName;

Trong đó:

- datatype là kiểu dữ liệu của biến
 - variableName là định danh (tên) của biến
- Có thể khai báo nhiều biến cùng kiểu giá trị trong một câu lệnh:

datatype variable1, variable2, ..., variablen;

- Ví dụ:

int i, j, k; //Khai báo các biến i, j, k là kiểu số nguyên



Gán giá trị cho biến

- Có thể gán giá trị cho biến ngay tại thời điểm khai báo
- Ví dụ:

```
int count = 1;
```

- Ví dụ trên tương đương với:

```
int count;  
count = 1;
```

- Có thể gán giá trị cho nhiều biến tại thời điểm khai báo
- Ví dụ:

```
int i = 1, j = 2;
```

Hằng (constant)



- Hằng là một tên gọi đại diện cho một giá trị cố định
- Giá trị của hằng không thể thay đổi
- Giá trị của hằng cần phải được gán tại thời điểm khai báo
- Ví dụ, sử dụng hằng PI thay cho giá trị 3.14159:

```
double area = radius * radius * 3.14159;
```

Được thay bằng:

```
final double PI = 3.14159;  
double area = radius * radius * PI;
```

Khai báo hằng



- Cú pháp khai báo hằng:

final datatype CONSTANTNAME = value;

Trong đó:

- *final* là từ khoá bắt buộc để khai báo hằng
- *datatype* là kiểu dữ liệu của hằng
- *CONSTANTNAME* là tên của hằng
- *value* là giá trị của hằng

8 kiểu dữ liệu nguyên thuỷ (primitive datatype)



Kiểu dữ liệu	Mô tả
byte	Kiểu số nguyên có kích thước 1 byte. Các giá trị nằm trong khoảng -128 đến 127.
short	Kiểu số nguyên có kích thước 2 byte. Các giá trị nằm trong khoảng -32768 đến 32767.
int	Kiểu số nguyên có kích thước 4 byte. Các giá trị nằm trong khoảng -2^{31} đến $2^{31} - 1$.
long	Kiểu số nguyên có kích thước 8 byte. Các giá trị nằm trong khoảng -2^{63} đến $2^{63} - 1$.
float	Kiểu số thực có kích thước 4 byte.
double	Kiểu số thực có kích thước 8 byte.
boolean	Bao gồm 2 giá trị là true và false.
char	Kiểu ký tự Unicode có kích thước 2 byte. Có giá trị nhỏ nhất là '\u0000' (tương đương với 0) và giá trị lớn nhất là '\uffff' (tương đương với 65535)

Giá trị mặc định



- Khi khai báo một biến của đối tượng (không phải biến địa phương) mà không gán giá trị cho nó thì nó sẽ có giá trị mặc định

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Toán tử số học



Toán tử	Mô tả	Ví dụ
+	Phép cộng	<code>int result = 1 + 2; //result = 3</code>
-	Phép trừ	<code>int result = 1 - 2; //result = -1</code>
*	Phép nhân	<code>int result = 2 * 3; //result = 6</code>
/	Phép chia	<code>int result = 3 / 2; //result = 1</code>
%	Phép chia lấy số dư	<code>int result = 5 % 3; //result = 2</code>

Toán tử một ngôi



Toán tử	Mô tả	Ví dụ
+	Toán tử cộng.	<i>int result = +1; //result = 1</i>
-	Toán tử trừ	<i>int result = -1; //result = -1</i>
++	Toán tử tăng 1 giá trị	<i>int result = 1;</i> <i>int result++; //result = 2</i>
--	Toán tử giảm 1 giá trị	<i>int result = 1;</i> <i>int result--; //result = 0</i>
!	Toán tử phủ định	<i>int value = true;</i> <i>result = !value; //result = false</i>

Toán tử tăng và giảm

- Toán tử tăng (++) và giảm (--) sẽ cho kết quả khác nhau, tùy thuộc vào vị trí của nó so với toán hạng
- Nếu đặt trước (prefix) toán hạng thì giá trị sẽ được tăng hoặc giảm trước khi biểu thức được đánh giá
- Nếu đặt sau (postfix) toán hạng thì giá trị sẽ được tăng hoặc giảm sau khi biểu thức được đánh giá
- Ví dụ:

```
int i = 3;  
int j = ++i; // i = 4 và j = 4;
```

```
int i = 3;  
int j = i++; // i = 4 và j = 3;
```




Toán tử so sánh (Comparission)

Toán tử	Mô tả	Ví dụ <code>int a = 5;</code> <code>int b = 6;</code>
<code>==</code>	So sánh bằng	<code>boolean result = a == b; //result = false</code>
<code>!=</code>	So sánh khác	<code>boolean result = a != b; //result = true</code>
<code>></code>	Lớn hơn	<code>boolean result = a > b; //result = false</code>
<code>>=</code>	Lớn hơn hoặc bằng	<code>boolean result = a >= b; //result = false</code>
<code><</code>	Nhỏ hơn	<code>boolean result = a < b; //result = true</code>
<code><=</code>	Nhỏ hơn hoặc bằng	<code>boolean result = a <= b; //result = true</code>

Toán tử logic



Toán tử	Mô tả	Ví dụ <code>int a = true;</code> <code>int b = false;</code>
<code>&&</code>	AND (và): Trả về đúng nếu cả 2 vế đều đúng	<code>boolean result = a && b; //result = false</code>
<code> </code>	Trả về đúng nếu ít nhất một vế đúng	<code>boolean result = a b; //result = true</code>
<code>!</code>	Phủ định	<code>boolean result = !a; //result = false</code>



Demo

Biến, toán tử và kiểu dữ liệu

Thảo luận

Lệnh if

Lệnh if-else

Lệnh if lồng nhau

Lệnh if bậc thang

Lệnh switch-case

Cú pháp câu lệnh if



- Cú pháp:

```
if (condition) {  
    // one or more statements;  
}
```

Trong đó:

- `condition`: là biểu thức trả về giá trị kiểu boolean
- `statements`: Các câu lệnh sẽ được thực thi nếu điều kiện trả về **true**

Cú pháp if-else



- Cú pháp:

```
if (condition) {  
    // one or more statements;  
}  
else {  
    // one or more statements;  
}
```

Trong đó:

- condition: điều kiện để đánh giá. Nếu condition trả về **true** thì khối lệnh bên trong **if** được thực thi. Nếu condition trả về **false** thì khối lệnh trong **else** được thực thi.



Câu lệnh if lồng nhau (nested if)

- Một câu lệnh if có thể được đặt trong câu lệnh if khác:

```
if(condition1) {  
    if(condition2)  
        true-block statement(s) ;  
    else  
        false-block statement(s) ;  
}  
else {  
    false-block statement(s) ;  
}
```



Câu lệnh if bậc thang

- Có thể đặt các câu lệnh điều kiện if-else liên tiếp nhau

```
if(condition) {  
    // one or more statements  
}  
else if (condition) {  
    // one or more statements  
}  
else {  
    // one or more statements  
}
```


switch-case: Cú pháp



```
switch (switch-expression) {  
    case value1: statement(s)1;  
                break;  
  
    case value2: statement(s)2;  
                break;  
  
    ...  
    case valueN: statement(s)N;  
                break;  
    default:    statement(s)-for-default;  
}
```

- Trong đó:
 - *switch-expression*: là biểu thức trả về giá trị thuộc một trong các kiểu: char, byte, short, int hoặc String
 - *value1, ..., valueN* có cùng kiểu dữ liệu so với *switch-expression*
 - *break* là từ khoá để dừng thực thi các câu lệnh ở phía sau. *break* là không bắt buộc.
 - *default* là từ khoá để quy định khối lệnh sẽ được thực thi nếu không có trường hợp nào ở các *case* là đúng. *default* là không bắt buộc.

So sánh if và switch-case



if	switch-case
Có thể sử dụng để so sánh lớn hơn, nhỏ hơn...	Chỉ có thể sử dụng để so sánh bằng hoặc khác nhau
Mỗi câu lệnh if có một biểu thức điều kiện, với giá trị trả về là true hoặc false	Tất cả các trường hợp (case) đều so sánh với giá trị của biểu thức điều kiện duy nhất
Biểu thức điều kiện cần trả về giá trị kiểu boolean	Biểu thức điều kiện cần trả về giá trị là kiểu byte, short, char, int , hoặc String
Chỉ có một khối lệnh được thực thi nếu điều kiện đúng	Nếu điều kiện đúng mà không có câu lệnh break thì tất cả các khối lệnh ở phía sau cũng được thực thi



Demo

if-else

switch-case

Thảo luận

Cấu trúc lặp for

Cấu trúc lặp while

Cấu trúc lặp do..while

Cấu trúc lặp for-each

Lệnh break

Lệnh continue

Vòng lặp for



- Cú pháp:

```
for (initial-action; loop-continuation-condition; action-after-each-iteration) {  
    statement(s);  
}
```

Trong đó:

- *initial-action*: là các câu lệnh được thực thi một lần duy nhất khi vòng lặp bắt đầu chạy
- *loop-continuation-condition*: là biểu thức điều kiện để xác định xem vòng lặp có được tiếp tục hay không
- *statement(s)*: là khối lệnh sẽ được thực thi trong mỗi lần lặp
- *action-after-each-iteration*: là các câu lệnh được thực thi sau mỗi lần lặp



Vòng lặp for: Ví dụ

- Hiển thị bảng cửu chương của 5

initial-action *loop-continuation-condition* *action-after-each-iteration*

```
for (int i = 1; i <= 10; i++){  
    int product = 5 * i;  
    System.out.println("5 x " + i + " = " + product);  
}
```

statement(s)

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Lưu ý: Biến *i* ở vòng lặp trên còn được gọi là biến điều khiển (control variable). Phạm vi của biến *i* là bên trong vòng lặp for.

Vòng lặp for-each

- Vòng lặp for-each (còn gọi là *enhanced for*) được sử dụng để duyệt qua các phần tử của một collection, chẳng hạn như mảng, ArrayList, LinkedList, HashSet...
- Cú pháp:

```
for (type var: collection) {  
  
}
```

Trong đó:

- type: Kiểu dữ liệu của các đối tượng của collection
- var: Biến đại diện lần lượt cho từng phần tử của collection trong mỗi lần lặp
- collection: đối tượng cần lặp



for-each: Ví dụ

- Duyệt qua các phần tử của một mảng:

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

```
for (int element: array) {  
    System.out.println("element = " + element);  
}
```

- Tương đương với câu lệnh for:

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


Vòng lặp while



- Vòng lặp *while* thực thi lặp lại một khối lệnh nếu biểu thức điều kiện trả về giá trị đúng
- Cú pháp:

```
while (loop-continuation-condition) {  
    statement(s);  
}
```

Trong đó:

- *loop-continuation-condition* : là biểu thức điều kiện
- *statement(s)*: là các câu lệnh được thực thi trong mỗi lần lặp

Vòng lặp while: Ví dụ



- Tìm đáp án của phép cộng:

```
int number1 = input.nextInt();
int number2 = input.nextInt();
int expectedAnswer = input.nextInt();
while (number1 + number2 != expectedAnswer) {
    System.out.print("Wrong answer");
    expectedAnswer = input.nextInt();
}
System.out.println("You got it!");
```

Vòng lặp do-while



- Cú pháp:

```
do {  
    statements(s);  
} while (loop-continuation-condition);
```

Trong đó:

- statement(s): Các câu lệnh được thực thi trong mỗi lần lặp
- loop-continuation-condition: Biểu thức điều kiện. Nếu biểu thức điều kiện trả về giá trị true thì vòng lặp sẽ tiếp tục thực thi. Nếu biểu thức điều kiện trả về false thì vòng lặp kết thúc

do-while: Ví dụ



- Tính tổng các số nguyên:
 - Tính tổng của các số được nhập vào từ bàn phím
 - Nếu nhập vào số 0 thì kết thúc chương trình

```
int sum = 0;
int number;
Scanner scanner = new Scanner(System.in);
do{
    System.out.print("Enter a number: ");
    number = scanner.nextInt();
    sum += number;
} while (number != 0);
System.out.println("The sum is: " + sum);
```

break



- Câu lệnh break được sử dụng để kết thúc một vòng lặp
- Ví dụ:

```
for (int i = 0; i < 10; i++){  
    if(i == 5)  
        break;  
    System.out.println("i = " + i);  
}  
System.out.println("End of loop");
```

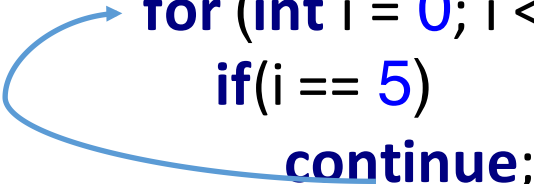
i = 0
i = 1
i = 2
i = 3
i = 4
End of loop

continue



- Câu lệnh continue được sử dụng để bỏ qua vòng lặp hiện tại
- Ví dụ:

```
for (int i = 0; i < 10; i++){  
    if(i == 5)  
        continue;  
    System.out.println("i = " + i);  
}  
System.out.println("End of loop");
```



i = 0
i = 1
i = 2
i = 3
i = 4
i = 6
i = 7
i = 8
i = 9
End of loop



Demo

for
while
do...while
for-each



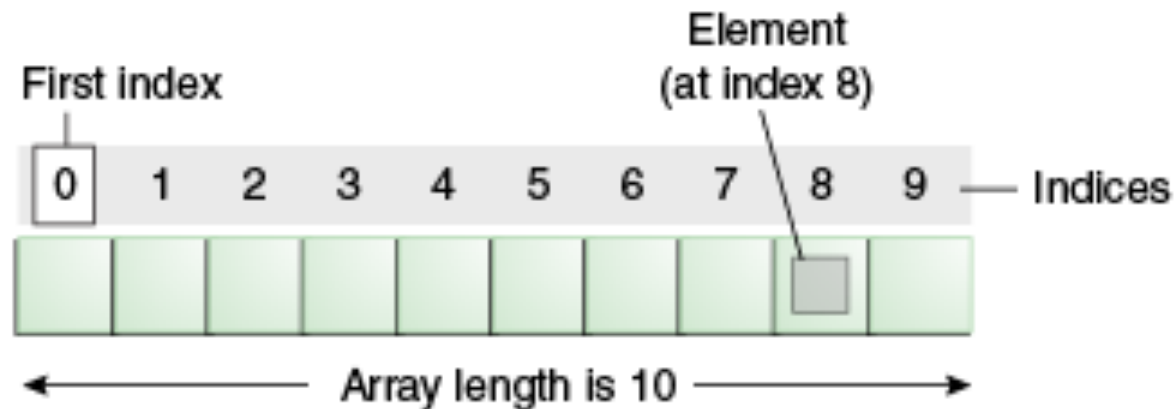
Thảo luận

Mảng

Mảng



- Mảng là một biến tham chiếu đến một loạt giá trị liên tiếp nhau
- Các giá trị được lưu trữ trong mảng có cùng kiểu dữ liệu
- Các khái niệm của mảng:
 - Tên mảng: Tuân thủ theo quy tắc đặt tên của biến
 - Phần tử: Các giá trị được lưu trữ trong mảng
 - Chỉ số: Vị trí của các phần tử trong mảng. Chỉ số bắt đầu từ 0.
 - Độ dài: Số lượng tối đa các phần tử mà mảng có thể lưu trữ



Khai báo mảng



- Cú pháp:

elementType[] arrayRefVar;

Trong đó:

- elementType: Kiểu dữ liệu của các phần tử trong mảng
 - arrayRefVar: Tên của mảng
- Ví dụ, khai báo một mảng có tên **myList** lưu trữ giá trị kiểu double:

double[] myList;

Lưu ý: Có thể sử dụng cú pháp *elementType arrayRefVar[]* để khai báo mảng.

Biến mảng là biến tham chiếu



- Khi khai báo các biến kiểu dữ liệu nguyên thủy thì chúng được cấp phát bộ nhớ tương ứng để lưu trữ dữ liệu
- Khi khai báo biến mảng thì sẽ không có việc cấp phát bộ nhớ ngay cho các phần tử của mảng. Chỉ có việc cấp phát bộ nhớ cho tham chiếu đến mảng
- Nếu không gán tham chiếu đến mảng thì giá trị của biến mảng là **null**
- Không thể gán các phần tử cho mảng nếu chưa khởi tạo mảng

Khởi tạo mảng



- Sử dụng từ khoá `new` để khởi tạo mảng:

arrayRefVar = **new** *elementType*[*arraySize*];

Câu lệnh trên thực hiện 2 việc:

1. **new** *elementType*[*arraySize*]: Khởi tạo một mảng mới
2. Gán tham chiếu của mảng vừa tạo cho biến *arrayRefVar*

- Có thể gộp chung việc khai báo mảng, khởi tạo mảng và gán tham chiếu cho biến mảng:

elementType[] *arrayRefVar* = **new** *elementType*[*arraySize*];

Hoặc:

elementType *arrayRefVar*[] = **new** *elementType*[*arraySize*];



Ví dụ khởi tạo mảng

- Khai báo và khởi tạo một mảng **double** tên là **myList** có thể chứa 10 phần tử:

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

Hoặc:

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

- Khai báo và khởi tạo một mảng String tên là names có thể chứa 30 phần tử:

```
String[] names = new String[30];
```

Hoặc:

```
String names[] = new String[30];
```

Gán giá trị cho các phần tử mảng



- Cú pháp:

arrayRefVar[index] = value;

Trong đó:

- *index*: Chỉ số (vị trí) của phần tử muốn gán giá trị. Chỉ số của phần tử đầu tiên là 0
- *value*: Giá trị muốn gán cho phần tử tại vị trí *index*

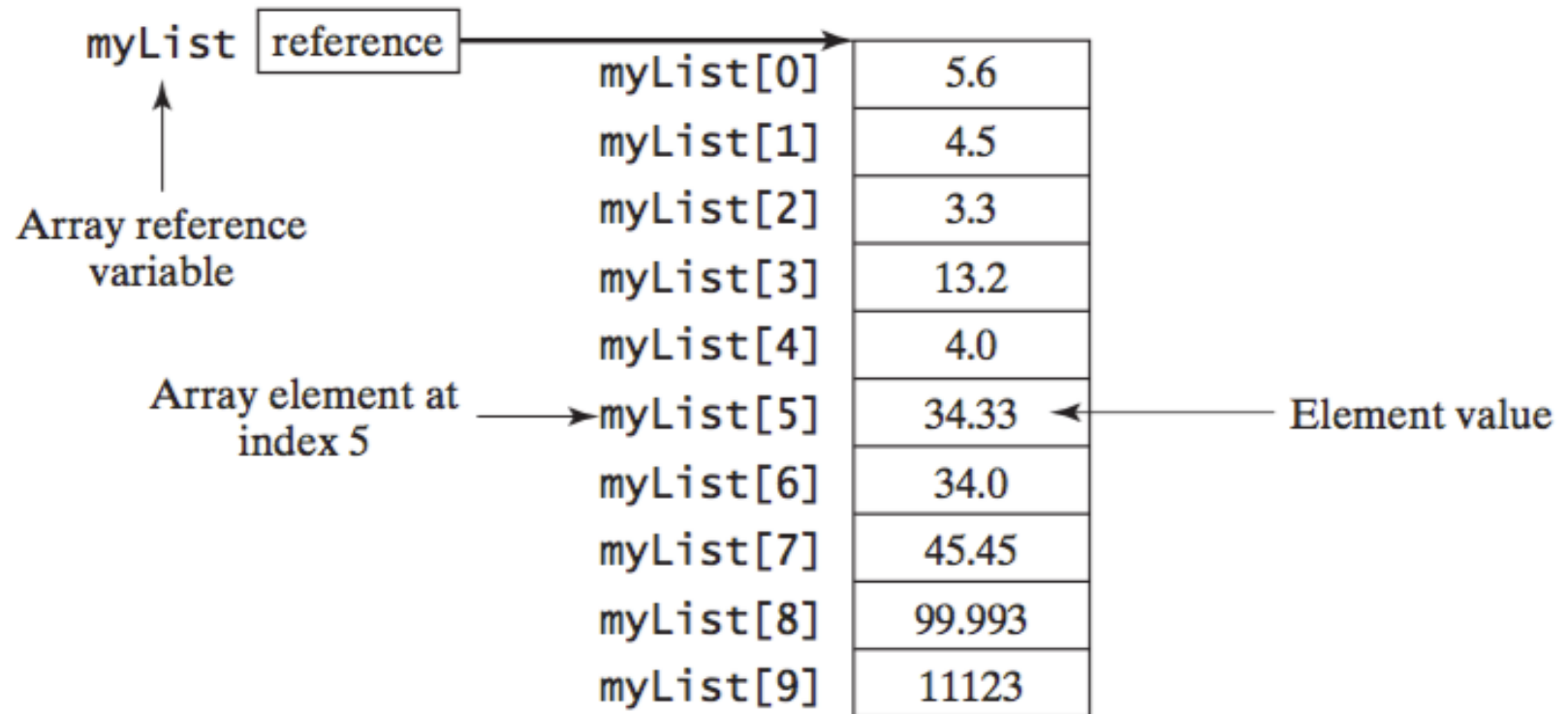
Lưu ý: Một mảng sau khi được khởi tạo mà không được gán giá trị cho các phần tử của nó thì các phần tử sẽ có giá trị mặc định tùy theo kiểu dữ liệu của mảng. (0, false, null, \u0000)

Ví dụ gán giá trị cho phần tử mảng



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

```
myList[0] = 5.6;  
myList[1] = 4.5;  
myList[2] = 3.3;  
myList[3] = 13.2;  
myList[4] = 4.0;  
myList[5] = 34.33;  
myList[6] = 34.0;  
myList[7] = 45.45;  
myList[8] = 99.993;  
myList[9] = 11123;
```





Phân biệt đối tượng mảng với biến mảng

- Đối tượng mảng (array object) là khác với biến mảng (array variable)
- Ví dụ, với khai báo:

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

- Chúng ta cần hiểu: *myList* là một biến chứa tham chiếu đến một mảng *double* có 10 phần tử
- Tuy nhiên, thông thường chúng ta cũng có thể nói ngắn gọn: *myList* là một mảng *double* chứa 10 phần tử

Lưu ý: Việc phân biệt biến tham chiếu và đối tượng là rất quan trọng. Vấn đề này sẽ được đề cập đến trong phần về đối tượng.



Độ dài của mảng

- Khi khởi tạo một mảng thì cần quy định độ dài (length) của mảng đó
- Độ dài của mảng là số lượng phần tử tối đa mà mảng có thể chứa
- Độ dài của mảng giúp máy tính biết dung lượng bộ nhớ cần cấp phát (allocate) cho mảng
- Độ dài của mảng còn được gọi là kích thước (size) của mảng
- Không thể thay đổi kích thước của mảng sau khi đã khởi tạo
- Để lấy được độ dài của mảng thì sử dụng thuộc tính *length*

Ví dụ:

```
int x = myList.length; //x có giá trị là 10
```

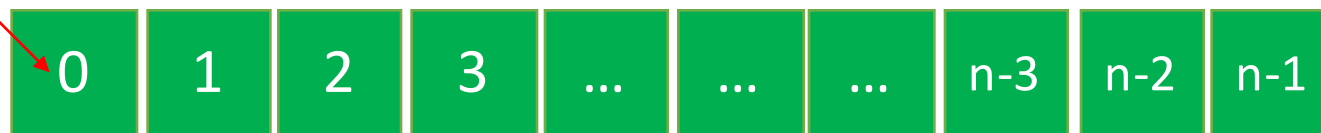


Chỉ số của các phần tử mảng

- Chỉ số (index) của phần tử còn được gọi là vị trí (position) của phần tử đó
- Chỉ số của phần tử đầu tiên là 0
- Chỉ số của phần tử cuối cùng là $n - 1$, trong đó n là độ dài của mảng

Phần tử đầu tiên

Phần tử cuối cùng



n : độ dài của mảng



Khởi tạo nhanh mảng

- Java cung cấp cú pháp ngắn gọn để khởi tạo nhanh mảng, còn được gọi là *array initializer*
- Cú pháp:

elementType[] arrayRefVar = {value0, value1, ..., valuek};

- Ví dụ:

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



Demo

Tạo mẫu

Duyệt mẫu



Thảo luận

Duyệt mǎng



Sử dụng vòng lặp for

- Sử dụng vòng lặp for để duyệt qua tất cả các phần tử của mảng

```
int[] mylist = {1,3,5,7,9};  
for (int i=0;i< mylist.length; i++){  
    System.out.println(mylist[i]);  
}
```



Sử dụng for - each

- Sử dụng vòng lặp foreach để duyệt qua tất cả các phần tử của mảng

```
for (int item: mylist){  
    System.out.println(item);  
}
```



Thảo luận

Phương thức

Phương thức

- Phương thức (method) là một nhóm các câu lệnh thực hiện một nhiệm vụ nhất định
- *Phương thức* là thuật ngữ được sử dụng phổ biến trong Lập trình hướng đối tượng. Trong nhiều trường hợp khác, các tên gọi được sử dụng là *hàm* (function) và *thủ tục* (procedure)
- `System.out.println()`, `Math.pow()`, `Math.random()` là các phương thức đã được định nghĩa sẵn cho chúng ta sử dụng

Lưu ý: Mặc dù tên gọi phương thức, hàm, procedure đôi khi có thể sử dụng thay thế cho nhau, nhưng giữa 3 khái niệm này có sự khác nhau.

Khai báo phương thức



- Cú pháp:

```
modifier returnType methodName(list of parameters) {  
    // Method body;  
}
```

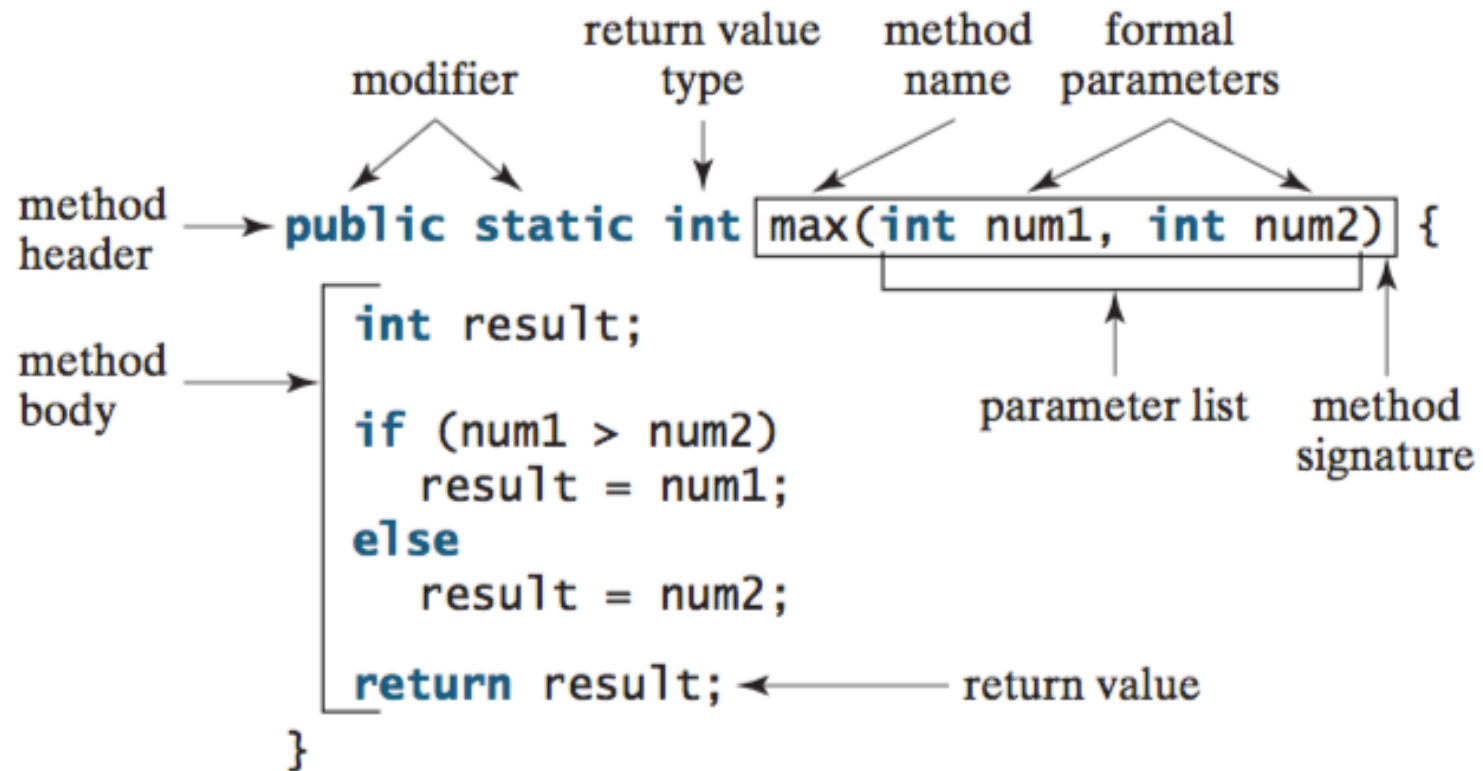
Trong đó:

- modifier có thể là các từ khoá để quy định các tính chất khác nhau của phương thức
- returnType là kiểu dữ liệu trả về của phương thức
- methodName là tên gọi của phương thức
- list of parameters là danh sách các tham số của phương thức
- Method body là phần thân của phương thức



Ví dụ: Cấu phần của một phương thức

- Phương thức xác định số lớn nhất trong 2 số:



Kiểu dữ liệu trả về



- Một phương thức có thể trả về một giá trị
- Nếu phương thức không trả về giá trị thì có kiểu dữ liệu trả về **void**
- Ví dụ, System.**out**.println() là một phương thức void
- Ví dụ, phương thức kiểm tra số chẵn có kiểu dữ liệu trả về là boolean:

```
public static boolean isEven(int number){  
    return number % 2 == 0;  
}
```



Tham số (parameter) và đối số (argument)

- Tham số (còn được gọi đầy đủ là tham số hình thức – formal parameter) là các biến được khai báo trong phần header
- Khi gọi phương thức thì giá trị của các biến này sẽ được truyền vào. Các giá trị này được gọi là tham số thực (actual parameter) hoặc đối số (argument)
- Ví dụ:

parameter
↓

```
public static boolean isEven(int number){  
    return number % 2 == 0;  
}
```

argument
↓

```
isEven(5);
```



Gọi phương thức

- Gọi (*call* hoặc *invoke*) phương thức là cách để thực thi một phương thức đã được định nghĩa trước đó
- Khi gọi phương thức thì cần truyền đối số vào
- Ví dụ, gọi phương thức void:

```
System.out.println("Welcome to Java!");
```

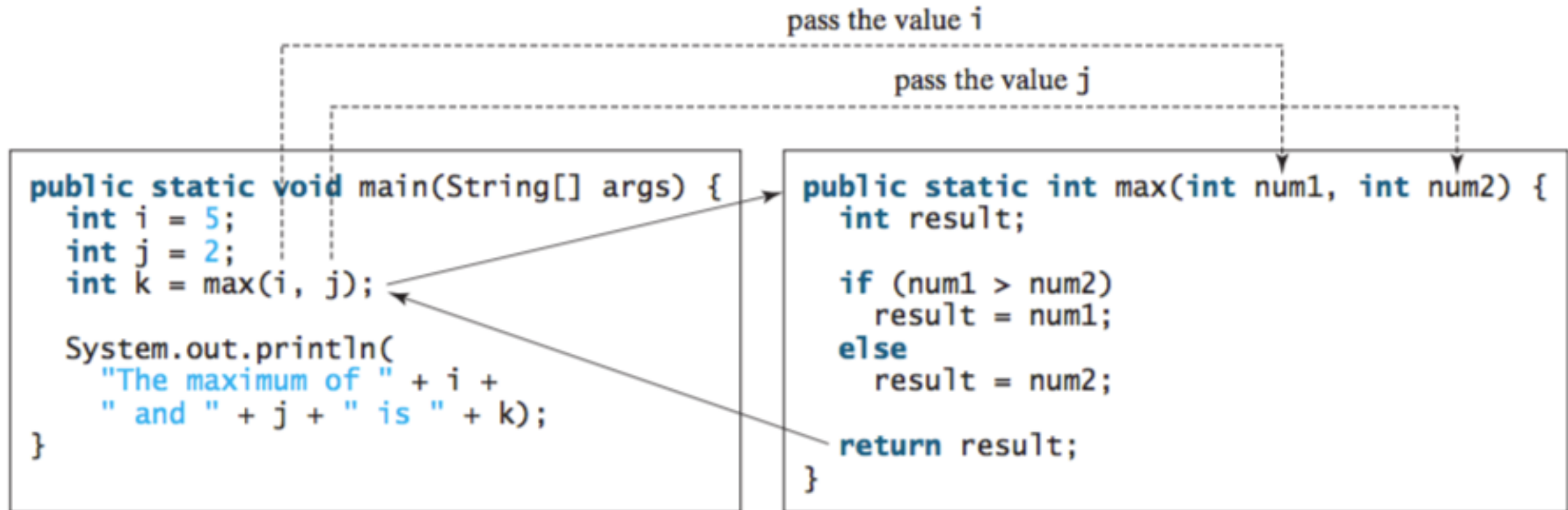
- Ví dụ, gọi phương thức có giá trị trả về:

```
int larger = max(3, 4);
```

Luồng điều khiển (control flow)



- Khi gọi phương thức, luồng điều khiển được chuyển cho phương thức đó





Phương thức main()

- Phương thức main() là một phương thức đặc biệt trong Java
- Phương thức main() là điểm khởi đầu (entry point) cho một chương trình
- Phương thức main() được gọi bởi JVM
- Header của phương thức main() được quy định sẵn

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




Demo

Sử dụng phương thức



Tóm tắt bài học

- Java hỗ trợ nhiều kiểu dữ liệu khác nhau
- Các câu lệnh điều khiển giúp điều hướng luồng thực thi của ứng dụng
- Các vòng lặp được Java hỗ trợ: for, for-each, while
- Các khái niệm của mảng: Tên mảng, kiểu dữ liệu, kích thước, phần tử, chỉ số
- Tên của mảng tuân theo quy tắc của tên biến
- Chỉ số của phần tử đầu tiên là 0
- Chỉ số của phần tử cuối cùng là length – 1
- Có thể sử dụng vòng lặp for và for-each để duyệt mảng

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: *Lớp và đối tượng*