**Buổi 1: Welcome to Java.**

I.Tìm hiểu chung về JAVA.

1.Định nghĩa :

**- Java** là một **một ngôn ngữ lập trình hiện đại, bậc cao, hướng đối tượng, bảo mật và mạnh mẽ.** và là một **Platform**.

**- Platform:** Bất cứ môi trường phần cứng hoặc phần mềm nào mà trong đó có một chương trình chạy, thì được hiểu như là một Platform. Với môi trường runtime riêng cho mình (JRE) và API, Java được gọi là Platform.

- Phương châm của java là **"Write Once, Run Anywhere"** - viết một lần chạy nhiều nơi, nghĩa là bạn chỉ cần viết một lần trên window chẳng hạn, sau đó vẫn chương trình đó bạn có thể chạy trên Linux, Android, các thiết bị J2ME...

- Java có thể tạo nhiều loại ứng dụng khác nhau:

* Standalone application: ứng dụng độc lập
* Web application: chạy trên server, tạo web động
* Enterprise application: ứng dụng dành cho doanh nghiệp, bank,... bảo mật, chịu tải cao
* Mobile application

Có khoảng 3 tỉ thiết bị chạy Java platform, chứng tỏ Java rất phổ biến hiện nay

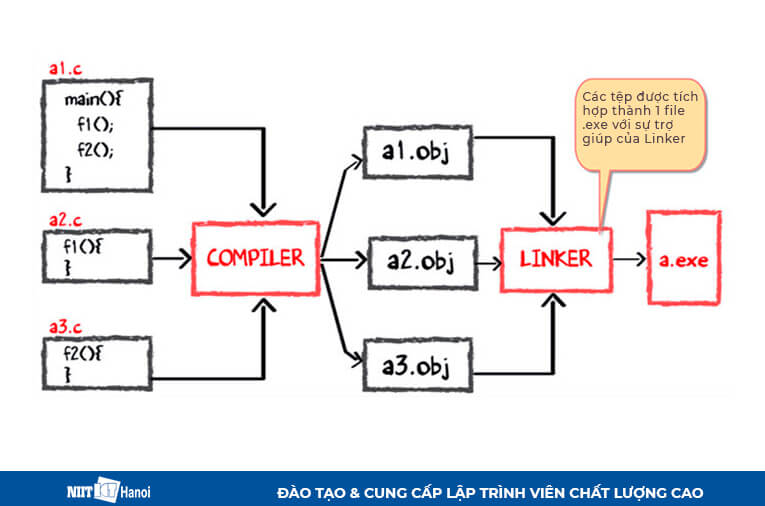
2. Lịch sử Java

**- Java** được khởi xướng bởi **James Gosling** và các bạn đồng nghiệp ở Sun Microsystems năm 1991. Từ ý tưởng muốn lập trình để điều khiển mà không phụ thuộc vào loại CPU cho các thiết bị điện tử như tivi, máy giặt, lò nướng... Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, gọn, hiệu quả, độc lập thiết bị và ngôn ngữ **“Oak”** ra đời, sau đó được đổi tên thành Java.

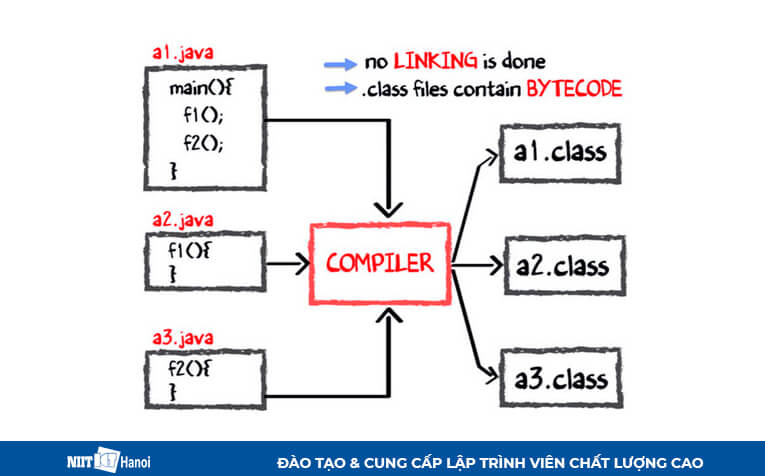
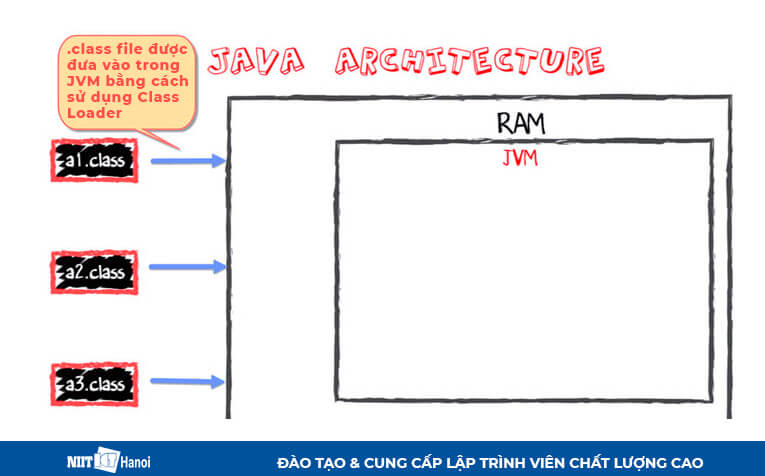
- Java là tên một hòn đảo của Indonesia nơi đầu tiên sản xuất ra cà phê. Đó là lý do tại sao biểu tượng của java là cốc coffee bốc khói nghi ngút thơm lừng.

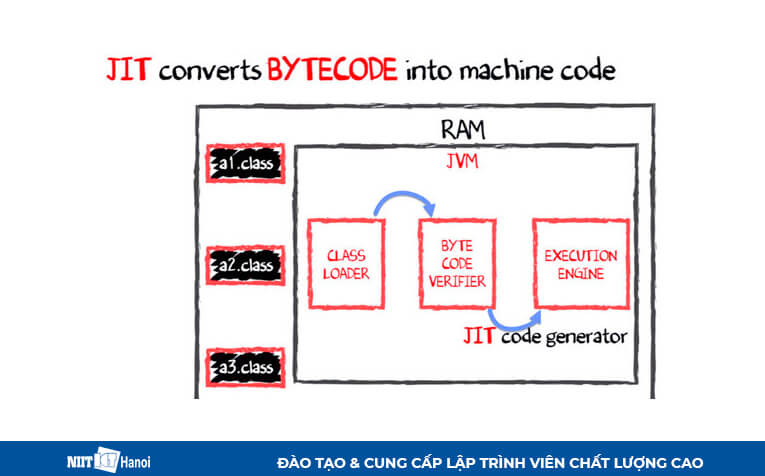
3. Cách hoạt động khi chạy 1 code JAVA.

a. Cách hoạt động của code C:



b. Cách hoạt động của Code Java:

 => 

=> 

- Java VM hoặc Máy ảo Java nằm trên RAM. Trong quá trình thực thi, sử dụng class loader, class files được đưa vào RAM. Tại đây BYTE code được xác minh cho tính bảo mật.

- Tiếp theo, Execution Engine sẽ chuyển đổi bytecode thành mã máy gốc. Đây chỉ là trong thời gian biên dịch. Đây là một trong những lý do chính tại sao Java tương đối chậm.

c. Tại sao Java vừa thông dịch vừa biên dịch?

- Trình biên dịch (compiler) là một chương trình chuyển đổi chương trình từ cấp độ ngôn ngữ này sang cấp độ khác. Ví dụ chuyển đổi chương trình C thành mã máy.  
- Trình Biên dịch Java chuyển đổi mã Java cấp cao thành bytecode (cũng là một loại mã máy).  
- Trình Thông dịch (Interpreter) là một chương trình chuyển đổi một chương trình ở cấp độ này sang ngôn ngữ lập trình khác ở cùng cấp độ. Ví dụ chuyển đổi chương trình Java thành C.  
- Trong Java, trình tạo mã Just-In-Time chuyển đổi bytecode thành mã máy gốc có cùng cấp độ.  
Do đó, Java vừa được biên dịch cũng như thông dịch.

d. Tại sao Java lại chậm?  
-Hai lý do chính đằng sau khiến Java chậm là:

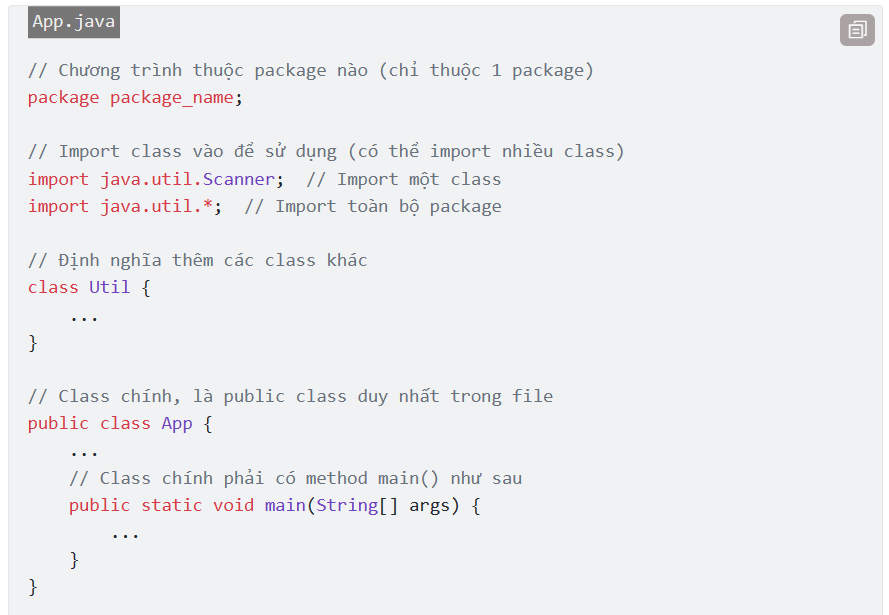
* Dynamic Linking: Không giống như C, trong Java liên kết được thực hiện lúc mỗi khi chương trình chạy
* Run-time Interpreter: Việc chuyển đổi bytecode thành mã máy gốc được thực hiện vào thời điểm mã chạy. Chứ không được chuyển đổi trước rồi mới chạy. Điều này là lý do làm cho Java chậm.

-Tuy nhiên, phiên bản mới nhất của Java đã giải quyết các vấn đề tắc nghẽn về hiệu năng ở mức độ lớn.

4. Cấu trúc 1 chương trình JAVA.

a. Cấu trúc 1 chương trình đơn giản:

**-** Cấu trúc chung của một chương trình java. Khi đi sâu hơn, thì mỗi class chứa trong một file riêng cũng có cấu trúc như thế này.



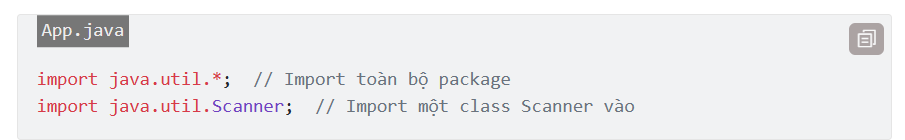
Source code lưu trữ trong file App.java, mỗi file như vậy đều phải có một public class duy nhất, các class khác là phụ và chỉ được khai báo class.

Tên public class cần trùng với tên file, như ví dụ trên file là App.java thì tên class chính phải là App. Trong class chính, phải có một method main() là public static void. Đây là nơi bắt đầu chạy toàn bộ chương trình.

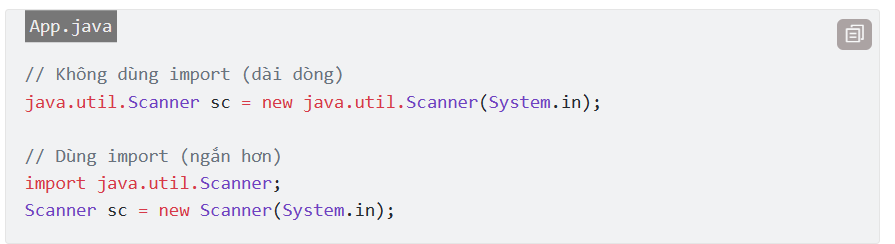
b. Comment

c. Import:

- Dùng từ khóa import để import một hoặc nhiều class vào chương trình. Các class có thể thuộc nhiều package khác nhau.



- Nếu không import như trên, thì khi dùng trong chương trình cần chỉ định rõ package.Class, ví dụ như sau.



- Câu lệnh import thường đặt ở ngoài và trên cùng, dưới khai báo package.

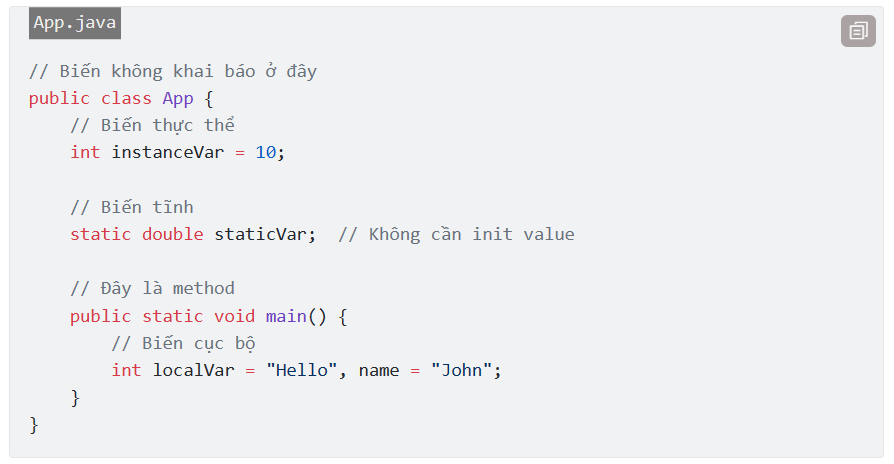
d. Biến và Hằng.

**Variable**

Biến trong Java có 3 loại, mỗi loại nằm ở vị trí riêng:

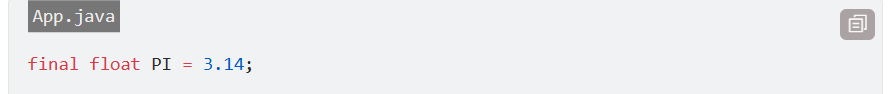
* Biến thực thể (instance variable): nằm trong class, không thuộc method nào. Biến này thuộc về các thực thể (instance - object) được tạo ra từ class.
* Biến tĩnh (static variable): tương tự biến thực thể, nhưng có từ khóa static để đánh dấu biến thuộc về class, không phải object.
* Biến cục bộ (local variable): được khai báo trong method.

Cú pháp khai báo tương tự C++. Có thể bỏ qua init value, và có thể khai báo nhiều biến cùng loại trên một dòng.



**Constant**

Hằng (constant) tương tự như khai báo biến. Trước đây dùng từ khóa const, nhưng được thay thế bằng final.

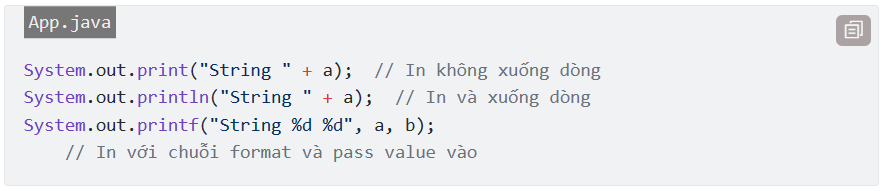


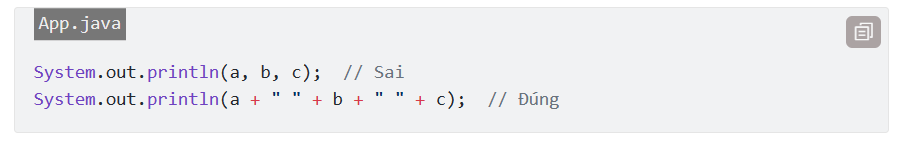
Hằng không thể gán giá trị mới, và bắt buộc có init value khi khai báo. Đối với hằng số là object, thì các trường của object vẫn có thể thay đổi được.

e. Input và Output

- Output

Dùng các lệnh print của System.out để xuất output ra màn hình console



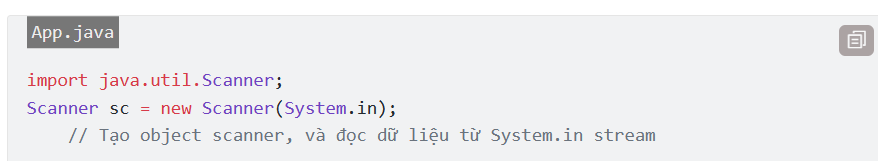


Phải dùng phép nối chuỗi +, java sẽ tự động convert các biến thành chuỗi và nối lại mới hợp lệ.

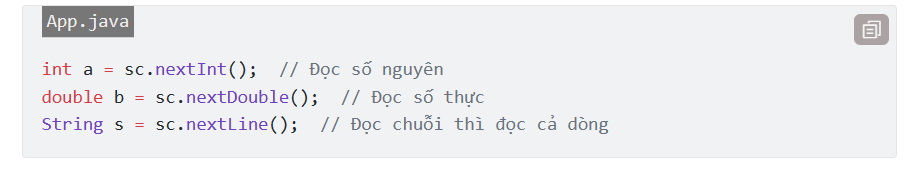
Ngoài output stream là System.out còn có System.err tương tự, nhưng chỉ dùng xuất lỗi ra màn hình. Console trong eclipse sẽ hiển thị các văn bản xuất ra System.err bằng màu đỏ.

- Input :

Dùng class java.util.Scanner như sau:

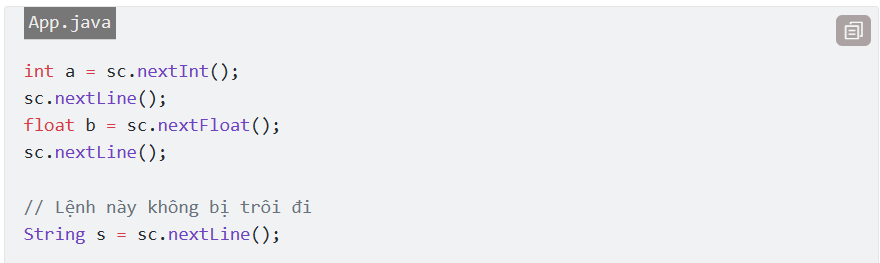


Để đọc dữ liệu, dùng các method next\_() phù hợp với loại dữ liệu cần đọc, ví dụ.



Sau khi dùng các lệnh nextInt(), nextFloat(),... nói chung là đọc số, thì dùng thêm một lệnh nextLine() phía sau để xóa kí tự \n thừa còn lại trong bộ đệm sau khi đọc số.

Các method đọc số chỉ lấy số, bỏ lại kí tự \n, làm cho lệnh đọc chuỗi phía sau bị trôi đi (kết quả chuỗi rỗng, nó chạy luôn mà không dừng lại để nhập). Do đó, dùng thêm nextLine() sau khi đọc số để xóa kí tự \n còn sót đi, để tránh lỗi trôi lệnh như trên.



Cuối cùng là hai lưu ý khi sử dụng Scanner class để tránh lỗi:

* Nên dùng một scanner duy nhất cho toàn bộ chương trình. Nhiều scanner cùng truy cập tới System.in stream thì sẽ gây xung đột (ngay cả khi một scanner đã đóng).
* Khi dùng xong scanner cần gọi method close() để đóng input stream lại.

## 5. Syntax là gì?

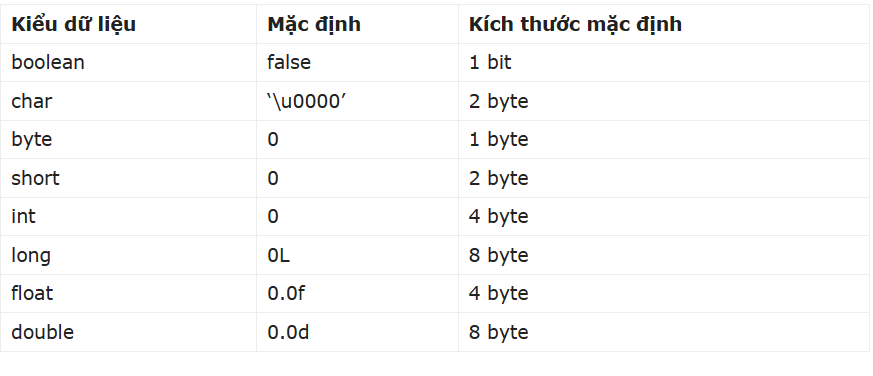
Syntax hay đầy đủ hơn là programming syntax – cú pháp câu lệnh, là thuật ngữ dùng để diễn tả các quy tắc, luật lệ về cách thức trình bày và logic của một câu lệnh nào đó. Syntax của mỗi ngôn ngữ lập trình sẽ có sự khác nhau về mặt cú pháp.

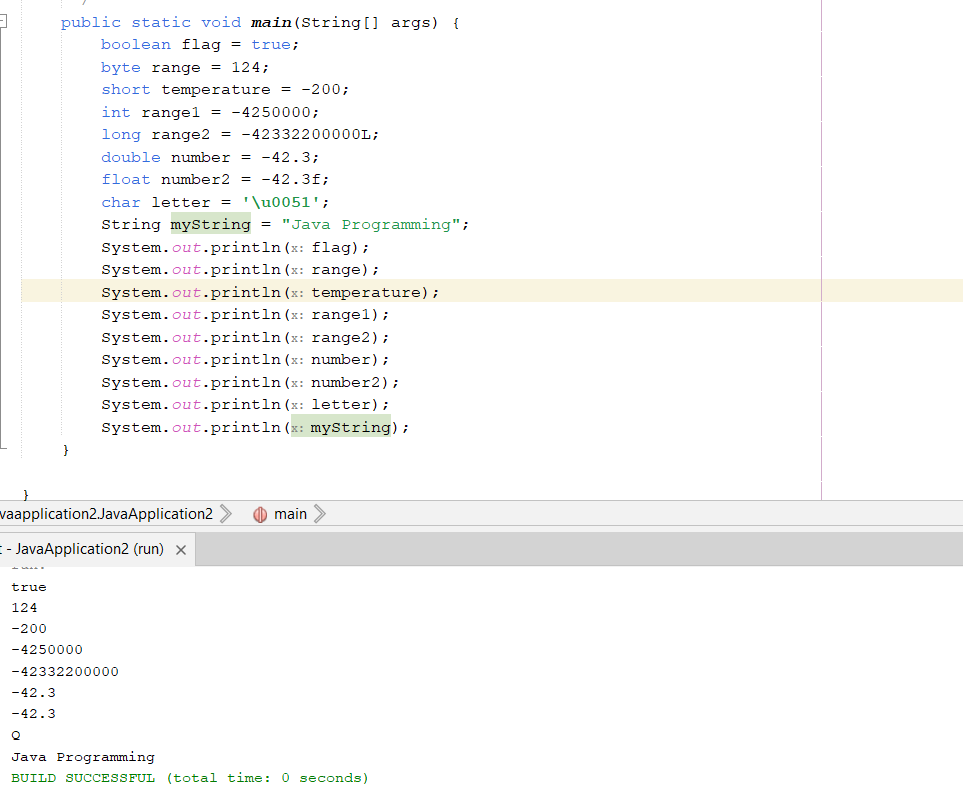
Một câu lệnh thông thường sẽ có cấu trúc về cú pháp như sau:

* Mở đầu bằng ký tự hay một từ riêng biệt để định danh câu lệnh
* Xác định trật tự của các toán tử, hệ thống các ký tự được dùng để diễn tả câu lệnh hoặc các tham số
* Tất cả mọi ngôn ngữ lập trình đều bắt buộc có ký tự kết thúc câu lệnh, và ký tự được dùng nhiều nhất thường là dấu chấm phẩy (“;”)
* Số lượng cú pháp của mỗi câu lệnh phụ thuộc vào yêu cầu của từng ngôn ngữ lập trình
* Tùy vào cú pháp được thiết kế mà mỗi câu lệnh sẽ có một giá trị riêng biệt. Những giá trị này sẽ được dùng cho một hoặc nhiều mục đích khác nhau. Nhưng, mục đích chung giữa các loại câu lệnh chính là tính khả thi khi đưa vào sử dụng thực tiễn của chúng.

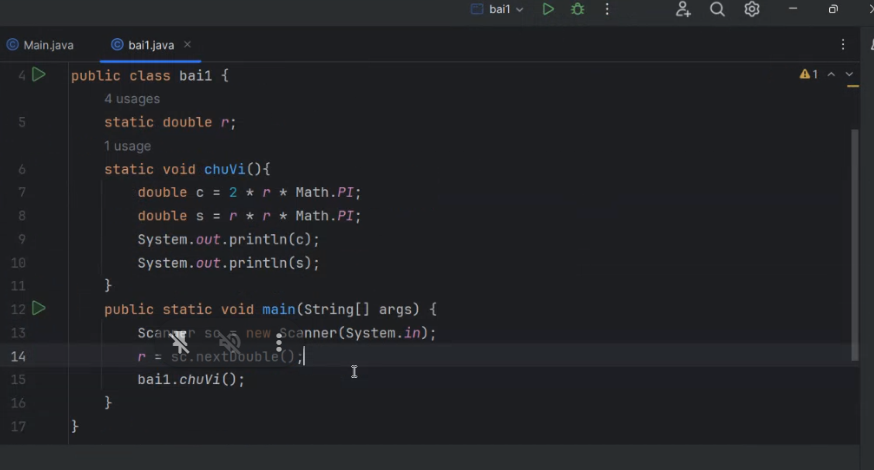
6. Khai báo biến nguyên thủy:

- Các kiểu dữ liệu nguyên thủy



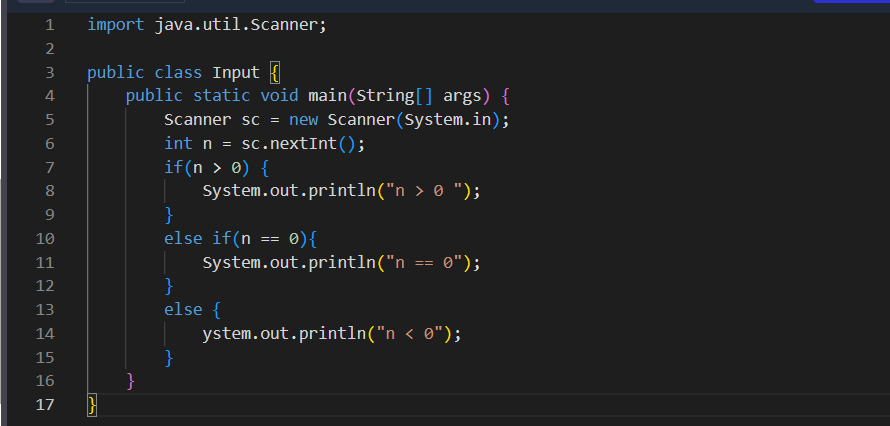


* + Khai báo biến toàn cục:



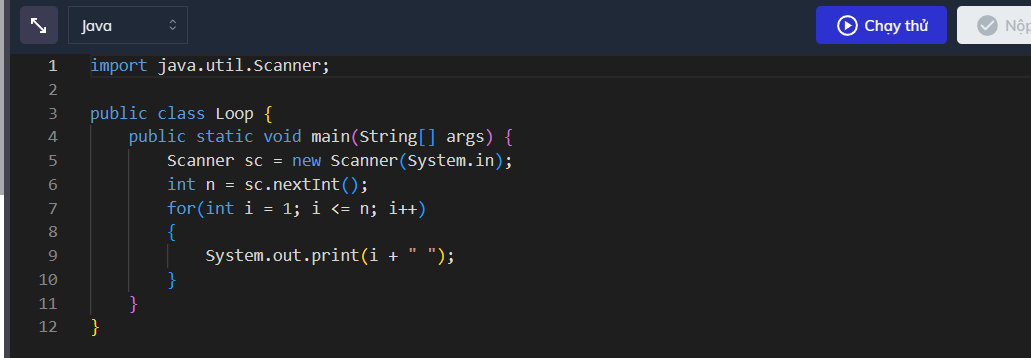
* + Lỗi xuống dòng :

7. Câu lệnh rẽ nhánh

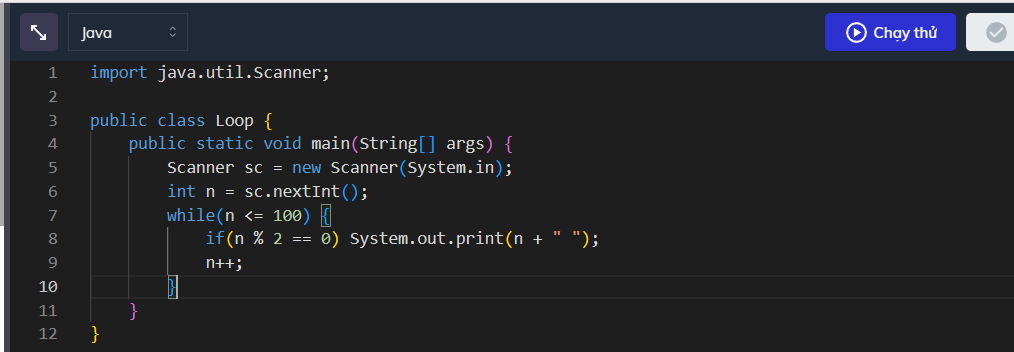


8. Vòng lặp

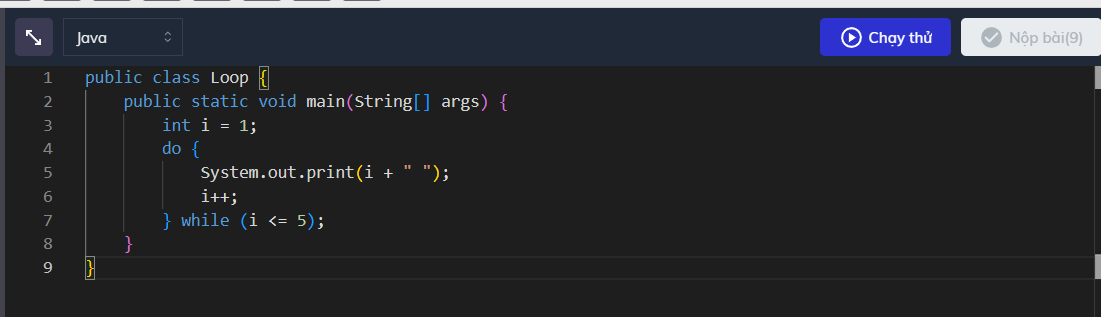
a. Vòng lặp for:



b. Vòng lặp while:



c. Vòng lặp do-while:



9.static

-Là 1 hàm thuộc class

10.