1. Khái quát chung.
2. **Khái niệm.**  
   - Java là một ngôn ngữ lập lập trình, được phát triển bởi Sun Microsystem vào năm 1995, là ngôn ngữ kế thừa trực tiếp từ C/C++ và là một ngôn ngữ lập trình hướng đối tượng. Ngôn ngữ này được thiết kế để có thể hoạt động trên nhiều nền tảng mà không cần thay đổi mã nguồn, nhờ vào đặc tính "Viết một lần, chạy mọi nơi" (Write Once, Run Anywhere - WORA).

- Java được sử dụng rộng rãi trong phát triển ứng dụng doanh nghiệp, ứng dụng web, và cả ứng dụng di động, đặc biệt là trên nền tảng Android.

1. **Lí do ra đời.**

Ngôn ngữ Java ra đời với mục tiêu giải quyết một số vấn đề tồn tại trong các ngôn ngữ lập trình phổ biến trước đó. Dưới đây là những lí do chính dẫn đến sự ra đời của Java:

**- Khả năng di động cao (Platform Independence):**

Một trong những vấn đề lớn nhất của các ngôn ngữ lập trình trước đây là tính phụ thuộc vào nền tảng (platform dependency). Ví dụ, một chương trình viết trên hệ điều hành Windows có thể không chạy được trên hệ điều hành Unix hay Mac mà không cần sửa đổi. Java ra đời để giải quyết vấn đề này bằng cách áp dụng nguyên lý "Viết một lần, chạy mọi nơi" (Write Once, Run Anywhere - WORA). Nhờ vào việc sử dụng Java Virtual Machine (JVM), mã bytecode của Java có thể chạy trên bất kỳ hệ điều hành nào có cài đặt JVM, giúp các chương trình Java trở nên đa nền tảng.

**- Tăng tính bảo mật (Security):**

Trong thời kỳ các mạng máy tính và Internet phát triển, nhu cầu về bảo mật trong các ngôn ngữ lập trình trở nên cấp thiết. Java được thiết kế với nhiều cơ chế bảo mật tích hợp, bao gồm các biện pháp kiểm soát truy cập và sandbox để hạn chế các ứng dụng nguy hiểm hoặc không đáng tin cậy. Đây là một trong những lý do khiến Java trở thành lựa chọn lý tưởng cho các ứng dụng web và doanh nghiệp.

**- Tính mạnh mẽ và đáng tin cậy (Robustness):**

Java ra đời với mong muốn tạo ra một ngôn ngữ lập trình có tính mạnh mẽ. Nó loại bỏ các lỗi phổ biến mà lập trình viên thường gặp trong các ngôn ngữ khác, chẳng hạn như tràn bộ nhớ (memory overflow) hoặc lỗi con trỏ (pointer errors). Java có cơ chế quản lý bộ nhớ tự động thông qua hệ thống "Garbage Collection" giúp loại bỏ các đối tượng không còn được sử dụng, qua đó giảm nguy cơ rò rỉ bộ nhớ.

**- Tính đơn giản trong lập trình (Simplicity):**

Java được thiết kế để đơn giản và dễ sử dụng. So với các ngôn ngữ lập trình như C và C++, Java loại bỏ những yếu tố phức tạp như quản lý con trỏ và hỗ trợ các cơ chế xử lý lỗi tốt hơn, giúp lập trình viên dễ dàng làm quen và phát triển ứng dụng nhanh chóng và hiệu quả.

1. **Cách Java hoạt động, điều gì xảy ra khi chạy code Java (.java)**

- Khi một chương trình Java được biên dịch, mã nguồn (.java) được chuyển thành mã bytecode (.class). Mã bytecode này không phụ thuộc vào nền tảng và có thể chạy trên bất kỳ hệ thống nào có cài đặt Java Virtual Machine (JVM). JVM sẽ thực thi mã bytecode bằng cách dịch nó thành mã máy cụ thể của nền tảng hiện tại. Quá trình này giúp Java có khả năng di động giữa các hệ thống khác nhau mà không cần phải biên dịch lại.

- Quá trình biên dịch và thực thi Java:

+ Biên dịch: Mã nguồn (.java) -> Mã bytecode (.class) bằng trình biên dịch javac.

+ Thực thi: JVM đọc mã bytecode và dịch sang mã máy để thực thi.

1. Cấu trúc của một chương trình Java.

Một chương trình Java cơ bản thường bao gồm:

- Khai báo package (nếu có)

- Khai báo class: Java là ngôn ngữ hướng đối tượng, do đó mọi chương trình đều phải nằm trong một class.

- Phương thức main: Đây là phương thức chính mà JVM sẽ gọi khi chương trình bắt đầu.

Note:

Class là đơn vị cơ bản trong Java.

Phương thức là các hành vi hoặc thao tác mà một đối tượng có thể thực hiện.

Ví dụ về chương trình Java:

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

1. Package.

- Package (gói) trong java là một nhóm các class, interface và các package khác. Trong java chúng ta sử dụng package để tổ chức cấu trúc dự án hợp lý.

- Java có 2 loại package chính:

+ Các package tích hợp sẵn

+ Các package do chúng ta tự định nghĩa.

- Tác dụng của package:  
+ Package giúp chúng ta tái sử dụng

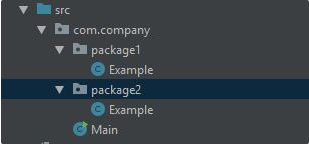
Trong một dự án, sẽ có lúc một logic được sử dụng ở nhiều nơi, nếu mỗi chỗ như vậy bạn đều code lại logic thì việc bị lặp code sẽ ngày càng nhiều Đến khi logic đó thay đổi thì chúng ta lại phải lục lọi tất cả mọi nơi sử dụng logic đó mà sửa lại. Điều này vừa gây rủi ro và vừa khổ cực. VÌ thế, trường hợp này chúng ta nên tạo ra package chứa class implement logic đó và import vào những nơi cùng sử dụng. Đến lúc sửa chúng ta chỉ cần tìm đến package chứa class implement nó và sửa lại thôi.

+ Sử dụng Package để có cấu trúc tốt

Trong một dự án lớn, nó có thể chứa đến hàm trăm, hàng ngàn class, interface để thực thi các nhóm chức năng khác nhau. Sử dụng package để phân tầng theo nhóm chức năng và chứa các class liên quan giúp chúng ta quản lý code hiệu quả. Tên package phải nói lên được ý nghĩa của package đó.

+ Package giúp tránh xung đột tên

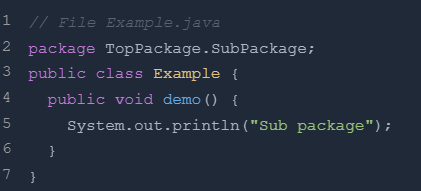
Đôi lúc chúng ta đặt tên của 2 hoặc nhiều class cùng tên. Dùng package chúng ta có thể tránh được xung đột bằng cách chỉ định đúng đường dẫn từ package cha đến các class cùng tên.



- Sub package:

+ Trong một package có thể chứa các package khác, các package này được gọi là sub package.

Ở ví dụ dưới đây chúng ta có TopPackage chứa SubPackage, và class Example thuộc về SubPackage.

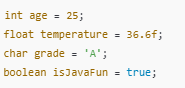


1. Syntax cơ bản trong Java.
2. Khai báo biến nguyên thủy.

- Java hỗ trợ các kiểu dữ liệu nguyên thủy như int, float, double, char, boolean, byte, short, long.

- Cách khai báo: <Kiểu dữ liệu> <Tên biến>;

Ví dụ:



1. Vòng lặp.

- Java có 3 loại vòng lặp:

+ Vòng lặp for: thường được sử dụng trong trường hợp biết trước số lần lặp.

+ Vòng lặp while: thường được sử dụng trong trường hợp số lần lặp không cố định.

+ Vòng lặp do…while: thường được sử dụng trong trường hợp số lần lặp không cố định.

a, Cấu trúc vòng lặp for.

- Đây là cấu trúc lặp phổ biến nhất trong các ngôn ngữ lập trình, mà nội dung cuả vòng lặp cần phải lặp đi lặp lại một số lần biết trước, cú pháp của nó như sau:

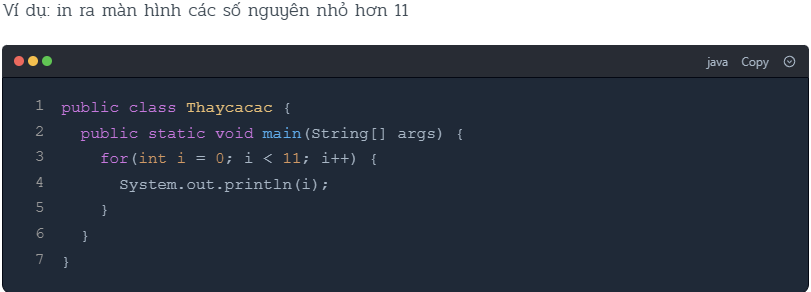
Cú pháp:

for (<Khởi tạo biến chạy>;<Biểu thức điều kiện>;<Thay đổi biến chạy>){

<Khối lệnh lặp lại>

}

Ví dụ:

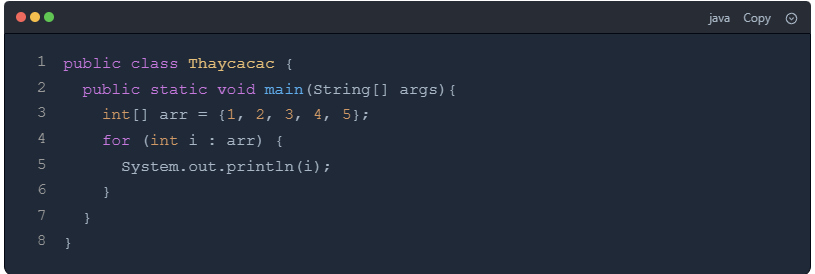


- Vòng lặp cải tiến: Vòng lặp for cải tiến được sử dụng để lặp mảng (array) hoặc collection trong java. Bạn có thể sử dụng nó dễ dàng, dễ hơn cả vòng lặp for đơn giản. Bởi vì bạn không cần phải tăng hay giảm giá trị của biến rồi kiểm tra điều kiện, bạn chỉ cần sử dụng ký hiệu hai chấm.  
for (<Kiểu tập hợp> var : <Tập hợp>) {

<Khối lệnh lặp lại>

}

Ví dụ:



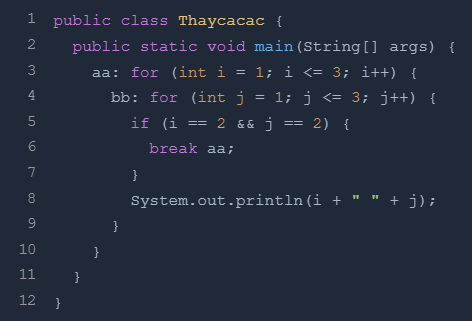
- Vòng lặp for gán nhãn: Chúng ta có để đặt tên cho mỗi vòng lặp for bằng cách gán nhãn trước vòng lặp for. Điều này rất hữu dụng khi chúng ta muốn thoát/tiếp tục(break/continues) chạy vòng lặp for.

Cú pháp:

<Tên nhãn>: for (<Khởi tạo biến chạy>; <Biểu thức điều kiện>; <Thay đổi biến chạy>) {

<Khối lệnh lặp lại>

}



b, Vòng lặp while.

- Vòng lặp while được sử dụng để thực thi nhiều lần một đoạn chương trình, khi một điều kiện vẫn còn đúng. Vòng lặp while thường được sử dụng khi số lần lặp không được xác định trước (Không cố định).

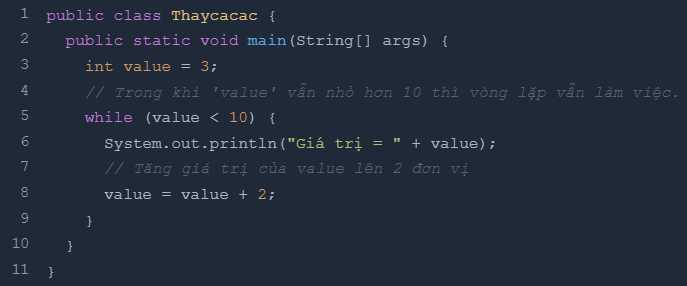
Cú pháp:

while (<Điều kiện lặp>) {

<Khối lệnh lặp lại>

}

Ví dụ:



c, Vòng lặp do-while.

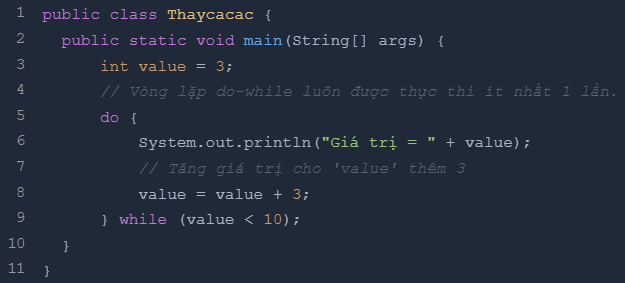
Cú pháp:

do {

<Khối lệnh lặp lại>

} while(<Điều kiện lặp>)

Ví dụ:



1. Cấu trúc rẽ nhánh.

a, if - else.

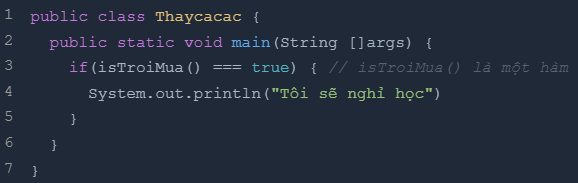
- Dạng khuyết:

if (<Điều kiện>){

<Khối lệnh>

}

Ví dụ:



- Dạng đủ:

if (<Điều kiện>){

<Khối lệnh>

}

else if{

<Khối lệnh>

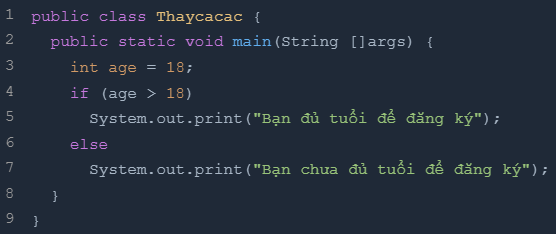
}

else{

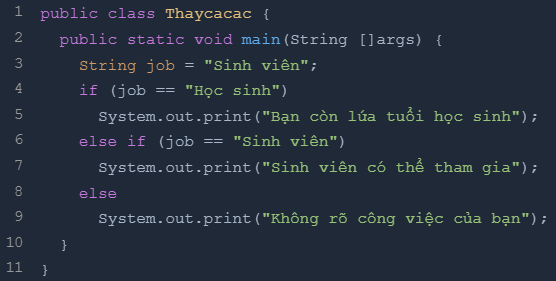
<Khối lệnh>

}

Ví dụ 1:



Ví dụ 2:



1. Mảng.

a, Mảng một chiều.

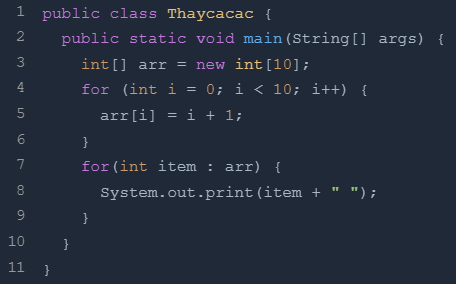
- Cách khai báo:

+ Cách 1: Khởi tạo không gán giá trị ban đầu

<Kiểu dữ liệu>[] <tên mảng> = new <kiểu dữ liệu>[số lượng phần tử];

+ Cách 2: Khởi tạo có gán giá trị ban đầu

<Kiểu dữ liệu>[] <tên mảng> = {value 0, value 1, ... value n};



- Truy xuất phần tử trong mảng:

<mảng>[vị trí phần tử trong mảng];

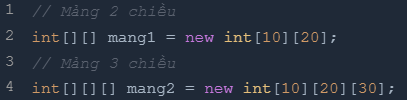
b, Mảng đa chiều.

- Cách khai báo:

+ Cách 1: Khai báo không gán giá trị ban đầu

<Kiểu dữ liệu>[][][]..[] <tên mảng> = new <kiểu dữ liệu>[<kích thước thứ 1>][<kích thước thứ 2>]….[<kích thước thứ n>];

Ví dụ:



+ Cách 2: Khai báo có gán giá trị ban đầu

<Kiểu dữ liệu>[][] <tên mảng> = {

{giaTriR1C1, giaTriR1C2, ....},

{giaTriR2C1, giaTriR2C2, ....}

};

- Truy xuất phần tử trong mảng:

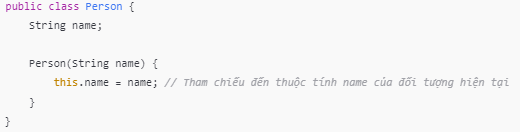
<Tên mảng>[<Vị trí hàng>][<Vị trí cột>]

1. Tổng quan về Class và Object.

a, this

***this*** được sử dụng để tham chiếu đến đối tượng hiện tại trong class.

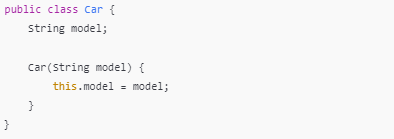
Ví dụ:



b, constructor

***constructor*** là một phương thức đặc biệt được gọi khi một đối tượng được khởi tạo. Constructor thường được sử dụng để gán giá trị ban đầu cho các thuộc tính của đối tượng.

Ví dụ:



c, access modifier

Java có các từ khóa kiểm soát truy cập: private, public, protected, và default, giúp kiểm soát việc truy cập vào các thuộc tính và phương thức.

+ public: Có thể truy cập từ bất kỳ đâu.

+ private: Chỉ có thể truy cập trong cùng một class.

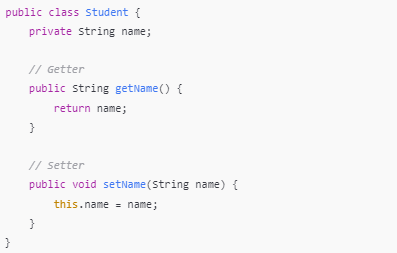
+ protected: Truy cập trong cùng package hoặc thông qua kế thừa.

+ default: Chỉ truy cập trong cùng package.

d, getter và setter

***getter và setter*** là các phương thức để truy cập và thay đổi giá trị của các thuộc tính private.

Ví dụ:



e, static

***static*** được sử dụng để khai báo các thuộc tính hoặc phương thức không phụ thuộc vào đối tượng cụ thể mà thuộc về class.

Ví dụ:

