Contents

[Introduction to Java 1](#_Toc110278911)

[1. Java 1](#_Toc110278912)

[2. Ưu điểm. 1](#_Toc110278913)

[3. JDK JRE JVM 1](#_Toc110278914)

[4. Kiểu dữ liệu trong Java 2](#_Toc110278915)

[Phím tắt. 3](#_Toc110278916)

[Linh Tinh 3](#_Toc110278917)

### Introduction to Java

## Java

* Java là ngôn ngữ lập trình hướng đối tượng.
* Có thể chạy trên nhiều nền tảng khác nhau
* Slogan “Write one, run anywhere”
* Có 4 nền tảng:

+ JSE (J Standard Edition): là J core- Phiên bản tiêu chuẩn và cở bản của J.

+ JEE (J Enterprise Edition): Phát triển web.

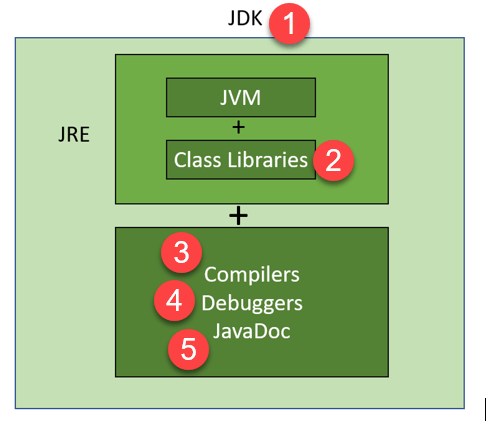
+ JME (J Micro Edition): Phát triển ứng dụng mobile và các thiết bị nhỏ.

+ JFX: Phát triển các ứng dụng liên quan đến mạng.

1. Write once, run anywhere

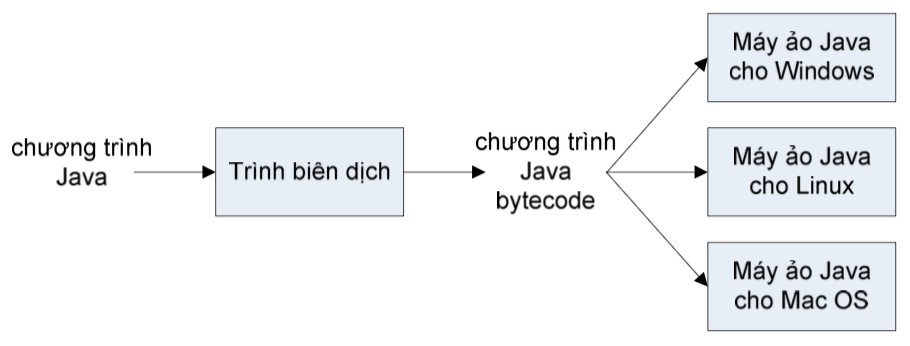
Java có tính độc lập nền tảng (platform independent). Một nền tảng (platform) ở đây có nghĩa một hệ thống máy tính với hệ điều hành cụ thể. Có nghĩa là một chương trình Java có thể chạy trên các nền tảng khác nhau mà không phải dịch lại.

## JDK JRE JVM

* JDK: J development Kit: Bộ công cụ để giúp chạy được một ứng dụng J hoàn chỉnh.
* 

1. Thông dịch, biên dịch.

Những người thiết kế Java chọn cách tổ hợp giữa trình biên dịch và trình thông dịch. Các chương trình viết bằng Java được biên dịch thành mã máy được gọi là Java bytecode, hay ngắn gọn là bytecode. Máy ảo Java (Java Virtual Machine – JVM) thông dịch J bytecode này cho máy tính bất kỳ để chạy chương trình Java.



## Kiểu dữ liệu trong Java

Các biến nguyên thủy (Primitives): Một biến nguyên thủy (primitive) có thể là một trong tám kiểu: char, boolean, byte, short, int, long, double, hoặc float. Khi một biến nguyên thủy đã được khai báo, kiểu nguyên thủy của nó không bao giờ có thể thay đổi, mặc dù trong hầu hết các trường hợp, giá trị của nó có thể thay đổi.

Các biến tham chiếu (Reference): Một biến tham chiếu (reference) được sử dụng để tham chiếu đến (hoặc truy cập) một đối tượng. Một biến tham chiếu được khai báo là một kiểu cụ thể và kiểu đó không bao giờ có thể thay đổi được. Một biến tham chiếu có thể được sử

dụng để tham chiếu đến bất kỳ đối tượng nào thuộc kiểu đã khai báo hoặc kiểu con của kiểu khai báo (kiểu tương thích).



1. Các loại toán tử:

Vở

1. So sánh If else – Switch Case

|  |  |
| --- | --- |
| If | switch-case |
| Có thể sử dụng để so sánh lớn hơn, nhỏ hơn… | Chỉ có thể sửdụng 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 kiệnđúng | Nếu điều kiện đúng màkhông có câu lệnh break thì tất cả các khối lệnh ở phía sau cũng được thựcthi |

### Vòng Lặp trong Java

1. For Each

**Cú pháp của vòng lặp for each trong Java**

for (kieuDuLieu bienDaiDien : tenMang) {

    // Khối lệnh được lặp lại

}

Vd:

in ra tất cả các phần tử trong mảng mà không sử dụng biến đếm.

// Khai báo một mảng

int[] arr = {1, 2, 5, 8, 9};

// Sử dụng vòng lặp for each để lặp qua mảng

for(int n : arr) {

    System.out.println(n);

}

**Ưu điểm của vòng lặp for each trong Java**

Sử dụng vòng lặp for each làm cho code dễ đọc hơn

Giảm bớt khả năng lỗi khi lập trình

**Nhược điểm của vòng lặp for each trong Java**

Vòng lặp for each không thể chỉnh sửa mảng, chỉ được dùng để duyệt qua tất cả các phần tử trong mảng.

Không thể tìm được vị trí phần tử trong mảng.

Vòng lặp for each chỉ duyệt được xuôi, không thể duyệt ngược mảng.

**Vậy khi nào thì nên sử dụng for each trong lập trình Java?**

Bạn có thể sử dụng for each khi bạn chỉ muốn duyệt qua tất cả các phần tử trong mảng hay collection, duyệt từ đầu đến cuối, không bỏ sót một phần tử nào, không cần chỉnh sửa giá trị mảng, không cần tìm index của phần tử trong mảng hay collection.

### Mảng và phương thức trong JAVA

1. Khải niệm 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.

1. Khai báo mảng

Để sử dụng một mảng trong một chương trình, bạn phải khai báo một biến để tham chiếu tới mảng và xác định kiểu dữ liệu của các phần tử trong mảng.

elementType[] arrayRefVar;

**elementType**có thể là bất cứ kiểu dữ liệu nào (int, double, char, String, Object …)

1. **Tạo mảng**

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

Câu lệnh này làm hai việc: (1) Tạo một mảng bằng cách sử dụng **new elementType[arraySize**; (2) Gán tham chiếu của mảng vừa được tạo ra cho biến arrayRefVar.

Khai báo biến mảng, tạo mảng, và gán tham chiếu của mảng vào biến có thể kết hợp trong một câu lệnh như sau:

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

Ví dụ:

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

**Chú ý:**

Một biến mảng lưu trữ một mảng thực sự chứa một tham chiếu đến mảng đó

Khi một mảng được tạo, các phần tử của nó được gán giá trị mặc định là 0 cho các kiểu dữ liệu số nguyên thủy, \ u0000 cho kiểu char, và false đối với các kiểu boolean.

Khởi tạo mảng

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

Ví dụ:

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

**Chú ý:**

Toán tử **new** không được sử dụng trong cú pháp khởi tạo mảng. Khi khởi tạo các phần tử trong mảng, bạn cần phải theo thứ tự *khai báo*, *tạo* và *khởi tạo* trên cùng một câu lệnh

1. Phương thức

Phương thức là tập các câu lệnh được nhóm lại với nhau để thực hiện một thao tác

Các phương thức cho phép chia sẻ mã lệnh và tái sử dụng lại.

Định nghĩa một phương thức bao gồm tên phương thức, danh sách các tham số, kiểu dữ liệu trả về và phần thân để thực thi phương thức.

Cú pháp để định nghĩa một phương thức như sau:

modifier returnValueType methodName(list of parameters) {

*// Method body;*

}

Nếu một phương thức trả về một giá trị, nó được gọi là phương thức trả về gía trị (value-returning method), nếu không nó được gọi là một phương thức không trả về gì (void method)

tham số hình thức (formal parameters)

tham số thực (actual parameters) hay đối số (argument)

Chúng ta nói “define a method – định nghĩa một phương thức” và “declare a variable – khai báo một biến”. Đó là cách để tạo sự khác biệt tinh tế ở đây. Một định nghĩa xác định thứ đã định nghĩa là gì, nhưng một khai báo thường liên quan đến việc cấp phát bộ nhớ để lưu trữ dữ liệu cho mục khai báo.

Gọi phuowgn thức: call hoặc invoke

Gọi phương thức là cách thực thi mã lệnh trong phương thức đó.

1. Một số phương thức trong java:

- Math.round

Math.round(double value) làm tròn số

**Phương thức trả về:**

* Giá trị int nếu đối số là float.
* Giá trị long nếu đối số là double.
* Math.random()

phương thức random() cho phép chúng ta tạo ra một số thập phân có giá trị lớn hơn hoặc bằng 0.0 và bé hơn 1.0(0.99999), giá trị của số thập phân này là ngẫu nhiên mỗi khi chạy phương thức.

int a = (int)Math.round(12 \* Math.random()) + 5;

Thực hiện: B1: Math.random() -> Ngẫu nhiên từ 0,00001-0,99999 kiểu double

B2: 12\* Math.random()-> ngẫu nhiên từ 0,00001-11,99999 kiểu double

B3: Math.round(12 \* Math.random()) -> làm tròn -> ngẫu nhiên từ 0-12 kiểu double

B4: (int)Math.round(12 \* Math.random()) -> ép thành kiểu int

B5: (int)Math.round(12 \* Math.random()) + 5 -> ngẫu nhiên từ 5-17 kiểu int

lớp ThreadLocalRandom

Lớp ThreadLocalRandom cung cấp phương thức nextInt() với hai tham số truyền vào là giá trị biên dưới và giá trị biên trên, giá trị trả về sẽ lớn hơn hoặc bằng biên dưới và bé hơn giá trị biên trên.

Do dó để có phạm vi sinh số ngẫu nhiên [1, 100] thì hai tham số truyền vào sẽ là 1 và 101

int ranNum = ThreadLocalRandom.current().nextInt(1,101);

Chỉ random trong số >0; nếu nhỏ hơn 0 chương trình sẽ trả về exception thuộc lớp java.lang.IllegalArgumentException.

arraycopy trong lớp java.lang.System

arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

Các tham số srcPos và tarPos chỉ ra các vị trí bắt đầu trong sourceArray và targetArray, tương ứng. Số phần tử được sao chép từ sourceArray sang targetArray được biểu thị bằng length

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

Phương thức arraycopy không cấp phát không gian bộ nhớ cho mảng đích. Mảng đích phải được tạo trước với không gian bộ nhớ được cấp phát. Sau khi sao chép diễn ra, targetArray và sourceArray có cùng nội dung nhưng vị trí bộ nhớ độc lập.

Tìm hiểu thêm equals() hashCode() notify() notifyAll()…

### Lớp và đối tượng trong Java

### **Các khái niệm trong lập trình hướng đối tượng**

**Đối tượng:** Một thực thể có trạng thái và hành vi. Ví dụ như xe đạp, bàn, ghế, … Nó có thể mang tính vật lý hoặc logic.

**Lớp:** Một tập hợp các đối tượng. Nó là một thực thể logic.

**Tính kế thừa:** Khi một đối tượng đạt được các thuộc tính và các hành vi của đối tượng cha, thì đó là tính kế thừa. Điều này làm tăng tính tái sử dụng cho code. Nó được sử dụng để đạt được tính đa hình tại runtime.

**Tính đa hình:** Khi một tác vụ được thực hiện theo nhiều cách khác nhau được gọi là tính đa hình. Ví dụ: như vẽ hình chữ nhật hoặc hình tam giác, … Trong Java, chúng ta sử dụng nạp chồng phương thức (method overloading) và ghi đè phương thức (method overriding) để có tính đa hình. Một ví dụ khác: con mèo kêu meooo, còn chú chó thì sủa goooo.

**Tính trừu tượng:** Đó là ẩn các chi tiết nội tại và hiển thị tính năng. Ví dụ, với cuộc gọi điện thoại, chúng ta không biết tiến trình xử lý nội tại là như thế nào. Trong Java, chúng là sử dụng lớp abstract và abstract interface để có tính trừu tượng.

**Tính bao đóng:** Đó là gắn kết code và dữ liệu cùng với nhau vào trong một đơn vị unit đơn. Ví dụ: có thể bạn đã biết đến viên thuốc con nhộng (hay đơn giản hơn là gói bột giặt), các viên thuốc (hạt bột giặt) khác nhau được đóng gói.

* 1. Constructor

*Constructor (phương thức khởi tạo) được gọi để tạo đối tượng sử dụng toán tử***new***.*Constructor là một phương thức đặc biệt. Chúng có 3 đặc điểm nổi bật sau:

* Constructor phải có cùng tên với tên lớp của nó
* Constructor không có kiểu dữ liệu trả về
* Constructor được gọi sử dụng toán tử new khi một đối tượng được tạo. Constructor đóng vai trò quan trọng trong việc khởi tạo đối tượng.

gọi constructor của lớp sử dụng toán tử new với cú pháp như sau:

**new** ClassName(arguments);

#### ****Biến tham chiếu****

* Câu lệnh sau khai báo biến myCircle thuộc kiểu Circle:
* Circle myCircle;
* Biến **myCircle**có thể tham chiếu tới đối tượng **Circle**. Câu lệnh tiếp theo tạo một đối tượng và gán tham chiếu của nó tới **myCircle**:
* myCircle = **new** Circle();
* Bạn có thể viết một câu lệnh đơn giản kết hợp khai báo một biến tham chiếu đối tượng, tạo ra một đối tượng và gán đối tượng tham chiếu đến biến với cú pháp sau:
* ClassName objectRefVar = **new** ClassName();
* Đây là một ví dụ:
* Circle myCircle = **new** Circle();

#### 4. ****Truy cập phương thức và dữ liệu của đối tượng****

Trong thuật ngữ OOP, thành viên của đối tượng được biết đến là trường dữ liệu và phương thức của nó. Sau khi đối tượng được tạo, trường dữ liệu và phương thức có thể được truy cập bằng cáh sử dụng toán tử chấm (.), toán tử (.) còn được gọi là toán tử truy cập thành viên đối tượng.

* Cú pháp tham chiếu tới trường dữ liệu trong đối tượng: **objectRefVar.dataField**
* Cú pháp gọi phương thức trên đối tượng: **objectRefVar.method(arguments)**

Giá trị mặc định của trường dữ liệu là **null**cho mọi kiểu tham chiếu, **0** cho kiểu dạng số, **false**cho kiểu **boolean**, và **\u0000** cho kiểu ký tự.

1. Có hai loại Modifier trong Java, đó là: Access Modifier và Non-access Modifier. Access Modifer trong Java xác định phạm vi có thể truy cập của thành viên dữ liệu, phương thức, constructor hoặc lớp. Có 4 loại Access Modifier là: private, default, protected, và public.

default: Truy cập trong nội bộ package

private: Truy cập trong nội bộ lớp

public: Thành phần công khai, truy cập tự do từ bên ngoài

protected: Thành phần được bảo vệ, bị hạn chế truy nhập từ bên ngoài

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **Bên trong lớp** | **Bên trong package** | **Bên ngoài package chỉ bởi lớp con** | **Bên ngoài package** |
| private | C | K | K | K |
| default | C | C | K | K |
| protected | C | C | C | K |
| public | C | C | C | C |

### **Dưới đây là một số khác biệt quan trọng nhất giữa Java constructor và Java method:**

* **Invoking:**Trong khi constructor được gọi ngầm, phương thức lại được gọi rõ ràng
* **Java compiler**: Trình biên dịch Java không bao giờ cung cấp một phương thức. Tuy nhiên, trình biên dịch Java cung cấp một constructor mặc định nếu không có một cái nào được định nghĩa trong class.
* **Cách đặt tên**: Tên của constructor trong Java phải giống với tên của class. Tuy nhiên, phương thức có thể cùng tên với class chứa nó hoặc không.
* **Số lần gọi**: Một hàm constructor được gọi một lần và chỉ trong thời gian tạo đối tượng. Mặt khác, một phương thức Java có thể được gọi là nhiều lần.
* **Kiểu trả về**: Một phương thức Java phải có kiểu trả về nhưng có cùng kiểu với hàm constructor hay không thì không bắt buộc.
* **Cách sử dụng**: Trong khi một phương thức được sử dụng để phơi bày hành vi của một đối tượng Java, thì một hàm constructor được sử dụng để khởi tạo trạng thái giống nhau.

## **Sự khác nhau giữa constructor và phương thức trong java**

|  |  |
| --- | --- |
| **Constructor** | **Phương thức** |
| Constructor được sử dụng để khởi tạo trạng thái của một đối tượng. | Phương thức được sử dụng để thể hiện hành động của một đối tượng. |
| Constructor không có kiểu trả về. | Phương thức có kiểu trả về. |
| Constructor được gọi ngầm. | Phương thức được gọi tường minh. |
| Trình biên dịch Java tạo ra constructor mặc định nếu bạn không có constructor nào. | Phương thức không được tạo ra bởi trình biên dịch Java. |
| Tên của constructor phải giống tên lớp. | Tên phương thức có thể giống hoặc khác tên lớp. |

### **Câu hỏi 1: Liệu constructor có trả về giá trị nào không?**

Trả lời:  
  
  
Mặc dù bạn không thể sử dụng kiểu trả về với Java constructor, nhưng nó trả về một giá trị. Một hàm tạo Java trả về thể hiện của clalss hiện tại.

### Access modifier, static method, static property

* 1. Static

## **Biến static trong Java**

Khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.

1. Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, ...
2. Biến static lấy bộ nhớ chỉ một lần trong Class Area tại thời gian tải lớp đó.

### **Lợi thế của biến static**

Sử dụng biến static giúp chương trình của bạn sử dụng bộ nhớ hiệu quả hơn (tiết kiệm bộ nhớ).

Như bạn đã thấy ở trên, biến static sẽ lấy bộ nhớ chỉ một lần, nếu bất cứ đối tượng nào thay đổi giá trị của biến static, nó sẽ vẫn ghi nhớ giá trị của nó.

## **Phương thức static trong Java**

Nếu bạn áp dụng từ khóa static với bất cứ phương thức nào, thì phương thức đó được gọi là phương thức static.

1. Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
2. Một phương thức static gọi mà không cần tạo một instance của một lớp.
3. Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó.

### **Sự hạn chế của phương thức static**

Có hai hạn chế chính đối với phương thức static. Đó là:

1. Phương thức static không thể sử dụng biến non-static hoặc gọi trực tiếp phương thức non-static.
2. Từ khóa this và super không thể được sử dụng trong ngữ cảnh static.

## **Khối static trong Java**

1. Được sử dụng để khởi tạo thành viên dữ liệu static.
2. Nó được thực thi trước phương thức main tại lúc tải lớp.

Getter seter

<https://james.codegym.vn/mod/page/view.php?id=9152>

getter lẫn setter được viết để cung cấp khả năng đọc/ghi lên một thuộc tính, làm nảy sinh một câu hỏi là tại sao không chỉ cần làm cho thuộc tính được public là được.

Nguyên nhân ở đây là do getter và setter không chỉ hướng đến những thao tác đọc và ghi giá trị đơn giản. Trong thực tế, như các phương thức bình thường, chúng có thể thực hiện bất kỳ hành động nào. Ví dụ, một getter có thể lưu số lần giá trị của thuộc tính được đọc ra:

public String getTitle() {

titleAccessCount++; *// Increment member variable titleAccessCount.*

return title;

}

và một setter có thể kiểm tra sự hợp lệ của giá trị đang được yêu cầu ghi vào:

public void setTitle( String newTitle ) {

if ( newTitle == null ) *// Don’t allow null strings as titles!*

title = "(Untitled)"; *// Use an appropriate default value instead.*

else

title = newTitle;

}

Kể cả khi các getter và setter đang không thực hiện công việc gì khác, điều đó không có nghĩa rằng tương lai vẫn sẽ như vậy. Nếu các lớp khác đã truy cập tới thuộc tính thông qua getter và setter ngay từ đầu, trong tương lai, những điều chỉnh bên trong các getter và setter có thể được thực hiện mà không kéo theo bất kỳ thay đổi nào trên các lớp bên ngoài.

Một vài lưu ý cuối cùng ở đây là, đầu tiên, một vài khía cạnh nâng cao của Java phụ thuộc mạnh mẽ vào quy ước đặt tên cho các getter và setter, cho nên luôn tuyệt đối tuân thủ quy ước. Điều thứ hai, các getter và setter cho một “thuộc tính” của class có thể tồn tại mà không cần đến thuộc tính private tương ứng. Ví dụ, một class có thuộc tính firstName và lastName, class đó có thể có cặp phương thức truy cập getFullname và setFullname mà không cần tới thuộc tính fullname thực sự.

Package

### **Lợi thế của package trong Java**

* Java package được sử dụng để phân loại các lớp và các interface để mà chúng có thể được duy trì dễ dàng hơn.
* Java package cung cấp bảo vệ truy cập.
* Java package xóa bỏ các xung đột về đặt tên.

### **Từ khóa import trong Java**

Nếu một class sử dụng một class khác cùng package, tên package không cần được sử dụng. Lớp trong cùng package tìm thấy nhau mà không cần cú pháp đặc biệt nào.

### Linh Tinh

1. Printf

Cú pháp:

System.out.printf(local, format, arguments1, arguments2,…, argumentsn);

Trong đó:

* Local: Nếu khác null sẽ được tự động định dạng theo khu vực.
* format: Quy định chuẩn định dạng đầu ra cho các đối số
* Các argument: Đối số cần định dạng.

%d: số nguyên

%s: chuỗi string

%f: số thực

\t khoảng tab

\n xuống dòng.

System.*out*.printf("%-20s%s", "bmi", "Interpretation\n");

System.*out*.printf("%-20.2f%s", bmi, "Underweight");

Trong đó:

%-20 : hiển thị phần tử đó trong 20 khoảng trống

%-20.2f để hiển thị số thực, trong 20 khoảng trống, lấy 2 chữ số sau phần thập phân.

**Dùng để căn lề**

Các bộ định dạng có sẵn trong Printf:

* %c: Ký tự
* %d: Số thập phân (số nguyên) (cơ số 10)
* %e: Dấu phẩy động theo cấp số nhân
* %f: Dấu phẩy động
* %i: Số nguyên (cơ sở 10)
* %o: Số bát phân (cơ sở 8)
* %s: Chuỗi
* %u: Số thập phân (số nguyên) không dấu
* %x: Số trong hệ thập lục phân (cơ sở 16)
* %t: Định dạng ngày / giờ
* %%: Dấu phần trăm
* \%: Dấu phần trăm

Khai báo khác khởi tạo?

Tác dụng của static? Lớp tiện ích? (toàn bộ thuốc tính và phương thức đều là static – có thể gọi lại để sử dụng như Math() Arrays.toString()