

Bài 1: Ngôn ngữ lập trình Java

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

Chương trình Java -> trình biên dịch-> chương trình java bytecode -> máy ảo Java cho Windows + máy ảo java cho Linux + máy ảo java cho Mac OS

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 sang ngôn ngữ máy mà được biên dịch (compile) ra bytecode

bytecode 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) sang 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

Source files (.java files) -> compiler (.class file) -> JVM other libraries (JRE (JVM gets environment to execute . class file)) debugger JDK

Demo: Tạo ứng dụng Java - 1

Bước 1: Tạo project mới trên IntelliJIDEA đặt tên helloworld

Bước 2: Tạo lớp HelloWorld với nội dung:

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

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

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
```

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;
```

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 thủy (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. | <code>int result = +1; //result = 1</code> |
| - | Toán tử trừ | <code>int result = -1; //result = -1</code> |
| ++ | Toán tử tăng 1 giá trị | <code>int result = 1; int result++; //result=2;</code> |
| -- | Toán tử giảm 1 giá trị | <code>int result = 1; int result--; //result = 0</code> |
| ! | Toán tử phủ định | <code>int value = true; result = !value; //result – false</code> |

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.

Vd:

```
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 | Vd: int a = 5; int b = 6; |
|---------|-------------------|---------------------------------------------|
| | | int b = 6; |
| == | So sánh bằng | boolean result = a == b; //result =false |
| != | So sánh khác | boolean result = a != b; //result = true |
| > | Lớn hơn | boolean result = a > b; //result =false |
| >= | Lớn hơn hoặc bằng | boolean result = a >= b; //result =false |
| < | Nhỏ hơn | boolean result = a < b; //result =true |
| <= | Nhỏ hơn hoặc bằng | boolean result = a <= b; //result = true |

Toán tử logic

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

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 (nestedif)

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 khóa để dừng thực thi các câu lệnh ở phía sau, break là k bắt buộc

Default là từ khóa để quy định khối lệnh sẽ được thực thi nếu k có trường hợp nào ở các case là đúng, default là k 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 |

NOTE:

Java hỗ trợ nhiều kiểu dữ liệu khác nhau

Các câu lệnh điều khiển giúp điều hướng luồng thực thi của ứng dụng

[Bài đọc] JDK vs JRE vs JVM

(Nguồn: <https://www.guru99.com/difference-between-jdk-jre-jvm.html>)

JDK là gì?

JDK là một môi trường phát triển phần mềm được sử dụng để tạo các applet và ứng dụng Java. Hình thức đầy đủ của JDK là Bộ công cụ phát triển Java. Các nhà phát triển Java có thể sử dụng nó trên Windows, macOS, Solaris và Linux. JDK giúp họ viết mã và chạy các chương trình Java. Có thể cài đặt nhiều phiên bản JDK trên cùng một máy tính.

JRE là gì?

JRE là một phần của phần mềm được thiết kế để chạy phần mềm khác. Nó chứa các thư viện lớp (class libraries), lớp bộ nạp (loader class) và JVM. Nói một cách dễ hiểu, nếu bạn muốn chạy chương trình Java, bạn cần có JRE. Nếu bạn không phải là lập trình viên, bạn không cần cài đặt JDK mà chỉ cần JRE để chạy các chương trình Java. Mặc dù vậy, tất cả các phiên bản JDK đều đi kèm với Môi trường thời gian chạy Java (JRE - Java Runtime Environment), vì vậy bạn không cần tải xuống và cài đặt JRE riêng biệt trong PC của mình. Dạng đầy đủ của JRE là Java Runtime Environment.

JVM là gì?

JVM là một công cụ cung cấp môi trường thời gian chạy để điều khiển mã Java hoặc các ứng dụng. Nó chuyển đổi Java bytecode thành ngôn ngữ máy. JVM là một phần của Java Runtime Environment (JRE). Nó không thể được tải xuống và cài đặt riêng biệt. Để cài đặt JVM, bạn cần cài đặt JRE. Hình thức đầy đủ của JVM là Máy ảo Java.

Trong nhiều ngôn ngữ lập trình khác, trình biên dịch tạo ra mã máy cho một hệ thống cụ thể. Tuy nhiên, trình biên dịch Java tạo ra mã cho một máy ảo được gọi là JVM.

Sự khác biệt giữa JDK vs JRE vs JVM

JDK là một bộ phát triển phần mềm trong khi JRE là một gói phần mềm cho phép chương trình Java chạy, trong khi JVM là một môi trường để thực thi byte code.

Dạng đầy đủ của JDK là Java Development Kit, trong khi dạng đầy đủ của JRE là Java Runtime Environment, trong khi dạng đầy đủ của JVM là Java Virtual Machine.

JDK phụ thuộc vào nền tảng, JRE cũng phụ thuộc vào nền tảng, nhưng JVM không phụ thuộc vào nền tảng.

JDK chứa các công cụ để phát triển, gỡ lỗi, v.v. JRE chứa các thư viện lớp và các tệp hỗ trợ khác, trong khi các công cụ phát triển phần mềm không được bao gồm trong JVM.

JDK đi kèm với trình cài đặt, mặt khác, JRE chỉ chứa môi trường để thực thi mã nguồn trong khi JVM được đóng gói trong cả phần mềm JDK và JRE.

Tại sao sử dụng JDK?

Dưới đây là những lý do quan trọng của việc sử dụng JDK:

JDK chứa các công cụ cần thiết để viết các chương trình Java và JRE để thực thi chúng.

Nó bao gồm một trình biên dịch, trình khởi chạy ứng dụng Java, Appletviewer, v.v.

Trình biên dịch chuyển đổi mã được viết bằng Java thành byte code.

Trình khởi chạy ứng dụng Java mở một JRE, tải lớp cần thiết và thực thi phương thức chính của nó.

Tại sao sử dụng JRE?

Dưới đây là những lý do quan trọng của việc sử dụng JRE:

JRE chứa các thư viện lớp, JVM và các tệp hỗ trợ khác. Nó không chứa bất kỳ công cụ nào để phát triển Java như trình gỡ lỗi, trình biên dịch, v.v.

Nó sử dụng các lớp gói quan trọng như thư viện math, swingetc, use, lang, awt và runtime.

Nếu bạn phải chạy các ứng dụng Java, thì JRE phải được cài đặt trong hệ thống của bạn.

Tại sao sử dụng JVM?

Dưới đây là những lý do quan trọng của việc sử dụng JVM:

JVM cung cấp một cách thực thi mã nguồn Java độc lập với nền tảng.

Nó có nhiều thư viện (libraries), công cụ (tools) và khuôn khổ (frameworks).

Khi bạn chạy chương trình Java, bạn có thể chạy trên bất kỳ nền tảng nào và tiết kiệm rất nhiều thời gian.

JVM đi kèm với trình biên dịch JIT (Just-in-Time) giúp chuyển đổi mã nguồn Java thành ngôn ngữ máy cấp thấp. Do đó, nó chạy nhanh hơn như một ứng dụng thông thường.

Các tính năng của JDK

Dưới đây là các tính năng quan trọng của JDK:

Nó cho phép bạn xử lý nhiều tiện ích mở rộng trong một khối bắt duy nhất.

JDK bao gồm tất cả các tính năng mà JRE có.

Nó chứa các công cụ phát triển như trình biên dịch, trình gỡ lỗi, v.v.

JDK cung cấp môi trường để phát triển và thực thi mã nguồn Java.

Nó có thể được cài đặt trên hệ điều hành Windows, Unix và Mac.

Toán tử Diamond có thể được sử dụng để chỉ định một giao diện kiểu chung thay vì viết chính xác.

Các tính năng của JRE

Dưới đây là các tính năng quan trọng của JRE:

Java Runtime Environment là một bộ công cụ sử dụng JVM thực sự chạy.

JRE chứa công nghệ triển khai, bao gồm Java Web Start và Java Plug-in.

Các nhà phát triển có thể dễ dàng chạy mã nguồn trong JRE, nhưng họ không thể viết và biên dịch chương trình Java.

Nó bao gồm các thư viện tích hợp như Java Database Connectivity (JDBC), Remote Method Invocation (RMI), Đặt tên Java và Giao diện Thư mục (JNDI), v.v.

JRE có ứng dụng khách máy ảo JVM và Java HotSpot.

Các tính năng của JVM

Dưới đây là các tính năng quan trọng của JVM:

Nó cho phép bạn chạy các ứng dụng trong môi trường đám mây hoặc trong thiết bị của bạn.

Máy ảo Java (JVM) chuyển đổi byte code thành mã dành riêng cho máy (machine-specific code).

Nó cung cấp các chức năng java cơ bản như quản lý bộ nhớ, bảo mật, thu gom rác và hơn thế nữa.

JVM chạy chương trình bằng cách sử dụng các thư viện và tệp do Java Runtime Environment cung cấp.

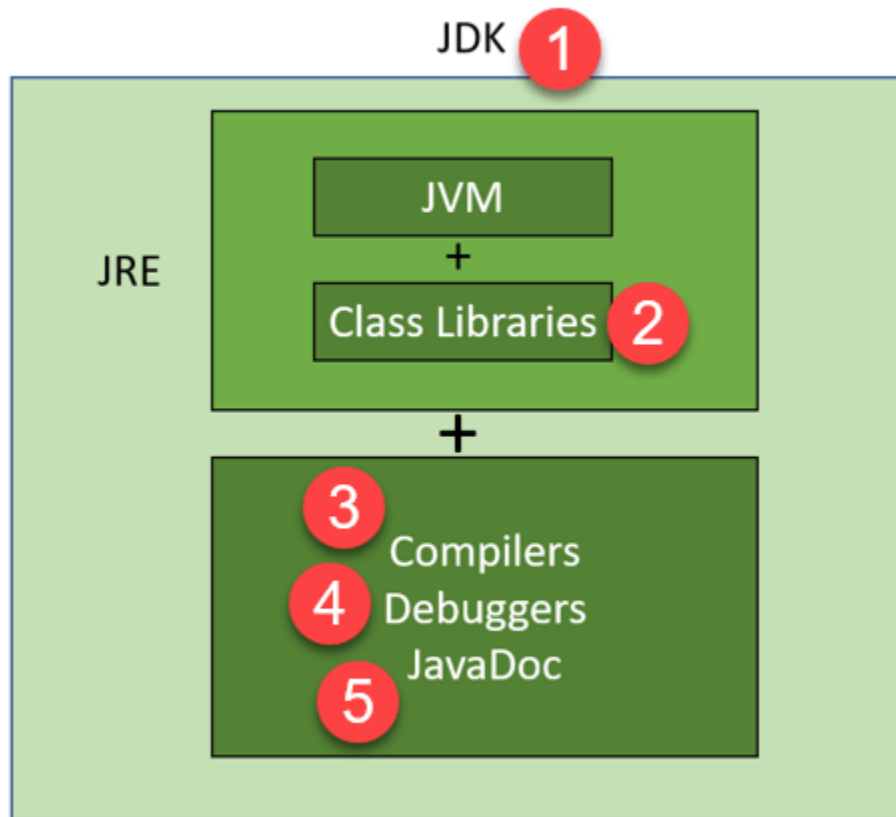
JDK và JRE đều chứa Máy ảo Java.

Nó có thể thực thi từng dòng chương trình java do đó nó còn được gọi là trình thông dịch.

JVM có thể dễ dàng tùy chỉnh, ví dụ, bạn có thể phân bổ bộ nhớ tối thiểu và tối đa cho nó.

Nó độc lập với phần cứng và hệ điều hành. Vì vậy, bạn có thể viết một chương trình java một lần và chạy ở bất cứ đâu.

Chức năng của JDK như thế nào?



JDK Functionality

Dưới đây là các thành phần quan trọng của JDK:

JDK và JRE: JDK cho phép các lập trình viên tạo các chương trình Java cốt lõi có thể được chạy bởi JRE, bao gồm JVM và các thư viện lớp (Class Libraries).

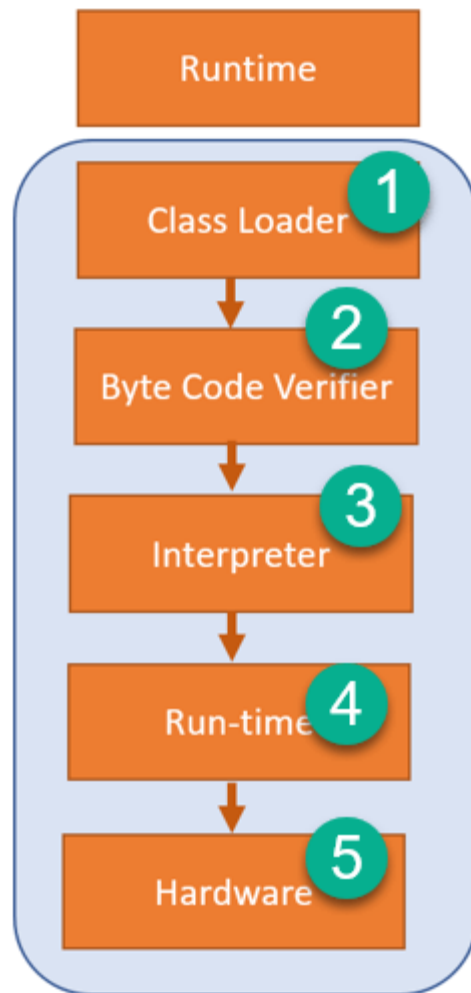
Thư viện lớp (Class Libraries): Nó là một nhóm các thư viện có thể tải động mà chương trình Java có thể gọi vào lúc chạy.

Trình biên dịch (Compilers): Nó là một chương trình Java chấp nhận tệp văn bản của các nhà phát triển và biên dịch thành tệp lớp Java (java Class). Đây là dạng đầu ra phổ biến do trình biên dịch cung cấp, chứa Java Byte Code. Trong Java, trình biên dịch chính là Javac.

Trình gỡ lỗi (Debuggers): Trình gỡ lỗi là một chương trình Java cho phép các nhà phát triển kiểm tra và gỡ lỗi các chương trình Java.

JavaDoc: JavaDoc là tài liệu được tạo bởi Sun Microsystems cho Java. JavaDoc có thể được sử dụng để tạo tài liệu API trong tệp HTML từ chương trình nguồn

Chức năng của JRE như thế nào?



JRE Functionality

JRE có một phiên bản của JVM với nó, các lớp thư viện và các công cụ phát triển. Khi bạn viết và biên dịch mã Java, trình biên dịch sẽ tạo ra một tệp lớp (java file) có byte code.

Dưới đây là các thành phần quan trọng của JRE:

Bộ tải lớp(Class Loader) : Bộ nạp lớp tải các lớp khác nhau cần thiết để chạy một chương trình Java. JVM sử dụng ba trình nạp lớp được gọi là trình nạp lớp bootstrap, trình nạp lớp mở rộng và trình nạp lớp hệ thống.

Trình xác minh byte code (Byte Code Verifier): Trình xác minh byte code xác minh byte code để mã không làm phiên trình thông dịch.

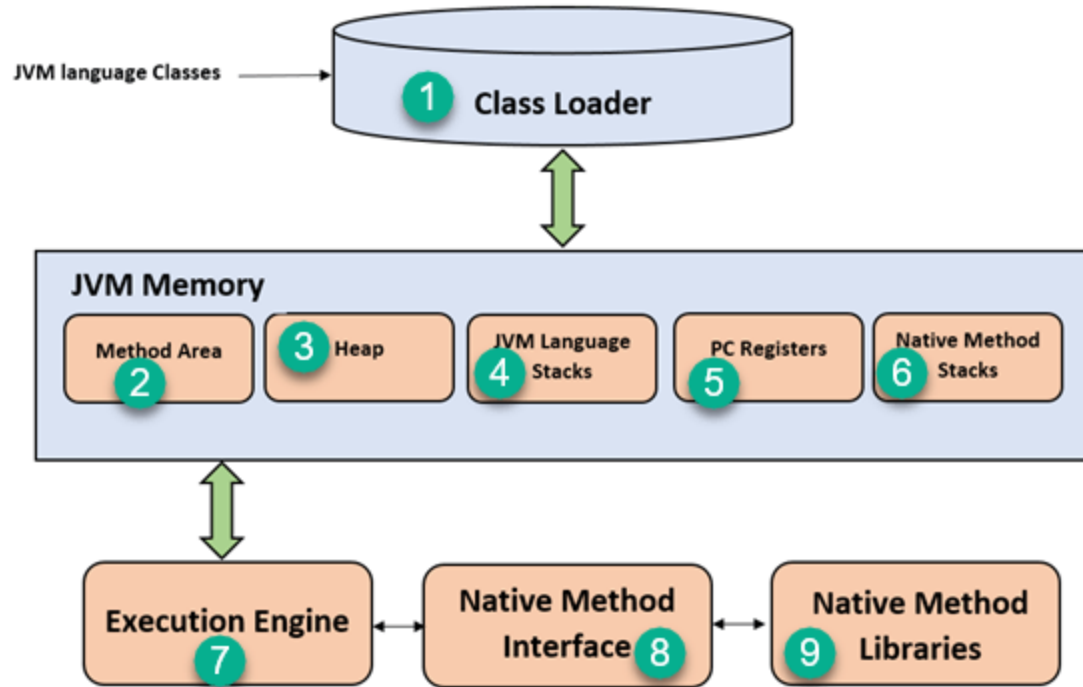
Trình thông dịch(Interpreter): Khi các lớp được tải và mã được xác minh, trình thông dịch sẽ đọc từng dòng mã.

Thời gian chạy(Run-time): Thời gian chạy là một hệ thống được sử dụng chủ yếu trong lập trình để mô tả khoảng thời gian mà một chương trình cụ thể đang chạy.

Phần cứng(Hardware): Sau khi bạn biên dịch mã gốc Java, nó sẽ chạy trên một nền tảng phần cứng cụ thể.

Bằng cách này, chương trình Java chạy trong JRE.

Chức năng của JVM như thế nào?



JVM functionality

Dưới đây là các thành phần quan trọng của JVM:

1) Bộ tải lớp (Class Loader)

Bộ tải lớp là một hệ thống con được sử dụng để tải các tệp lớp. Nó thực hiện ba chức năng chính: tải, liên kết và khởi tạo.

2) Vùng phương thức (Method Area)

Vùng phương thức JVM lưu trữ cấu trúc của lớp như siêu dữ liệu, mã cho các phương thức Java và constant runtime pool

3) Heap

Tất cả các đối tượng, mảng và biến khởi tạo được lưu trữ trong một heap. Bộ nhớ này được chia sẻ trên nhiều luồng.

4) Ngăn xếp ngôn ngữ JVM (JVM Language Stack)

Ngăn xếp lưu trữ các biến cục bộ và các kết quả thành phần của nó. Mỗi và mọi luồng đều có ngăn xếp ngôn ngữ JVM của riêng nó, được tạo đồng thời khi luồng được tạo. Khung mới được tạo khi phương thức được gọi và nó bị xóa khi quá trình gọi phương thức hoàn tất.

5) Đăng ký PC (PC Registers)

Thanh ghi PC lưu trữ địa chỉ của lệnh máy ảo Java, lệnh này hiện đang thực thi. Trong Java, mỗi luồng có một thanh ghi PC riêng biệt.

6) Ngăn xếp phương thức gốc (Native Method Stacks)

Các ngăn xếp phương thức gốc giữ lệnh của mã gốc phụ thuộc vào thư viện gốc. Nó phân bổ bộ nhớ trên các heap gốc hoặc sử dụng bất kỳ loại ngăn xếp nào.

7) Công cụ thực thi (Execution Engine)

Nó là một loại phần mềm được sử dụng để kiểm tra phần mềm, phần cứng hoặc hệ thống hoàn chỉnh. Công cụ thực thi thử nghiệm không bao giờ mang bất kỳ thông tin nào về sản phẩm được thử nghiệm.

8) Native Method Interface

Đây là một khung lập trình. Nó cho phép mã Java, đang chạy trong JVM để gọi bởi các thư viện và ứng dụng gốc.

9) Native Method Libraries

Đây là một tập hợp các Thư viện (C, C ++), được Công cụ thực thi cần.

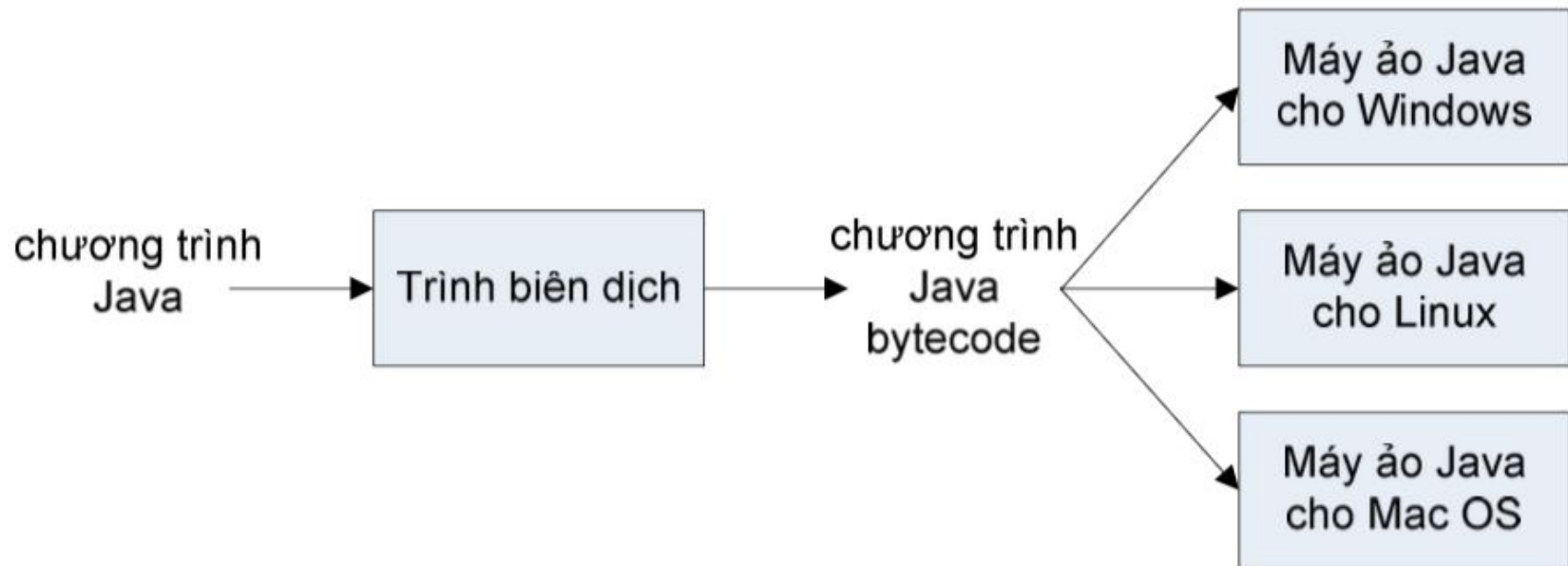
Ngôn ngữ máy bao gồm những chỉ thị (instruction) rất đơn giản mà CPU máy tính có thể thực hiện trực tiếp. Tuy nhiên, hầu hết các chương trình đều được viết bằng các ngôn ngữ lập trình bậc cao như Java hay C++. Một chương trình viết bằng ngôn ngữ bậc cao cần được dịch sang ngôn ngữ máy trước khi có thể được chạy trên máy tính. Việc dịch này do trình biên dịch thực hiện. Để chạy trên các loại máy tính với các ngôn ngữ máy khác nhau, cần đến các trình biên dịch phù hợp với loại ngôn ngữ máy đó.

Có một lựa chọn khác thay vì biên dịch chương trình viết bằng ngôn ngữ bậc cao. Thay vì dùng một trình biên dịch để dịch thẳng toàn bộ chương trình, ta có thể dùng một trình thông dịch, nó dịch từng chỉ thị một và chỉ dịch khi cần đến. Một trình thông dịch là một chương trình hoạt động gần như một CPU với một dạng chu trình nạp-và-thực-thi (fetch-and-execute). Để thực thi một chương trình, trình thông dịch lặp đi lặp lại chuỗi công việc: đọc một chỉ thị từ trong chương trình, xác định xem cần làm gì để thực hiện chỉ thị đó, và rồi thực hiện các lệnh mà máy thích hợp để thực hiện chỉ thị đó.

Một công dụng của trình thông dịch là để thực thi các chương trình viết bằng ngôn ngữ bậc cao, chẳng hạn như ngôn ngữ Lisp. Công dụng thứ hai là chúng cho phép ta chạy một chương trình ngôn ngữ máy dành cho một loại máy tính này trên một loại máy tính hoàn toàn khác. Ví dụ, có một chương trình tên là "Virtual PC" chạy trên các máy tính cài hệ điều hành Mac OS, đó là một trình thông dịch thực thi các chương trình mà máy viết cho các máy tính tương thích IBM PC. Nếu ta chạy "Virtual PC" trên một máy Mac OS, ta có thể chạy bất cứ chương trình PC nào, trong đó có cả các chương trình viết cho Windows.

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, nhưng đây là loại ngôn ngữ máy dành cho loại máy tính không tồn tại – loại máy "ảo" này được gọi là Máy ảo Java (Java Virtual Machine – JVM). Ngôn ngữ máy dành cho máy ảo Java được gọi là Java bytecode, hay ngắn gọn là bytecode. Để chạy được các chương trình Java trên một loại máy tính bất kỳ, người ta chỉ cần một trình thông dịch dành cho Java bytecode, trình thông dịch này giả lập máy ảo Java theo kiểu mà Virtual PC giả lập một máy tính PC. Máy ảo Java cũng chính là tên gọi dành cho trình thông dịch bytecode thực hiện nhiệm vụ giả lập, do đó ta nói rằng một

máy tính cần một máy ảo Java để chạy các chương trình Java.



Tất nhiên, mỗi loại máy tính cần một trình thông dịch Java bytecode khác, nhưng một khi đã có một trình thông dịch như vậy, nó có thể chạy một chương trình Java bytecode bất kì. Và cũng chính chương trình Java bytecode đó có thể chạy trên bất cứ máy tính nào có một trình thông dịch Java bytecode. Đây chính là một trong các đặc điểm quan trọng của Java: một chương trình sau khi biên dịch có thể chạy trên nhiều loại máy tính khác nhau.

Tại sao nên dùng mã trung gian là Java byte code

Có nhiều lý do tại sao nên dùng mã trung gian là Java bytecode thay cho việc phân phát mã nguồn chương trình Java và để cho mỗi người tự biên dịch nó sang mã máy của máy tính họ đang dùng. Thứ nhất, trình biên dịch là một chương trình phức tạp trong khi trình thông dịch chỉ là một chương trình nhỏ và đơn giản. Viết một trình thông dịch cho một loại máy tính mới dễ hơn là viết một trình biên dịch. Thứ hai, nhiều chương trình Java cần được tải xuống từ mạng máy tính. Việc này dẫn đến các mối quan tâm về bảo mật: ta không muốn tải về và chạy một chương trình sẽ phá hoại máy tính hoặc các file

trong máy tính của ta. Trình thông dịch bytecode hoạt động với vai trò bộ đệm giữa máy tính của ta và chương trình ta tải về. Nó có thể bảo vệ ta khỏi các hành động nguy hiểm tiềm tàng của chương trình đó.

Khi Java còn là một ngôn ngữ mới, nó đã bị chỉ trích là chạy chậm. Do Java bytecode được thực thi bởi một trình thông dịch, có vẻ như các chương trình bytecode không bao giờ có thể chạy nhanh bằng các chương trình đã được biên dịch ra ngôn ngữ máy của chính máy tính mà chương trình đang chạy trên đó. Tuy nhiên, vấn đề này đã được giải quyết gần như toàn bộ bằng việc sử dụng trình biên dịch JIT (just-in-time compiler) cho việc thực thi Java bytecode. Trình biên dịch JIT dịch Java bytecode thành mã máy. Nó làm việc này trong khi thực thi chương trình. Cũng như một trình thông dịch thông thường, đầu vào cho một trình biên dịch JIT là một chương trình Java bytecode, và nhiệm vụ của nó là thực thi chương trình đó. Nhưng trong khi thực thi chương trình, nó dịch một phần của chương trình ra mã máy. Những phần được biên dịch này khi đó có thể được thực thi nhanh hơn là so với khi chúng được thông dịch. Do một phần của chương trình thường được thực thi nhiều lần trong khi chương trình chạy, một trình biên dịch JIT có thể cải thiện đáng kể tổng thời gian chạy của chương trình.

Sự khác biệt giữa JDK, JRE và JVM

Dưới đây là những điểm khác biệt chính giữa JDK so với JRE và JVM:

| JDK | JRE | JVM |
|---------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| Hình thức đầy đủ của JDK là Java Development Kit | Dạng đầy đủ của JRE là Java Runtime Environment. | Hình thức đầy đủ của JVM là Java Virtual Machine |
| JDK là một bộ công cụ phát triển phần mềm để phát triển các ứng dụng bằng Java. | Nó là một gói phần mềm cung cấp các thư viện lớp Java với các thành phần cần thiết để chạy mã Java. | JVM thực thi mã byte Java và cung cấp môi trường để thực thi nó. |
| JDK phụ thuộc vào nền tảng. | JRE cũng phụ thuộc vào nền tảng. | JVM không phụ thuộc nhiều vào nền tảng. |
| Nó chứa các công cụ để phát triển, gỡ lỗi và giám sát mã java. | Nó chứa các thư viện lớp và các tệp hỗ trợ khác mà JVM yêu cầu để thực thi chương trình. | Các công cụ phát triển phần mềm không được bao gồm trong JVM. |
| Nó là tập cha của JRE | Nó là tập con của JDK. | JVM là một tập con của JRE. |

| | | |
|--------------------------------------------------------------------------------------------------------|-----------------------------------------------|---------------------------------------------------|
| JDK cho phép các nhà phát triển tạo các chương trình Java có thể được thực thi và chạy bởi JRE và JVM. | JRE là một phần của Java tạo ra JVM. | Nó là thành phần nền tảng Java thực thi mã nguồn. |
| JDK đi kèm với trình cài đặt. | JRE chỉ chứa môi trường để thực thi mã nguồn. | JVM được đóng gói trong cả phần mềm JDK và JRE. |

[Bài đọc] Một chương trình Java đơn giản

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 12)

Một chương trình Java được thực thi từ phương thức **main** trong lớp.


Hãy bắt đầu với một chương trình Java đơn giản hiển thị thông báo **Welcome to Java!** trên màn hình console. (Console là một thuật ngữ máy tính dùng để chỉ việc nhập và hiển thị văn bản trên thiết bị máy tính, có nghĩa là nhận đầu vào từ bàn phím và hiển thị đầu ra trên màn hình.) Chương trình ví dụ như sau:

LISTING 1.1 Welcome.java

```

1 public class Welcome {
2     public static void main(String[] args) {
3         // Display message Welcome to Java! on the console
4         System.out.println("Welcome to Java!");
5     }
6 }

```



Welcome to Java!

Lưu ý rằng số dòng chỉ dành cho mục đích tham khảo; chúng không phải là một phần của chương trình. Vì vậy, đừng nhập số dòng trong chương trình của bạn.

Dòng 1 xác định một lớp. Mỗi chương trình Java phải có ít nhất một lớp. Mỗi lớp có một Tên. Theo quy ước, tên lớp bắt đầu bằng một chữ cái viết hoa. Trong ví dụ này, lớp tên là **Welcome**

Dòng 2 xác định phương thức **main**. Chương trình được thực thi từ phương thức **main**. Một lớp có thể chứa một số phương thức. Phương thức **main** là (entry point), nơi chương trình bắt đầu được thực thi.

Phương thức là một cấu trúc có chứa các câu lệnh. Phương thức **main** trong chương trình này chứa câu lệnh `System.out.println(...)`. Câu lệnh này hiển thị chuỗi `Welcome to Java!` trên màn hình console (dòng 4). Chuỗi là một thuật ngữ lập trình có nghĩa là một chuỗi các ký tự. Một chuỗi phải được đặt trong dấu ngoặc kép. Mọi câu lệnh trong Java đều kết thúc bằng dấu chấm phẩy (;), được gọi là dấu chấm câu lệnh.

Các từ khóa (keywords) có ý nghĩa cụ thể đối với trình biên dịch và không thể được sử dụng cho các mục đích khác trong chương trình. Ví dụ: khi trình biên dịch nhìn thấy từ **class**, nó hiểu rằng từ sau **class** là tên của lớp. Các từ khóa khác trong chương trình này là `public`, `static`, và `void`.

Dòng 3 là một chú thích ghi lại chương trình là gì và nó được xây dựng như thế nào. Chú thích giúp lập trình viên có thể giao tiếp và hiểu chương trình. Trình biên dịch sẽ bỏ qua chú thích. Trong Java, các chú thích được đứng trước hai dấu gạch chéo (//) trên một dòng, được gọi là chú thích dòng hoặc được đặt giữa `/*` và `*/` trên một hoặc một số dòng, được gọi là chú thích khối hoặc chú thích đoạn. Khi trình biên dịch nhìn thấy //, nó sẽ bỏ qua tất cả văn bản sau // trên cùng một dòng. Khi nó nhìn thấy `/*`, nó sẽ quét `*/` tiếp theo và bỏ qua bất kỳ văn bản nào giữa `/*` và `*/`. Dưới đây là các ví dụ về chú thích:

```
// This application program displays Welcome to Java!  
/* This application program displays Welcome to Java! */  
/* This application program  
displays Welcome to Java! */
```

Một cặp dấu ngoặc nhọn trong chương trình tạo thành một khối, nhóm các thành phần của chương trình.

Trong Java, mỗi khối bắt đầu bằng dấu ngoặc nhọn mở (`{`) và kết thúc bằng dấu ngoặc nhọn đóng (`}`). Mỗi lớp có một khối lớp, nhóm dữ liệu và phương thức của lớp. Tương tự, mọi phương thức có một khối phương thức, nhóm các câu lệnh trong phương thức. Các khối có thể được lồng vào nhau, có nghĩa là một khối có thể được đặt trong một khối khác, như được hiển thị trong đoạn mã sau:

```

public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}

```

Bạn đã thấy một số ký tự đặc biệt (ví dụ: {}, //,;) trong chương trình. Chúng được sử dụng trong hầu hết mọi chương trình. Bảng 1.2 tóm tắt công dụng của chúng. Các lỗi phổ biến nhất bạn sẽ mắc phải khi học lập trình sẽ là lỗi cú pháp. Giống như bất kỳ ngôn ngữ lập trình nào, Java có cú pháp riêng và bạn cần tuân theo các quy tắc cú pháp. Nếu chương trình của bạn vi phạm quy tắc — ví dụ: nếu dấu chấm phẩy bị thiếu, thiếu dấu ngoặc nhọn, thiếu dấu ngoặc kép hoặc một từ bị sai chính tả, trình biên dịch Java sẽ báo lỗi cú pháp. Hãy thử biên dịch chương trình với những lỗi này và xem những gì trình biên dịch báo cáo.

| Kí tự | Tên | Mô tả |
|-------|---------------------------|------------------------------------------|
| { } | Dấu ngoặc nhọn mở và đóng | Biểu thị một khối để gom các câu lệnh |
| () | Mở và đóng ngoặc đơn | Được sử dụng với các phương thức. |
| [] | Mở và đóng ngoặc nhọn | Biểu thị một mảng. |
| // | Dấu gạch chéo kép | Đặt trước một dòng nhận xét. |
| " " | Dấu ngoặc kép mở và đóng | Định kèm một chuỗi (tức là chuỗi ký tự). |
| ; | Dấu chấm phẩy | Đánh dấu phần cuối của một câu lệnh. |

Bảng 1.2 Ký tự đặc biệt

[Bài đọc] Tạo, biên dịch và thực thi một chương trình Java

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 15)

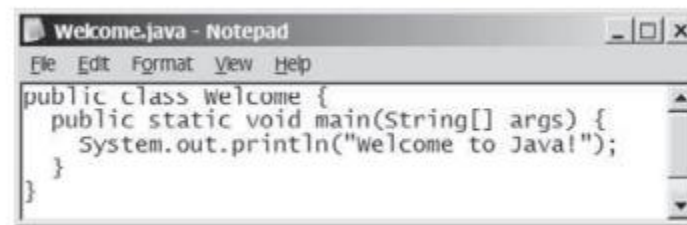
Bạn lưu một chương trình Java trong tệp .java và biên dịch nó thành tệp .class. Tệp .class được thực thi bởi Máy ảo Java (JVM).

Bạn phải tạo chương trình của mình và biên dịch nó trước khi nó có thể được thực thi. Quá trình này là lặp đi lặp lại. Nếu chương trình của bạn có lỗi biên dịch (compile errors), bạn phải sửa đổi chương trình để sửa chữa chúng, và sau đó biên

dịch lại nó. Nếu chương trình của bạn có lỗi thời gian chạy (runtime errors) hoặc không tạo ra kết quả chính xác, bạn phải sửa đổi chương trình, biên dịch lại và thực thi lại.

Bạn có thể sử dụng bất kỳ trình soạn thảo văn bản hoặc IDE nào để tạo và chỉnh sửa tệp mã nguồn Java. Phần này trình bày cách tạo, biên dịch và chạy các chương trình Java từ cửa sổ lệnh (command window).

Từ cửa sổ lệnh, bạn có thể sử dụng trình soạn thảo văn bản như Notepad để tạo tệp mã nguồn Java, như trong hình sau đây:



Hình: Bạn có thể tạo tệp mã nguồn Java bằng Notepad.

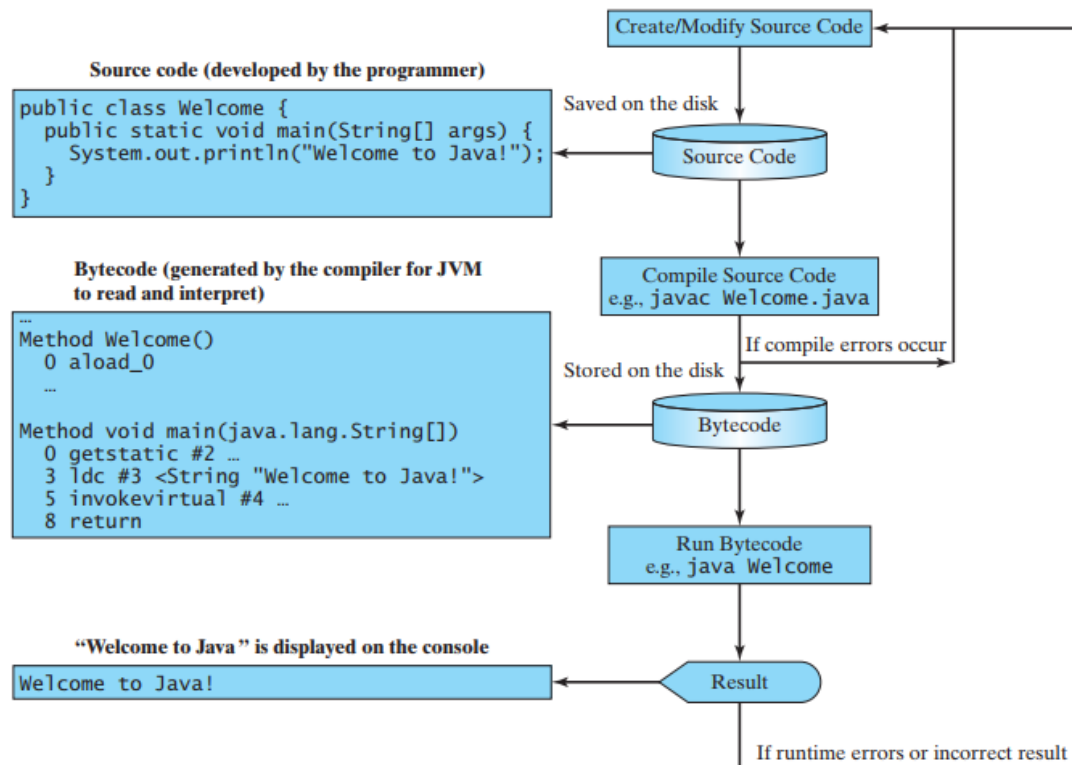
Ghi chú:

Tệp chứa mã nguồn phải kết thúc bằng phần mở rộng .java và phải có cùng tên với tên lớp công khai (public class). Ví dụ: tệp chứa mã nguồn phải được đặt tên là Welcome.java, vì tên lớp công khai là Welcome.

Trình biên dịch Java dịch tệp mã nguồn Java thành tệp Java Byte Code. Lệnh sau biên dịch Welcome.java:

javac Welcome.java

Nếu không có bất kỳ lỗi cú pháp nào, trình biên dịch sẽ tạo tệp bytecode với phần mở rộng .class. Do đó, lệnh “javac Welcome.java” tạo một tệp có tên là Welcome.class.



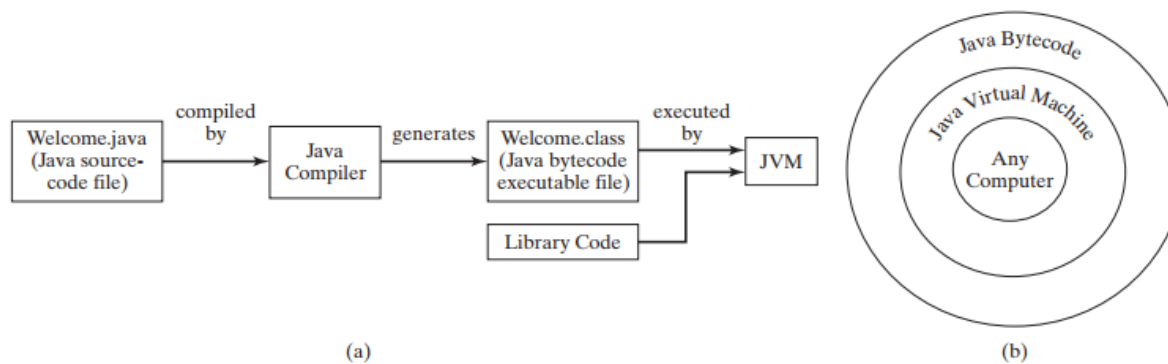
Hình: Quá trình phát triển chương trình Java bao gồm nhiều lần tạo / sửa đổi mã nguồn, biên dịch, và thực thi các chương trình.

Ngôn ngữ Java là một ngôn ngữ bậc cao, nhưng Java bytecode là một ngôn ngữ bậc thấp. Bytecode tương tự như lệnh máy nhưng là kiến trúc trung lập và có thể chạy trên bất kỳ nền tảng nào có Máy ảo Java (JVM), máy ảo là một chương trình thông dịch mã bytecode của Java. Cái này là một trong những lợi thế chính của Java: Mã bytecode của Java có thể chạy trên nhiều nền tảng phần cứng và hệ điều hành khác nhau. Mã nguồn Java được biên dịch thành Java bytecode và Java bytecode được thông dịch bởi JVM. Mã Java của bạn có thể sử dụng mã trong thư viện Java. JVM thực thi mã của bạn cùng với mã trong thư viện.

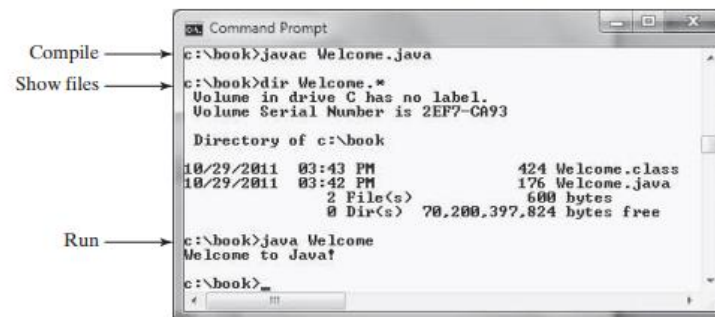
Để thực thi một chương trình Java là chạy mã bytecode của chương trình. Bạn có thể thực thi mã bytecode trên bất kỳ nền tảng nào có JVM, là một trình thông dịch. Nó dịch các hướng dẫn riêng lẻ trong mã bytecode thành ngôn ngữ máy đích tại một thời điểm thay vì toàn bộ chương trình như một đơn vị duy nhất. Mỗi bước được thực hiện ngay sau khi nó được dịch.

Lệnh sau chạy mã bytecode:

java Welcome



Hình: (a) Mã nguồn Java được dịch thành mã bytecode. (b) Mã bytecode của Java có thể được thực thi trên bất kỳ máy tính nào có Máy ảo Java.



Hình: Lệnh javac để biên dịch Welcome.java. Trình biên dịch tạo tệp Welcome.class và tệp này được thực thi bằng lệnh java.

[Bài đọc] Phong cách lập trình và tài liệu

(Nguồn: Introduction to Java Programming, Comprehensive Version (10th Edition) - Y. Daniel Liang – Tr 18)

Phong cách lập trình tốt và tài liệu thích hợp làm cho một chương trình dễ đọc và giúp người lập trình phòng tránh lỗi.

Phong cách lập trình giải quyết các chương trình trông như thế nào. Một chương trình có thể biên dịch và chạy đúng cách ngay cả khi chỉ viết trên một dòng, nhưng viết tất cả trên một dòng sẽ là kiểu lập trình tồi vì khó đọc. Tài liệu là chú thích liên quan đến một chương trình. Phong cách lập trình và tài liệu là quan trọng như mã hóa. Phong cách lập trình tốt và tài liệu thích hợp làm giảm khả năng xảy ra lỗi và làm cho chương trình dễ đọc.

Các kiểu chú thích thích hợp

Thêm vào một bản tóm tắt ở đầu chương trình giải thích những gì chương trình làm, các tính năng chính và bất kỳ kỹ thuật độc đáo nào mà nó sử dụng. Trong một chương trình dài, bạn cũng nên thêm chú thích giới thiệu từng bước chính và giải thích bất cứ điều gì khó đọc. Điều quan trọng là đưa ra các chú thích ngắn gọn để chúng không lấn át chương trình hoặc gây khó đọc.

Ngoài chú thích dòng (bắt đầu bằng //) và chú thích khối (bắt đầu bằng /*), Java hỗ trợ các chú thích thuộc loại đặc biệt, được gọi là chú thích javadoc. Các chú thích javadoc bắt đầu bằng /** và kết thúc bằng */. Chúng có thể được trích xuất thành một tệp HTML bằng cách sử dụng lệnh javadoc của JDK.

Sử dụng chú thích javadoc (/ ** ... * /) để chú thích về toàn bộ lớp hoặc toàn bộ phương thức của lớp. Các chú thích này phải đứng trước lớp hoặc tiêu đề phương thức để được trích xuất

vào một tệp HTML javadoc. Để chú thích về các bước bên trong một phương thức, hãy sử dụng chú thích dòng (//).

Thụt lề và giãn cách thích hợp

Kiểu thực lệ nhất quán làm cho các chương trình rõ ràng và dễ đọc, dễ gỡ lỗi và bảo trì.

Thực lệ được sử dụng để minh họa mối quan hệ cấu trúc giữa các thành phần của chương trình

hoặc các câu lệnh. Java có thể đọc chương trình ngay cả khi tất cả các câu lệnh đều dài như nhau nhưng con người thấy dễ dàng hơn khi đọc và bảo trì mã nguồn được căn chỉnh đúng cách. Thực lệ từng thành phần con hoặc câu lệnh nhiều hơn ít nhất hai dấu cách so với cấu trúc mà nó nằm trong đó lồng vào nhau.

Một khoảng trắng duy nhất nên được thêm vào cả hai bên của toán tử nhị phân, như thể hiện trong câu lệnh sau:

`System.out.println(3+4*4);` Bad style

`System.out.println(3 + 4 * 4);` Good style

Kiểu khối

Một khối là một nhóm các câu lệnh được bao quanh bởi các dấu ngoặc nhọn. Có hai kiểu phổ biến, kiểu Next-line (dòng tiếp theo) và kiểu End-of-line (cuối dòng), như hình dưới đây:

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

Next-line style

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

End-of-line style

Kiểu next-line sắp xếp các dấu ngoặc nhọn theo chiều dọc và làm cho các chương trình dễ đọc, trong khi kiểu end-of-line giúp tiết kiệm dung lượng và có thể giúp tránh một số lỗi lập trình tinh vi. Cả hai đều là các kiểu khối có thể chấp nhận được. Sự lựa chọn phụ thuộc vào sở thích cá nhân hoặc tổ chức. Bạn nên sử dụng kiểu khối một cách nhất quán — không khuyến khích trộn kiểu.

Định dạng lại chương trình sau theo phong cách lập trình và tài liệu thích hợp. Sử dụng kiểu end-of-line:

```
public class Test
{
    // Main method
    public static void main(String[] args) {
        /** Display output */
        System.out.println("Welcome to Java");
    }
}
```

[Bài đọc] Lỗi lập trình

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 20)

Lỗi lập trình có thể được phân thành ba loại: lỗi cú pháp (syntax errors), lỗi thời gian chạy (runtime errors) và lỗi logic (logic errors).

Lỗi cú pháp

Các lỗi được phát hiện bởi trình biên dịch được gọi là lỗi cú pháp hoặc lỗi biên dịch. Cú pháp lỗi do lỗi trong quá trình xây dựng mã, chẳng hạn như nhập sai từ khóa, bỏ qua một số dấu câu cần thiết hoặc sử dụng dấu ngoặc nhọn mở mà không có dấu ngoặc nhọn đóng tương ứng.

Những lỗi này thường dễ phát hiện vì trình biên dịch cho bạn biết chúng ở đâu và điều gì đã gây ra chúng. Ví dụ: chương trình sau có lỗi cú pháp:

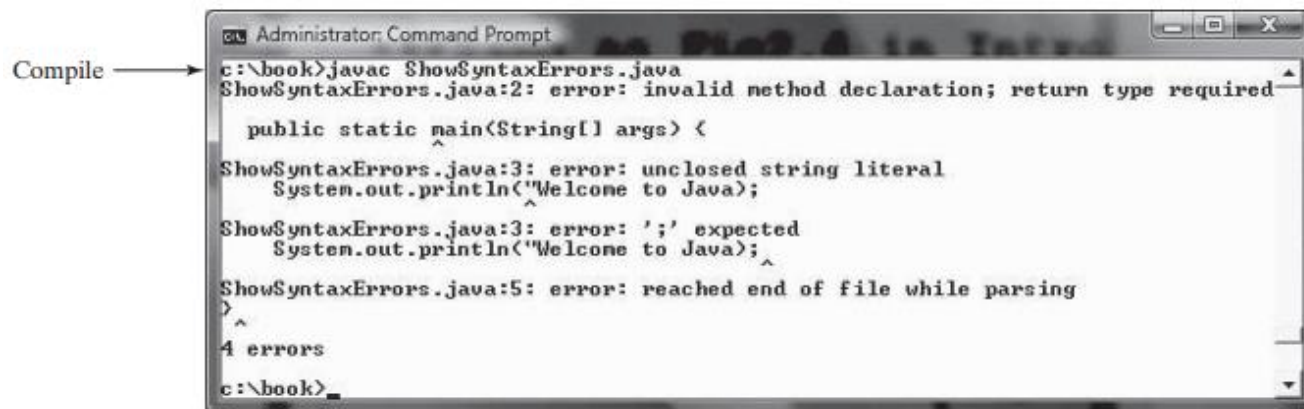
```
ShowSyntaxErrors.java
1 public class ShowSyntaxErrors {
```

```
2 public static main(String[] args) {  
3     System.out.println("Welcome to Java");  
4 }  
5 }
```

Bốn lỗi được báo cáo, nhưng chương trình thực sự có hai lỗi:

- Từ khóa `void` bị thiếu trước **main** ở dòng 2.
- Chuỗi “Welcome to Java” nên được đóng bằng dấu ngoặc kép ở dòng 3.

Vì một lỗi đơn lẻ thường sẽ hiển thị nhiều dòng lỗi biên dịch, nên cách thực hành tốt là sửa lỗi từ dòng trên và làm việc từ dòng trên xuống. Sửa các lỗi xảy ra trước đó trong chương trình cũng có thể sửa các lỗi bổ sung xảy ra sau này.



Hình: Trình biên dịch báo lỗi cú pháp

Mẹo:

Nếu bạn không biết cách sửa lỗi, hãy rà soát chương trình của bạn chặt chẽ, từng ký tự, với các ví dụ tương tự trong văn bản. Trong vài tuần đầu tiên, bạn sẽ có lẽ dành nhiều thời gian để sửa lỗi cú pháp. Bạn sẽ sớm làm quen với cú pháp Java và có thể nhanh chóng sửa lỗi cú pháp.

Lỗi thời gian chạy

Lỗi thời gian chạy là lỗi khiến chương trình kết thúc bất thường. Chúng xảy ra trong khi một chương trình đang chạy nếu môi trường phát hiện ra một hoạt động không thể thực hiện được.

Lỗi đầu vào thường gây ra lỗi thời gian chạy. Một lỗi đầu vào xảy ra khi chương trình chờ người dùng nhập giá trị nhưng người dùng nhập giá trị mà chương trình không xử lý được.

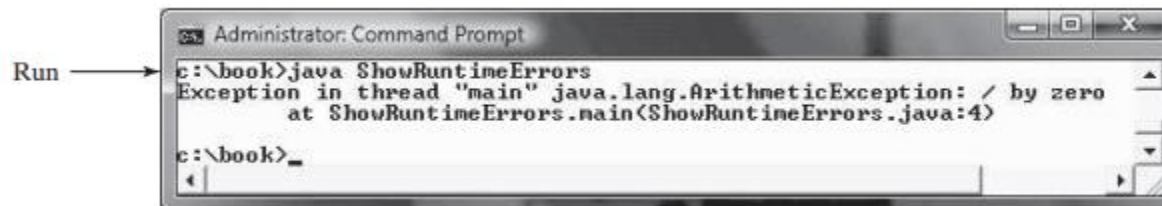
Ví dụ: nếu chương trình muốn đọc bằng một số, nhưng thay vào đó người dùng nhập một chuỗi, điều này gây ra lỗi kiểu dữ liệu xảy ra trong chương trình.

Một ví dụ khác về lỗi thời gian chạy là phép chia cho số không. Điều này xảy ra khi số chia là 0 cho phép chia số nguyên.

Ví dụ: chương trình sau sẽ gây ra lỗi thời gian chạy:

ShowRuntimeErrors.java

```
1 public class ShowRuntimeErrors {  
2     public static void main(String[] args) {  
3         System.out.println(1 / 0);  
4     }  
5 }
```




Hình: Lỗi thời gian chạy khiến chương trình kết thúc bất thường

Lỗi logic

Lỗi logic xảy ra khi một chương trình không thực hiện theo cách mà nó đã dự định. Lỗi của loại này xảy ra vì nhiều lý do khác nhau. Ví dụ: giả sử bạn đã viết chương trình sau để chuyển 35 độ C sang độ F:

ShowLogicErrors.java

```
1 public class ShowLogicErrors {  
2     public static void main(String[] args) {  
3         System.out.println("Celsius 35 is Fahrenheit degree ");  
4         System.out.println((9 / 5) * 35 + 32);  
5     }  
6 }
```



```
Celsius 35 is Fahrenheit degree  
67
```

Bạn sẽ nhận được 67 độ F, điều này là sai. Nó phải là 95.0. Trong Java, phép chia cho số nguyên là thương số, phần phân số bị cắt ngắn, vì vậy trong Java $9/5$ là 1. Để nhận được kết quả chính xác, bạn cần sử dụng $9.0 / 5$, mà kết quả là 1.8.

Nhìn chung, các lỗi cú pháp rất dễ tìm và dễ sửa vì trình biên dịch cho biết lỗi đến từ đâu và tại sao chúng sai. Lỗi thời gian chạy không khó tìm vì lý do và vị trí của lỗi được hiển thị trên màn hình console khi chương trình hủy bỏ. Mặt khác, việc tìm ra các lỗi logic có thể rất khó khăn. Bên trong các chương sắp tới, bạn sẽ học các kỹ thuật tìm lỗi logic.

Lỗi phổ biến

Thiếu dấu ngoặc nhọn, thiếu dấu chấm phẩy, thiếu dấu ngoặc kép cho chuỗi và sai chính tả là những lỗi phổ biến đối với các lập trình viên mới.

Lỗi phổ biến 1: Thiếu dấu ngoặc nhọn

Dấu ngoặc nhọn được sử dụng để biểu thị một khối trong chương trình. Mỗi dấu ngoặc nhọn mở phải được khớp với nhau bằng một dấu ngoặc nhọn đóng. Một lỗi phổ biến là thiếu dấu ngoặc nhọn. Để tránh lỗi này, hãy nhập dấu ngoặc nhọn đóng bất cứ khi nào một dấu ngoặc nhọn được nhập, như thể hiện trong ví dụ sau:

```
public class Welcome {
```

```
} ← Type this closing brace right away to match the opening brace
```

Lỗi phổ biến 2: Thiếu dấu chấm phẩy

Mỗi câu lệnh kết thúc bằng một dấu chấm dứt câu lệnh (;). Thông thường, một lập trình viên mới quên đặt một dấu chấm dứt câu lệnh cho câu lệnh cuối cùng trong một khối, như được hiển thị trong ví dụ sau:

```
public static void main(String[] args) {  
    System.out.println("Programming is fun!");  
    System.out.println("Fundamentals First");  
    System.out.println("Problem Driven")  
}
```

Missing a semicolon

Lỗi phổ biến 3: Thiếu dấu ngoặc kép

Một chuỗi phải được đặt bên trong dấu ngoặc kép. Thông thường, một lập trình viên mới quên đặt dấu ngoặc kép ở cuối chuỗi, như được hiển thị trong ví dụ sau:

```
System.out.println("Problem Driven ");
```

Missing a quotation mark

Lỗi phổ biến 4: Tên sai chính tả

Java phân biệt chữ hoa chữ thường. Tên sai chính tả là một lỗi phổ biến đối với các lập trình viên mới. Ví dụ: từ main bị sai chính tả thành Main và String bị sai chính tả thành main trong đoạn mã sau:


```
1 public class Test {  
2     public static void Main(string[] args) {  
3         System.out.println((10.5 + 2 * 3) / (45 - 3.5));  
4     }  
5 }
```

Xác định và sửa các lỗi trong đoạn mã sau:

```
1 public class Welcome {  
2     public void Main(String[] args) {  
3         System.out.println('Welcome to Java!');  
4     }  
5 }
```

[Bài đọc] Khai báo biến và kiểu dữ liệu

(Nguồn: *OCA Java SE 8 Programmer I Certification Guide – Tr 74*)

Có hai loại biến 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.

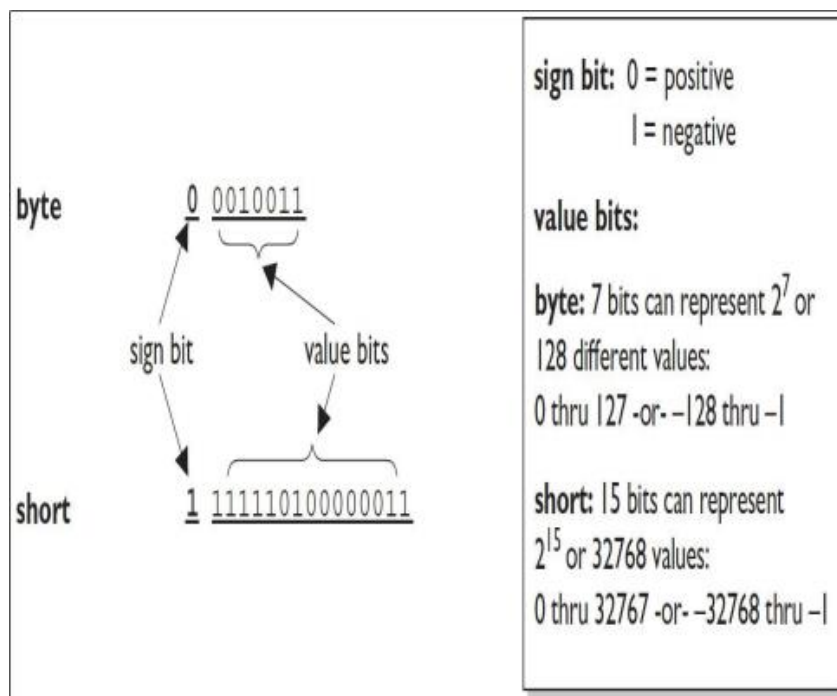
Các biến tham chiếu (Reference): Một biến tham chiếu (reference) được sử dụng để tham chiếu đến (hoặc truy cập) một đối tượng. Một biến tham chiếu được khai báo là một kiểu cụ thể và kiểu đó không bao giờ có thể thay đổi được. Một biến tham chiếu có thể được sử dụng để tham chiếu đến bất kỳ đối tượng nào thuộc kiểu đã khai báo hoặc kiểu con của kiểu khai báo (kiểu tương thích). Chúng ta sẽ nói nhiều hơn về việc sử dụng biến tham chiếu để tham chiếu khi chúng ta thảo luận về tính đa hình.

Khai báo biến nguyên thủy

Các biến nguyên thủy có thể được khai báo dưới dạng biến lớp (tĩnh), biến khởi tạo, tham số phương thức, hoặc các biến cục bộ. Bạn có thể khai báo một hoặc nhiều kiểu nguyên thủy, cùng kiểu nguyên thủy, trong một dòng duy nhất. Ví dụ:

Đối với các loại số nguyên, trình tự từ nhỏ đến lớn là byte, short, int và long, và số doubles là lớn hơn floats.

Bạn cũng cần biết rằng các kiểu số (cả kiểu số nguyên và dấu phẩy động) đều có dấu và điều đó ảnh hưởng đến phạm vi của chúng như thế nào. Trước tiên, hãy xem lại các khái niệm. Tất cả sáu kiểu số trong Java đều được tạo thành từ một số byte 8 bit nhất định và có dấu, nghĩa là chúng có thể là âm hoặc dương. Bit ngoài cùng bên trái (chữ số có nghĩa nhất) được sử dụng để biểu thị dấu hiệu, trong đó 1 có nghĩa là âm và 0 có nghĩa là dương. Phần còn lại của các bit đại diện cho giá trị.



Hình 1: Bit dấu cho một byte

| Type | Bits | Bytes | Minimum Range | Maximum Range |
|--------|------|-------|---------------|---------------|
| byte | 8 | 1 | -2^7 | $2^7 - 1$ |
| short | 16 | 2 | -2^{15} | $2^{15} - 1$ |
| int | 32 | 4 | -2^{31} | $2^{31} - 1$ |
| long | 64 | 8 | -2^{63} | $2^{63} - 1$ |
| float | 32 | 4 | n/a | n/a |
| double | 64 | 8 | n/a | n/a |

Hình 2: Dải của số nguyên thủy

Bảng trên cho thấy các loại biến nguyên thủy với kích thước và phạm vi của chúng. Hình 1 cho thấy rằng với một byte, có 256 số có thể (hoặc 2^8). Một nửa trong số này là số âm và một nửa là số dương. Các số dương nhỏ hơn một so với số âm vì số 0 được lưu trữ dưới dạng nhị phân dương. Sử dụng công thức $-2^{(\text{bit}-1)}$ để tính toán dải số âm và sử dụng $2^{(\text{bit}-1)} - 1$ cho dải số dương.

Không có dải cho giá trị boolean; một giá trị boolean chỉ có thể true hoặc false.

Kiểu char (một ký tự) chứa một ký tự Unicode 16 bit. Các ký tự Unicode thực sự được đại diện bởi số nguyên 16 bit không dấu, có nghĩa là 2^{16} giá trị có thể có, nằm trong khoảng từ 0 đến 65535 ($2^{16} - 1$).

Khai báo các biến tham chiếu

Các biến tham chiếu có thể được khai báo dưới dạng biến tĩnh, biến khởi tạo, tham số phương thức hoặc biến cục bộ. Bạn có thể khai báo một hoặc nhiều biến tham chiếu, cùng kiểu, trong một dòng duy nhất. Ví dụ:

```
Object o;  
Dog myNewDogReferenceVariable;  
String s1, s2, s3;           // declare three String vars.
```

[Bài đọc] Các toán tử số học và phép gán

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 41, 46, 50)

Câu lệnh gán

Một câu lệnh gán chỉ định một giá trị cho một biến. Một câu lệnh gán có thể được sử dụng như một biểu thức trong Java.

Sau khi một biến được khai báo, bạn có thể gán một giá trị cho nó bằng cách sử dụng một câu lệnh gán. Trong Java, dấu bằng (=) được sử dụng làm toán tử gán. Cú pháp cho các câu lệnh gán như sau:

biến = biểu thức;

Một biểu thức đại diện cho một phép tính liên quan đến các giá trị, biến và toán tử, kết hợp chúng với nhau, đánh giá một giá trị. Ví dụ, hãy xem xét đoạn mã sau:

```
int y = 1; // Assign 1 to variable y
double radius = 1.0; // Assign 1.0 to variable radius
int x = 5 * (3 / 2); // Assign the value of the expression to x
x = y + 1; // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159; // Compute area
```

Bạn có thể sử dụng một biến trong một biểu thức. Một biến cũng có thể được sử dụng trong cả hai phía của toán tử = . Ví dụ:

```
x = x + 1;
```

Trong câu lệnh gán này, kết quả của $x + 1$ được gán cho x . Nếu x là 1 trước khi câu lệnh được thực thi, thì nó sẽ trở thành 2 sau khi câu lệnh được thực hiện.

Để gán giá trị cho một biến, bạn phải đặt tên biến ở bên trái toán tử gán. Do đó, phát biểu sau đây là sai:

```
1 = x; // Sai lầm
```

Trong Java, một câu lệnh gán về cơ bản là một biểu thức đánh giá giá trị được gán cho biến ở phía bên trái của toán tử gán. Vì lý do này, một câu lệnh gán còn được gọi là biểu thức gán. Ví dụ như sau là đúng:

```
System.out.println (x = 1);
```

tương đương với

$x = 1;$

`System.out.println (x);`

Nếu một giá trị được gán cho nhiều biến, bạn có thể sử dụng cú pháp sau:

$i = j = k = 1;$

tương đương với

$k = 1;$

$j = k;$

$i = j;$

Toán tử số học

Các toán tử cho kiểu dữ liệu số bao gồm các toán tử số học tiêu chuẩn: phép cộng (+), phép trừ (-), phép nhân (*), phép chia (/) và phần dư (%). Các toán hạng là các giá trị được vận hành bởi một toán tử.

| <i>Name</i> | <i>Meaning</i> | <i>Example</i> | <i>Result</i> |
|-------------|----------------|----------------|---------------|
| + | Addition | 34 + 1 | 35 |
| - | Subtraction | 34.0 – 0.1 | 33.9 |
| * | Multiplication | 300 * 30 | 9000 |
| / | Division | 1.0 / 2.0 | 0.5 |
| % | Remainder | 20 % 3 | 2 |

Hình: Toán tử số học

Khi cả hai toán hạng của một phép chia là số nguyên, kết quả của phép chia là thương và phần phân số bị cắt bớt. Ví dụ: $5/2$ mang lại 2, không phải 2,5 và $-5 / 2$ cho kết quả -2, không phải $-2,5$. Để thực hiện phép chia dấu phẩy động, một trong các toán hạng phải là số dấu phẩy động. Ví dụ: $5,0 / 2$ cho kết quả 2,5.

Toán tử %, được gọi là toán tử phần dư hoặc modulo, mang lại phần còn dư sau khi chia. Toán hạng bên trái là số bị chia và toán hạng bên phải là số bị chia. Do đó, $7 \% 3$ sinh ra 1 (dư 1), $3 \% 7$ sinh ra 3 (dư 3), $12 \% 4$ sinh ra 0 (dư 0), $26 \% 8$ sinh ra 2 và $20 \% 13$ sinh ra 7.

Toán tử % thường được sử dụng cho số nguyên dương, nhưng nó cũng có thể được sử dụng với số nguyên âm và giá trị dấu phẩy động. Phần số dư là âm chỉ khi số bị chia là âm. Ví dụ, $-7 \% 3$ cho kết quả -1, $-12 \% 4$ cho kết quả 0, $-26 \% -8$ cho kết quả -2 và $20 \% -13$ cho kết quả 7.

Phần số dư rất hữu ích trong lập trình. Ví dụ: số chẵn % 2 luôn là 0 và một số lẻ % 2 luôn là 1. Do đó, bạn có thể sử dụng thuộc tính này để xác định xem số chẵn hoặc lẻ.

Đánh giá biểu thức và mức độ ưu tiên của các toán tử

Biểu thức Java được đánh giá giống như biểu thức số học. Ví dụ, biểu thức số học:

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

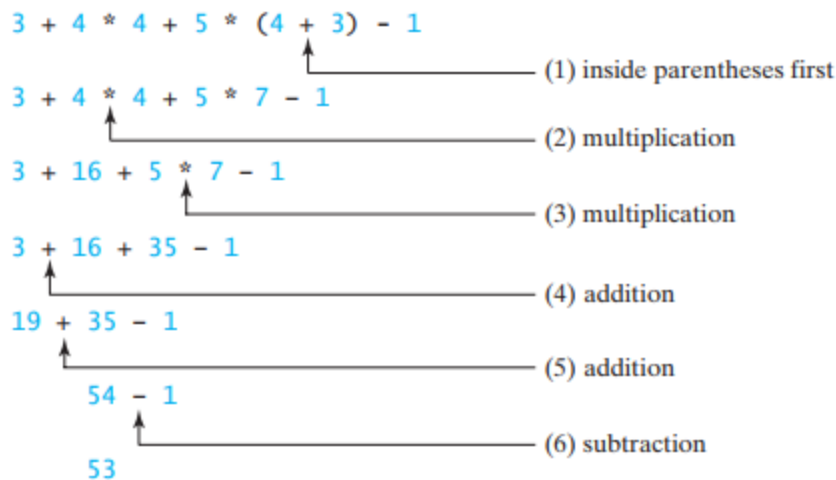
có thể được chuyển sang một biểu thức Java như sau:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

Mặc dù Java có cách riêng để đánh giá một biểu thức số học, kết quả của một biểu thức Java và biểu thức số học tương ứng của nó sẽ vẫn giống nhau. Do đó, bạn có thể áp dụng quy tắc số học để đánh giá một biểu thức Java. Các toán tử chứa trong các cặp dấu ngoặc đơn được đánh giá đầu tiên. Các dấu ngoặc đơn có thể được lồng vào nhau, trong trường hợp đó biểu thức trong ngoặc đơn bên trong được đánh giá đầu tiên. Khi nhiều toán tử được sử dụng trong một biểu thức, quy tắc ưu tiên toán tử sau được sử dụng để xác định thứ tự của đánh giá.

- Các toán tử nhân, chia và chia lấy phần dư được đánh giá đầu tiên. Nếu một biểu thức chứa một số toán tử nhân, chia và chia lấy phần dư, chúng được áp dụng từ trái sang phải.
- Các toán tử cộng và trừ được đánh giá sau cùng. Nếu một biểu thức chứa một số các toán tử cộng và trừ, chúng được đánh giá từ trái sang phải

Dưới đây là một ví dụ về cách một biểu thức được đánh giá:



[Bài đọc] Toán tử gán tăng cường và toán tử tăng giảm

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 54, 55)

Toán tử gán tăng cường

Các toán tử $+$, $-$, $*$, $/$, và $\%$ có thể được kết hợp với toán tử gán để tạo thành toán tử gán tăng cường. Thông thường, giá trị hiện tại của một biến được sử dụng, sửa đổi và sau đó được gán lại cho cùng một biến. Ví dụ: câu lệnh sau đây tăng số lượng biến lên 1:

```
count = count + 1;
```

Java cho phép bạn kết hợp các toán tử gán và bổ sung bằng cách sử dụng một hàm tăng cường (hoặc ghép) toán tử gán. Ví dụ: câu lệnh trước có thể được viết là

```
count += 1;
```

Dấu += được gọi là toán tử gán phép cộng. Bảng sau cho thấy các toán tử gán tăng cường.

| <i>Operator</i> | <i>Name</i> | <i>Example</i> | <i>Equivalent</i> |
|-----------------|---------------------------|----------------|-------------------|
| += | Addition assignment | i += 8 | i = i + 8 |
| -= | Subtraction assignment | i -= 8 | i = i - 8 |
| *= | Multiplication assignment | i *= 8 | i = i * 8 |
| /= | Division assignment | i /= 8 | i = i / 8 |
| %= | Remainder assignment | i %= 8 | i = i % 8 |

Hình: Toán tử gán tăng cường

Toán tử gán tăng cường được thực hiện cuối cùng sau tất cả các toán tử khác trong biểu thức được đánh giá. Ví dụ:

```
x /= 4 + 5,5 * 1,5;
```

tương đương:

```
x = x / (4 + 5.5 * 1.5);
```

Toán tử tăng giảm

Toán tử tăng (++) và toán tử giảm (--) dùng để tăng và giảm một biến số 1. ++ và -- là hai toán tử viết tắt để tăng và giảm một biến bằng 1. Những điều này rất hữu ích vì đó thường là giá trị cần thay đổi trong nhiều tác vụ lập trình. Ví dụ, mã sau đây tăng i lên 1 và giảm j đi 1.

```
int i = 3, j = 3;  
i++; // i becomes 4
```

```
j——; // j becomes 2
```

i++ được phát âm là i cộng cộng và i—— là i trừ trừ. Các toán tử này được gọi là tăng sau và giảm sau, bởi vì toán tử ++ và —— được đặt sau biến. Các toán tử này cũng có thể được đặt trước biến. Ví dụ,

```
int i = 3, j = 3;
```

```
++i; // i becomes 4
```

```
——j; // j becomes 2
```

++ i tăng i lên 1 và ——j giảm j bằng 1. Các toán tử này được gọi là tiền tố tăng trước và giảm trước. Như bạn thấy, hiệu ứng của i++ và ++ i hoặc i—— và ——i là giống nhau trong các ví dụ trước. Tuy nhiên, tác dụng của chúng khác nhau khi chúng được sử dụng trong các câu lệnh hỗn hợp chứ không phải chỉ có toán tử tăng và giảm. Bảng sau mô tả sự khác biệt của chúng và đưa ra các ví dụ:

| Operator | Name | Description | Example (assume i = 1) |
|--------------|---------------|-----------------------------------------------------------------------------------|------------------------------------------|
| ++var | preincrement | Increment var by 1, and use the new var value in the statement | int j = ++i; // j is 2, i is 2 |
| var++ | postincrement | Increment var by 1, but use the original var value in the statement | int j = i++; // j is 1, i is 2 |
| ——var | predecrement | Decrement var by 1, and use the new var value in the statement | int j = ——i; // j is 0, i is 0 |
| var—— | postdecrement | Decrement var by 1, and use the original var value in the statement | int j = i——; // j is 1, i is 0 |

Hình: Toán tử tăng giảm

Dưới đây là các ví dụ bổ sung để minh họa sự khác biệt giữa dạng tiền tố của ++ (hoặc ——) và dạng hậu tố của ++ (hoặc ——). Hãy xem xét đoạn mã sau:

```
int i = 10;  
int newNum = 10 * i++;  
System.out.print("i is " + i  
+ ", newNum is " + newNum);
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

```
i is 11, newNum is 100
```

Trong trường hợp này, i được tăng thêm 1, thì giá trị cũ của i được sử dụng trong phép nhân. Vì thế newNum trở thành 100. Nếu i++ được thay thế bằng ++i như sau:

```
int i = 10;  
int newNum = 10 * (++i);  
System.out.print("i is " + i  
+ ", newNum is " + newNum);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

```
i is 11, newNum is 110
```

i được tăng thêm 1 và giá trị mới của i được sử dụng trong phép nhân. Do đó newNum trở thành 110.

Đây là một ví dụ khác:

```
double x = 1.0;  
double y = 5.0;  
double z = x + (++y);
```

Sau khi tất cả ba dòng được thực thi, y trở thành 6.0, z trở thành 7.0 và x trở thành 0.0

[Bài đọc] Toán tử logic

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 93)

Các toán tử logic `!`, `&&`, `||`, `^` có thể được sử dụng để tạo một biểu thức Boolean hỗn hợp. Đôi khi, một câu lệnh có được thực thi hay không được xác định bởi sự kết hợp của một số điều kiện. Bạn có thể sử dụng các toán tử logic để kết hợp các điều kiện này để tạo thành một biểu thức Boolean hỗn hợp.

Toán tử logic, còn được gọi là toán tử Boolean, hoạt động trên các giá trị Boolean để tạo một giá trị Boolean mới. Bảng 1 liệt kê các toán tử Boolean. Bảng 2 xác định toán tử `not` (`!`), phủ định `true` thành `false` và `false` thành `true`. Bảng 3 xác định toán tử `and` (`&&`). `and` (`&&`) của hai toán hạng Boolean là `true` nếu và chỉ khi cả hai toán hạng đều `true`. Bảng 4 định nghĩa toán tử `or` (`||`). `or` (`||`) của hai toán hạng Boolean là `true` nếu ít nhất một trong các toán hạng là `true`. Bảng 5 xác định toán tử `exclusive or` (`^`). `exclusive or` (`^`) của hai toán hạng Boolean là `true` nếu và chỉ khi hai toán hạng có giá trị Boolean khác nhau. Lưu ý rằng `p1 ^ p2` giống với `p1 != p2`.

| <i>Operator</i> | <i>Name</i> | <i>Description</i> |
|-------------------------|---------------------------|---------------------|
| <code>!</code> | <code>not</code> | logical negation |
| <code>&&</code> | <code>and</code> | logical conjunction |
| <code> </code> | <code>or</code> | logical disjunction |
| <code>^</code> | <code>exclusive or</code> | logical exclusion |

Hình 1: Các toán tử Boolean

| p | !p | Example (assume age = 24, weight = 140) |
|-------|-------|-------------------------------------------------------------|
| true | false | !(age > 18) is false, because (age > 18) is true. |
| false | true | !(weight == 150) is true, because (weight == 150) is false. |

Hình 2: Toán tử chân lý !

| p ₁ | p ₂ | p ₁ && p ₂ | Example (assume age = 24, weight = 140) |
|----------------|----------------|----------------------------------|----------------------------------------------------------------------------------------------|
| false | false | false | |
| false | true | false | (age > 28) && (weight <= 140) is true, because (age > 28) is false. |
| true | false | false | |
| true | true | true | (age > 18) && (weight >= 140) is true, because (age > 18) and (weight >= 140) are both true. |

Hình 3: Toán tử chân lý &&

| p ₁ | p ₂ | p ₁ p ₂ | Example (assume age = 24, weight = 140) |
|----------------|----------------|----------------------------------|------------------------------------------------------------------------------------------------|
| false | false | false | (age > 34) (weight >= 150) is false, because (age > 34) and (weight >= 150) are both false. |
| false | true | true | |
| true | false | true | (age > 18) (weight < 140) is true, because (age > 18) is true. |
| true | true | true | |

Hình 4: Toán tử chân lý ||

| p ₁ | p ₂ | p ₁ ^ p ₂ | Example (assume age = 24, weight = 140) |
|----------------|----------------|---------------------------------|------------------------------------------------------------------------------------------------|
| false | false | false | (age > 34) ^ (weight > 140) is false, because (age > 34) and (weight > 140) are both false. |
| false | true | true | (age > 34) ^ (weight >= 140) is true, because (age > 34) is false but (weight >= 140) is true. |
| true | false | true | |
| true | true | false | |

Hình 5: Toán tử chân lý ^

[Bài đọc] Toán tử quan hệ và kiểu dữ liệu Boolean

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 76)

Kiểu dữ liệu boolean khai báo một biến có giá trị là true hoặc false.

Làm cách nào để bạn so sánh hai giá trị, chẳng hạn như bán kính lớn hơn 0, bằng 0 hay nhỏ hơn

hơn 0? Java cung cấp sáu toán tử quan hệ (còn được gọi là toán tử so sánh), được hiển thị trong Bảng 1, có thể được sử dụng để so sánh hai giá trị.

| Java Operator | Mathematics Symbol | Name | Example (radius is 5) | Result |
|---------------|--------------------|--------------------------|-----------------------|--------|
| < | < | less than | radius < 0 | false |
| <= | ≤ | less than or equal to | radius <= 0 | false |
| > | > | greater than | radius > 0 | true |
| >= | ≥ | greater than or equal to | radius >= 0 | true |
| == | = | equal to | radius == 0 | false |
| != | ≠ | not equal to | radius != 0 | true |

Bảng 1: Toán tử quan hệ

Kết quả của phép so sánh là một giá trị Boolean: true hoặc false. Ví dụ: câu lệnh sau hiển thị true:

```
double radius = 1;  
System.out.println(radius > 0);
```

Một biến chứa giá trị Boolean được gọi là biến Boolean. Kiểu dữ liệu Boolean được sử dụng để khai báo các biến Boolean. Một biến boolean có thể chứa một trong các giá trị true hoặc false.

[Bài đọc] Câu lệnh if

(Nguồn: Introduction to Java Programming, Comprehensive Version (10th Edition) - Y. Daniel Liang – Tr 78)

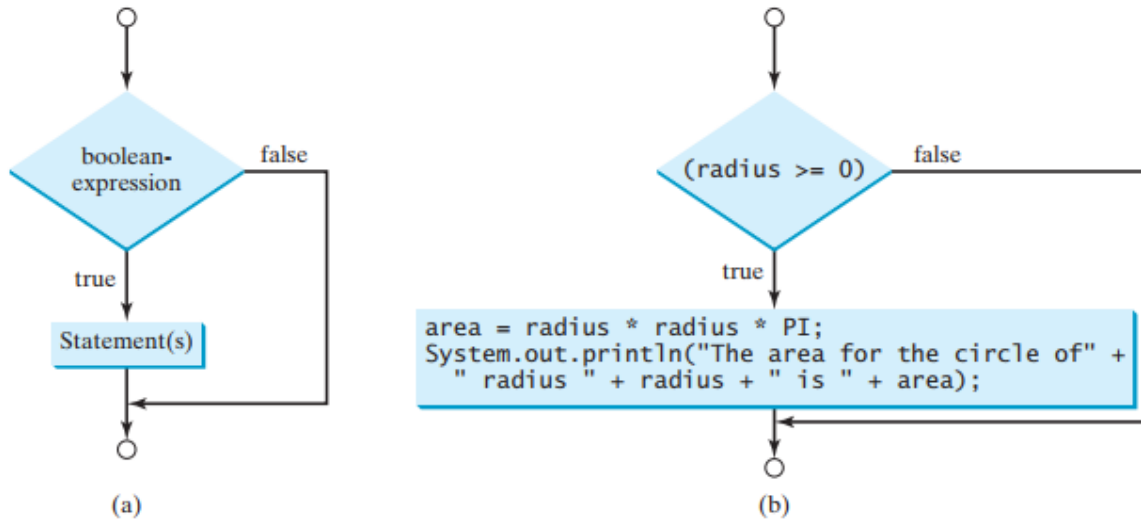
Câu lệnh if là một cấu trúc cho phép chương trình xác định các luồng thực thi thay thế.

Java có một số loại câu lệnh if: câu lệnh if một chiều, câu lệnh if-else hai chiều, câu lệnh if lồng nhau, câu lệnh if-else đa chiều, câu lệnh switch và biểu thức điều kiện.

Câu lệnh if một chiều thực hiện một hành động nếu và chỉ khi điều kiện là đúng. Cú pháp cho câu lệnh if một chiều là:

```
if (boolean-expression) {  
    statement(s);  
}
```

Lưu đồ trong Hình 1(a) minh họa cách Java thực thi cú pháp của câu lệnh if. Lưu đồ là một sơ đồ mô tả thuật toán hoặc quy trình, hiển thị các bước dưới dạng các loại hình hộp khác nhau và thứ tự của chúng bằng cách kết nối chúng với các mũi tên. Quy trình hoạt động là được thể hiện trong các hộp này và các mũi tên nối chúng biểu thị luồng điều khiển. Hộp hình kim cương biểu thị điều kiện Boolean và hộp hình chữ nhật biểu thị các câu lệnh.



Hình 1: Một câu lệnh if thực thi các câu lệnh nếu biểu thức boolean đánh giá là true.

Nếu biểu thức boolean đánh giá là true, các câu lệnh trong khối sẽ được thực thi. Để làm ví dụ, hãy xem đoạn mã sau:

```

if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
        radius + " is " + area);
}
  
```

Lưu đồ của câu lệnh trên được thể hiện trong Hình 1(b). Nếu giá trị của radius lớn hơn hoặc bằng 0, thì area được tính và kết quả được hiển thị; nếu không thì, hai câu lệnh trong khối sẽ không được thực thi.

Biểu thức boolean được đặt trong dấu ngoặc đơn. Ví dụ, mã trong (a) là sai. Nó phải được sửa lại, như thể hiện trong (b).

```
if i > 0 {  
    System.out.println("i is positive");  
}
```

(a) Wrong

```
if (i > 0) {  
    System.out.println("i is positive");  
}
```

(b) Correct

Các dấu ngoặc nhọn có thể được bỏ qua nếu chúng bao gồm một câu lệnh duy nhất. Ví dụ, các câu lệnh sau là tương đương:

```
if (i > 0) {  
    System.out.println("i is positive");  
}
```

(a)

Equivalent

```
if (i > 0)  
    System.out.println("i is positive");
```

(b)

Ghi chú:

Bỏ qua dấu ngoặc nhọn làm cho mã ngắn hơn, nhưng nó dễ bị lỗi.

[Bài đọc] Câu lệnh if-else

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 80)

Câu lệnh if-else quyết định luồng thực thi dựa trên điều kiện đúng hay sai.

Câu lệnh if một chiều thực hiện một hành động nếu điều kiện được chỉ định là đúng. Nếu điều kiện là sai, không có gì được thực hiện. Nhưng nếu bạn muốn thực hiện các hành động thay thế khi tình trạng là sai? Bạn có thể sử dụng câu lệnh if-else hai chiều. Các hành động mà một câu lệnh if-else hai chiều chỉ định khác nhau dựa trên việc điều kiện là đúng hay sai.

Đây là cú pháp cho câu lệnh if-else hai chiều:

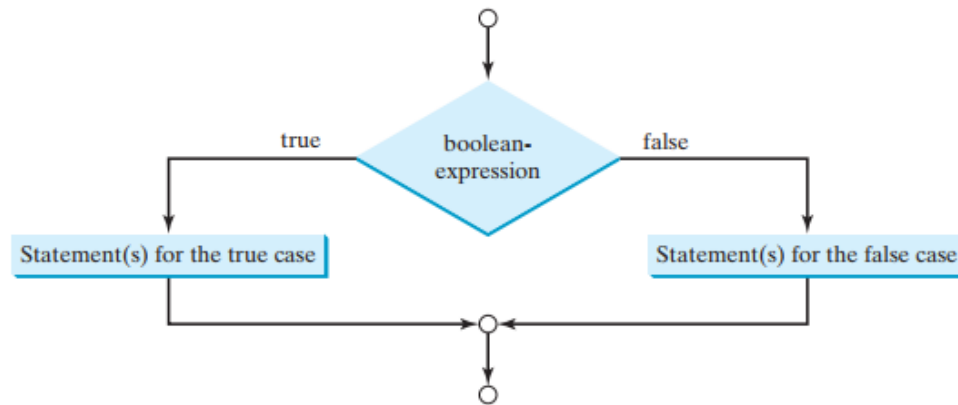
```
if (boolean-expression) {
```

```

statement(s)-for-the-true-case;
}
else {
statement(s)-for-the-false-case;
}

```

Lưu đồ của câu lệnh được thể hiện trong Hình 1:



Hình 1: Một câu lệnh if-else thực hiện các câu lệnh cho trường hợp true nếu biểu thức đánh giá boolean-expression là true; nếu không, các câu lệnh cho trường hợp sai được thực thi

Nếu biểu thức đánh giá boolean là true, (các) câu lệnh cho trường hợp true là được thực thi; nếu không, (các) câu lệnh cho trường hợp false được thực thi. Ví dụ, hãy xem xét mã sau:

```

if (radius >= 0) {
area = radius * radius * PI;
System.out.println("The area for the circle of radius " +
radius + " is " + area);
}

```

```
}  
else {  
System.out.println("Negative input");  
}
```

Nếu $\text{radius} \geq 0$ là đúng, area được tính toán và hiển thị; nếu nó là sai, thông báo "Negative input" được hiển thị.

Đây là một ví dụ khác về việc sử dụng câu lệnh if-else. Ví dụ kiểm tra xem một số chẵn hoặc lẻ, như sau:

```
if (number % 2 == 0)  
System.out.println(number + " is even.");  
else  
System.out.println(number + " is odd.");
```

[Bài đọc] Câu lệnh if lồng và câu lệnh if-else đa chiều

(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 81)

Một câu lệnh if có thể nằm bên trong một câu lệnh if khác để tạo thành một câu lệnh if lồng nhau.

Câu lệnh trong câu lệnh if hoặc if-else có thể là bất kỳ câu lệnh Java hợp pháp nào, bao gồm một câu lệnh if hoặc if-else khác. Câu lệnh if bên trong được cho là được lồng bên trong câu lệnh if bên ngoài. Câu lệnh if bên trong có thể chứa một câu lệnh if khác; trong thực tế, không có giới hạn độ sâu của câu lệnh if lồng nhau. Ví dụ, sau đây là một câu lệnh if lồng nhau:

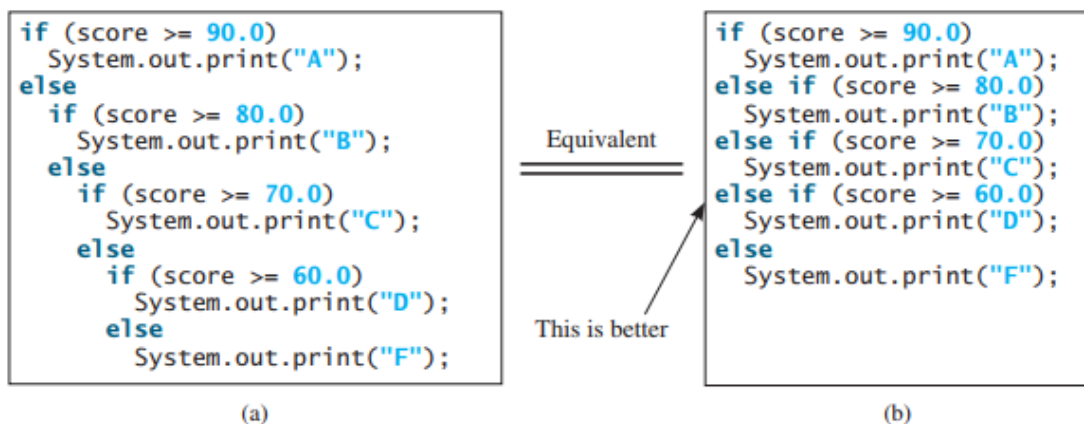
```
if (i > k) {  
if (j > k)  
System.out.println("i and j are greater than k");  
}
```

else

```
System.out.println("i is less than or equal to k");
```

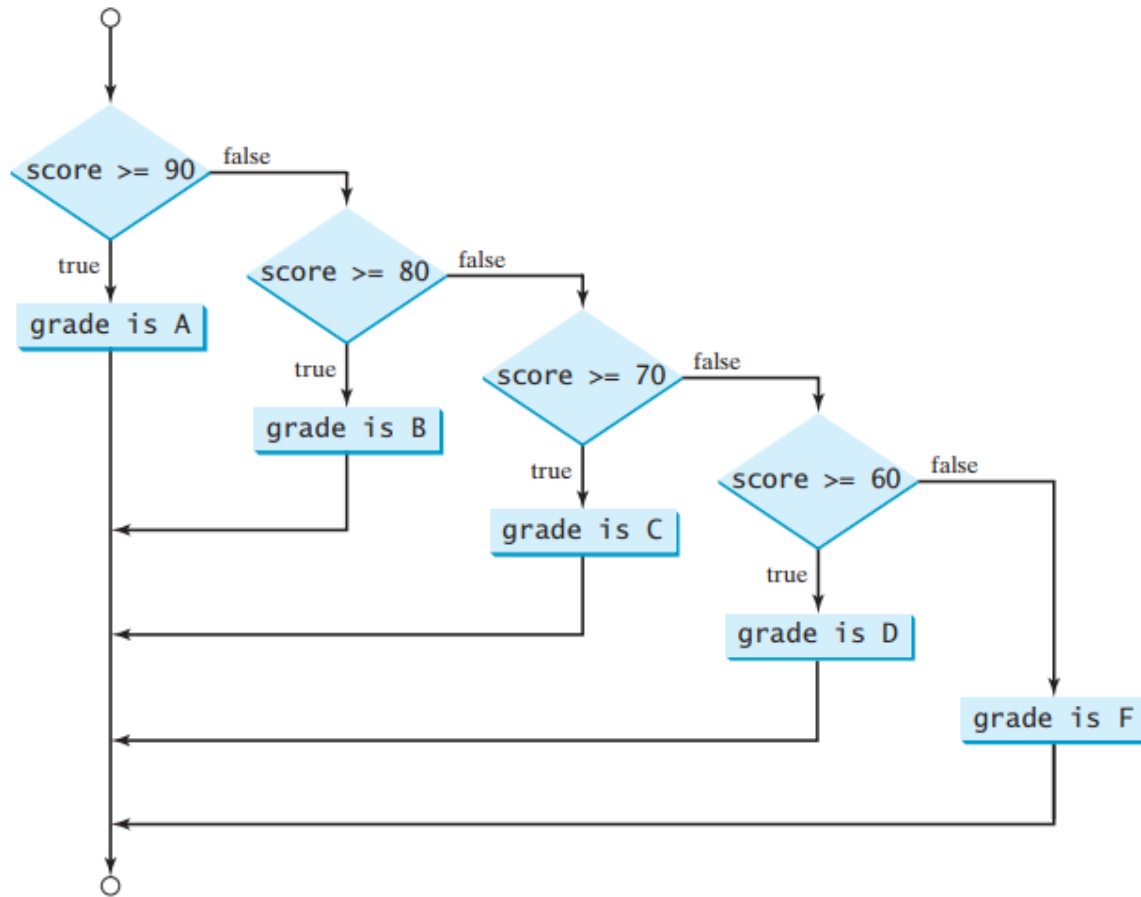
Câu lệnh if ($j > k$) được lồng bên trong câu lệnh if ($i > k$).

Câu lệnh if lồng nhau có thể được sử dụng để triển khai nhiều lựa chọn thay thế. Ví dụ, cho trong Hình 1, in một loại chữ cái theo điểm số, với nhiều lựa chọn thay thế.



Hình 1: Định dạng ưa thích cho nhiều lựa chọn thay thế được hiển thị trong (b) bằng cách sử dụng câu lệnh if-else đa chiều.

Việc thực hiện câu lệnh if này được tiến hành như trong Hình 2. Điều kiện đầu tiên ($\text{score} \geq 90.0$) được kiểm tra. Nếu đúng, grade là A. Nếu sai, điều kiện thứ hai ($\text{score} \geq 80.0$) được kiểm tra. Nếu điều kiện thứ hai là đúng thì grade là B. Nếu điều kiện đó là sai, điều kiện thứ ba và các điều kiện còn lại (nếu cần) được kiểm tra cho đến khi điều kiện được đáp ứng hoặc tất cả các điều kiện được chứng minh là sai. Nếu tất cả các điều kiện đều sai, grade là F. Lưu ý rằng một điều kiện chỉ được kiểm tra khi tất cả các điều kiện trước đó nó là sai.



Hình 2: Bạn có thể sử dụng câu lệnh if-else đa chiều để gán grade.

Câu lệnh if trong Hình 1(a) tương đương với câu lệnh if trong Hình 1(b). Trên thực tế, Hình 1(b) là kiểu trình bày ưa thích cho nhiều câu lệnh if thay thế. Phong cách này, được gọi là câu lệnh if-else đa chiều, tránh phức tạp và giúp chương trình dễ dàng đọc.

[Bài đọc] Câu lệnh switch-case

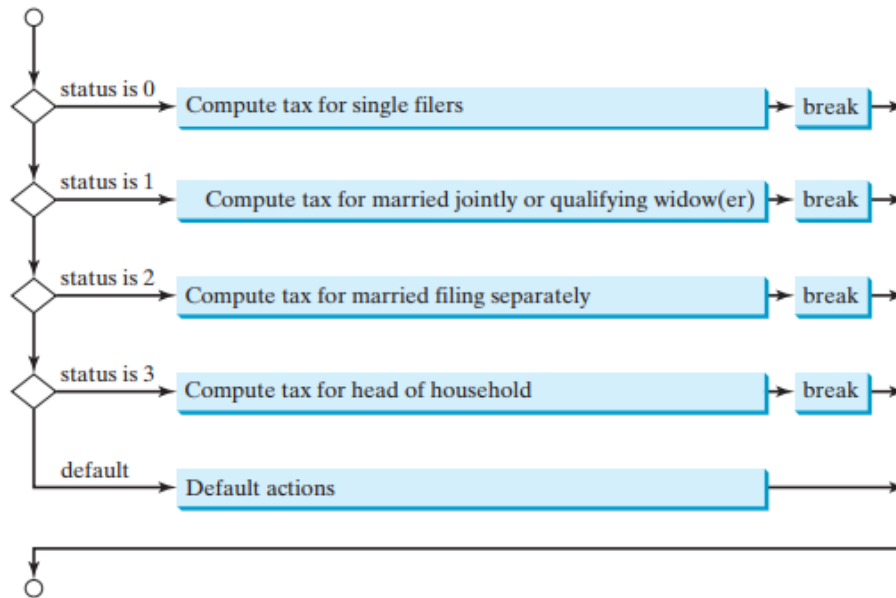
(Nguồn: *Introduction to Java Programming, Comprehensive Version (10th Edition)* - Y. Daniel Liang – Tr 100)

Câu lệnh switch thực thi các câu lệnh dựa trên giá trị của một biến hoặc một biểu thức

Ví dụ sau đây, có bốn trường hợp tính thuế, phụ thuộc vào giá trị của trạng thái (status). Để giải thích đầy đủ cho tất cả các trường hợp, các câu lệnh if lồng nhau có thể được sử dụng. Tuy nhiên, lạm dụng quá mức việc lồng ghép các câu lệnh if làm cho một chương trình khó đọc. Java cung cấp một câu lệnh switch để đơn giản hóa việc viết mã cho trường hợp có nhiều điều kiện. Bạn có thể viết câu lệnh switch sau để thay thế câu lệnh if lồng nhau:

```
switch (status) {  
    case 0: compute tax for single filers;  
        break;  
    case 1: compute tax for married jointly or qualifying widow(er);  
        break;  
    case 2: compute tax for married filing separately;  
        break;  
    case 3: compute tax for head of household;  
        break;  
    default: System.out.println("Error: invalid status");  
        System.exit(1);  
}
```

Lưu đồ của câu lệnh switch được thể hiện trong hình sau:



Hình 1: Câu lệnh switch kiểm tra tất cả các trường hợp và thực hiện các câu lệnh trong trường hợp phù hợp.

Câu lệnh này kiểm tra xem status có khớp với giá trị 0, 1, 2 hoặc 3, theo thứ tự. Nếu khớp, thuế (tax) tương ứng sẽ được tính; nếu không khớp, một thông báo sẽ được hiển thị.

Đây là cú pháp đầy đủ cho câu lệnh switch:

```
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;  
}
```

Câu lệnh switch tuân theo các quy tắc sau:

- Biểu thức switch (switch-expression) chỉ chứa giá trị char, byte, short, int hoặc String và phải luôn được đặt trong dấu ngoặc đơn.
- Value1, . . . , và valueN phải có cùng kiểu dữ liệu với giá trị của switch-expression. Lưu ý rằng value1, . . . , và valueN là các biểu thức không đổi, có nghĩa là chúng không thể chứa các biến, chẳng hạn như $1 + x$.
- Khi giá trị trong câu lệnh case khớp với giá trị của switch-expression, các câu lệnh bắt đầu từ case này được thực thi cho đến khi gặp câu lệnh break hoặc kết thúc của câu lệnh switch.
- Trường hợp default, là tùy chọn, có thể được sử dụng để thực hiện các hành động khi không có các case cụ thể khớp với switch-expression.
- Break là từ khóa là tùy chọn. Câu lệnh break ngay lập tức kết thúc câu lệnh switch.

Chú ý:

Đừng quên sử dụng câu lệnh break khi cần thiết. Sau khi một case được khớp, các câu lệnh bắt đầu từ case đã so khớp được thực thi cho đến khi một câu lệnh break hoặc kết thúc của câu lệnh switch.

Ví dụ: mã sau đây hiển thị các ngày trong tuần cho ngày từ 1 đến 5, và các ngày cuối tuần cho ngày 0 và ngày 6:

```
switch (day) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:
```

```
case 5: System.out.println("Weekday"); break;
case 0:
case 6: System.out.println("Weekend");
}
```

[Bài đọc] Lệnh switch-case

Khi chúng ta muốn viết một cấu trúc rẽ nhánh có nhiều lựa chọn, ta có thể sử dụng nhiều lệnh if-else lồng nhau. Tuy nhiên, trong trường hợp việc lựa chọn rẽ nhánh phụ thuộc vào giá trị (kiểu số nguyên hoặc kí tự, hoặc xâu kí tự kể từ JDK 7.0) của một biến hay biểu thức, ta có thể sử dụng cấu trúc switch để chương trình dễ hiểu hơn. Lệnh switch điển hình có dạng như sau:

```
switch (biểu_thức) {
    case hằng_1:
        tập_lệnh_1;
        break;
    case hằng_2:
        tập_lệnh_2;
        break;
    ...
    default:
        tập_lệnh_mặc_định;
}
```

Khi lệnh switch được chạy, biểu_thức được tính giá trị và so sánh với hằng_1. Nếu bằng nhau, chuỗi lệnh kể từ tập_lệnh_1 được thực thi cho đến khi gặp lệnh break đầu tiên, đến đây chương trình sẽ nhảy tới điểm kết thúc cấu trúc switch. Nếu biểu_thức không có giá trị bằng hằng_1, nó sẽ được so sánh với hằng_2, nếu bằng nhau, chương trình sẽ thực thi chuỗi lệnh kể từ tập_lệnh_2 tới khi gặp lệnh break đầu tiên thì nhảy tới cuối cấu trúc switch. Quy trình cứ tiếp diễn như vậy. Cuối cùng, nếu biểu_thức có giá trị khác với tất cả các giá trị đã được liệt kê (hằng_1, hằng_2, ...), chương trình sẽ thực thi tập_lệnh_mặc_định nằm sau nhãn default: nếu như có nhãn này (không bắt buộc).

Ví dụ, lệnh sau so sánh giá trị của biến grade với các hằng kí tự 'A', 'B', 'C' và in ra các thông báo khác nhau cho từng trường hợp.

```
switch (grade) {  
    case 'A':  
        System.out.print("Grade = A"); break;  
    case 'B':  
        System.out.print("Grade = B"); break;  
    case 'C':  
        System.out.print("Grade = C"); break;  
    default:  
        System.out.print("Grade's not A, B or C");  
}
```

Nó tương đương với khối lệnh if-else lồng nhau sau:

```
if(grade == 'A')  
    System.out.println("Grade = A");  
else if(grade == 'B')  
    System.out.println("Grade = B");  
else if(grade == 'C')  
    System.out.println("Grade = C");  
else  
    System.out.println("Grade's not A, B or C");
```

Lưu ý, các nhãn case trong cấu trúc switch phải là hằng chứ không thể là biến hay biểu thức. Nếu cần so sánh với biến hay biểu thức, ta nên dùng khối lệnh if-else lồng nhau.

Lệnh break

Vấn đề đặc biệt của cấu trúc switch là các lệnh break. Nếu ta không tự gắn một lệnh break vào cuối chuỗi lệnh cần thực hiện cho mỗi trường hợp, chương trình sẽ chạy tiếp chuỗi lệnh của trường hợp sau chứ không tự động nhảy tới cuối cấu trúc switch. Ví dụ, đoạn chương trình sau sẽ chạy lệnh in thứ nhất nếu grade nhận một trong ba giá trị 'A', 'B', 'C' và chạy lệnh in thứ hai trong trường hợp còn lại:

```

switch (grade) {
    case 'A':
    case 'B':
    case 'C':
        System.out.println("Grade is A, B or C.");
        break;
    default:
        System.out.println("Grade is not A, B or C.");
}

```

Chương trình sau là một ví dụ hoàn chỉnh sử dụng cấu trúc switch để in ra các thông báo khác nhau tùy theo xếp loại học lực (grade) mà người dùng nhập từ bàn phím. Trong đó, case 'A' kết thúc với break sau chỉ một lệnh, còn case 'B' chạy tiếp qua case 'C', 'D' rồi mới gặp break và thoát khỏi lệnh switch. Nhãn default được dùng để xử lý trường hợp biến grade giữ giá trị không hợp lệ đối với xếp loại học lực. Trong nhiều chương trình, phần default thường được dùng để xử lý các trường hợp không mong đợi, chẳng hạn như để bắt lỗi các kí hiệu học lực không hợp lệ mà người dùng có thể nhập sai.

```

Scanner input = new Scanner(System.in);
System.out.print("Enter your grade: ");
String userInput = input.next();
char grade = userInput.charAt(0);
switch (grade) {
    case 'A':
        System.out.println("Excellent!");
        break;
    case 'B':
        System.out.println("Great!");
    case 'C':
    case 'D':
        System.out.println("Well done!");
        break;
    case 'F':
        System.out.println("Sorry, you failed.");
}

```

```
        break;
    default:
        System.out.println("Error! Invalid grade.");
}
```

switch-case với kiểu dữ liệu String

Kể từ Java SE 7, ta có thể dùng các đối tượng String làm nhãn cho các lệnh case.

Ví dụ:

```
switch (answer) {
    case "yes":
        System.out.print("You said 'yes'"); break;
    case "no":
        System.out.print("You said 'no'"); break;
    default:
        System.out.print("I don't get what you mean.");
}
```

[Bài đọc] Biểu thức điều kiện

(Nguồn: Introduction to Java Programming, Comprehensive Version (10th Edition) - Y. Daniel Liang – Tr 103)

Một biểu thức điều kiện đánh giá một biểu thức dựa trên một điều kiện.

Bạn có thể muốn gán một giá trị cho một biến bị hạn chế bởi các điều kiện nhất định. Ví dụ, câu lệnh sau đây gán 1 cho y nếu x lớn hơn 0 và -1 cho y nếu x nhỏ hơn hoặc bằng 0.

```
if (x > 0)
    y = 1;
```

else

```
y = -1;
```

Ngoài ra, như trong ví dụ sau, bạn có thể sử dụng biểu thức điều kiện để đạt được cùng một kết quả.

```
y = (x > 0) ? 1 : -1;
```

Biểu thức điều kiện ở một kiểu hoàn toàn khác, không có if trong câu lệnh. Cú pháp là:

```
boolean-expression ? expression1 : expression2;
```

Kết quả của biểu thức điều kiện này là biểu thức 1 (expression1) nếu biểu thức boolean (boolean-expression) là true; nếu không thì kết quả là biểu thức 2 (expression2).

Giả sử bạn muốn gán biến lớn hơn trong 2 biến num1 và num2 cho max. Bạn có thể

chỉ cần viết một câu lệnh sử dụng biểu thức điều kiện:

```
max = (num1 > num2) ? num1 : num2;
```

Ví dụ khác, câu lệnh sau sẽ hiển thị thông báo “num is even” nếu num là chẵn, và nếu không thì hiển thị “num is odd”.

```
System.out.println((num % 2 == 0) ? "num is even" : "num is odd");
```

Như bạn có thể thấy từ các ví dụ này, biểu thức điều kiện cho phép bạn viết mã ngắn gọn.

Ghi chú:

Các ký hiệu ? và : cùng xuất hiện trong một biểu thức điều kiện. Chúng tạo thành một toán tử điều kiện và còn được gọi là toán tử bậc ba vì nó sử dụng ba toán hạng. Nó là toán tử bậc ba duy nhất trong Java.

[Bài đọc] Mức độ ưu tiên và sự kết hợp của toán tử

(Nguồn: Introduction to Java Programming, Comprehensive Version (10th Edition) - Y. Daniel Liang – Tr 104)

Mức độ ưu tiên của toán tử và sự kết hợp các toán tử xác định thứ tự các toán tử được đánh giá. Giả sử rằng bạn có biểu thức này:

```
3 + 4 * 4 > 5 * (4 + 3) - 1 && (4 - 3 > 5)
```

Giá trị của biểu thức là gì? Thứ tự thực hiện của các toán tử là gì?

Biểu thức trong dấu ngoặc đơn được đánh giá đầu tiên. (Các dấu ngoặc đơn có thể được lồng vào nhau, trong đó trường hợp biểu thức trong dấu ngoặc đơn bên trong được thực hiện trước.) Khi đánh giá một biểu thức không có dấu ngoặc đơn, các toán tử được áp dụng theo quy tắc ưu tiên và quy tắc kết hợp.

Quy tắc ưu tiên xác định mức độ ưu tiên cho các toán tử, như được hiển thị trong Bảng 1, chứa các toán tử mà bạn đã học cho đến nay. Các toán tử được liệt kê theo thứ tự ưu tiên giảm dần từ trên xuống dưới. Các toán tử logic có mức độ ưu tiên thấp hơn so với các toán tử quan hệ và toán tử quan hệ có mức độ ưu tiên thấp hơn toán tử số học.

Các toán tử có cùng thứ tự ưu tiên xuất hiện trong cùng một nhóm.

| <i>Precedence</i> | <i>Operator</i> |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------|
| | <code>var++</code> and <code>var--</code> (Postfix) |
| | <code>+</code> , <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix) |
| | <code>(type)</code> (Casting) |
| | <code>!</code> (Not) |
| | <code>*</code> , <code>/</code> , <code>%</code> (Multiplication, division, and remainder) |
| | <code>+</code> , <code>-</code> (Binary addition and subtraction) |
| | <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> (Relational) |
| | <code>==</code> , <code>!=</code> (Equality) |
| | <code>^</code> (Exclusive OR) |
| | <code>&&</code> (AND) |
| | <code> </code> (OR) |
| | <code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> (Assignment operator) |

Hình 1: Biểu đồ ưu tiên của toán tử

Nếu các toán tử có cùng mức độ ưu tiên ở cạnh nhau, quy tắc kết hợp của chúng sẽ xác định thứ tự đánh giá. Tất cả các toán tử nhị phân ngoại trừ các toán tử gán đều được kết hợp trái.

Ví dụ: vì `+` và `-` có cùng mức độ ưu tiên và được kết hợp trái:

$$a - b + c - d \quad \underline{\underline{\text{is equivalent to}}} \quad ((a - b) + c) - d$$

Các toán tử gán là kết hợp phải:

`a = b += c = 5` is equivalent to `a = (b += (c = 5))`

Giả sử a, b, c có giá trị là 1 trước khi gán; sau khi toàn bộ biểu thức được đánh giá, giá trị biểu thức là 6, b có giá trị là 6, và c có giá trị là 5.