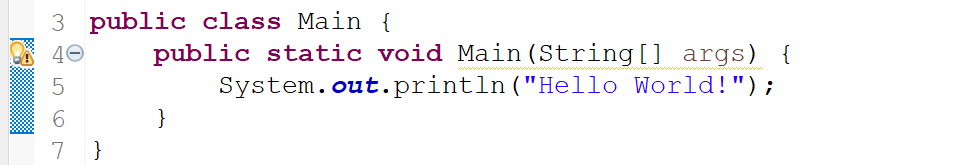
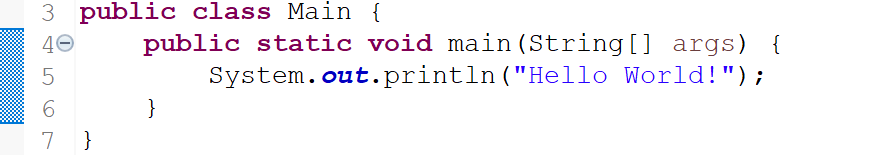
Java

Friday, January 12, 2024

11:43 AM

**Constructor name >< method name**





**Output syntax**

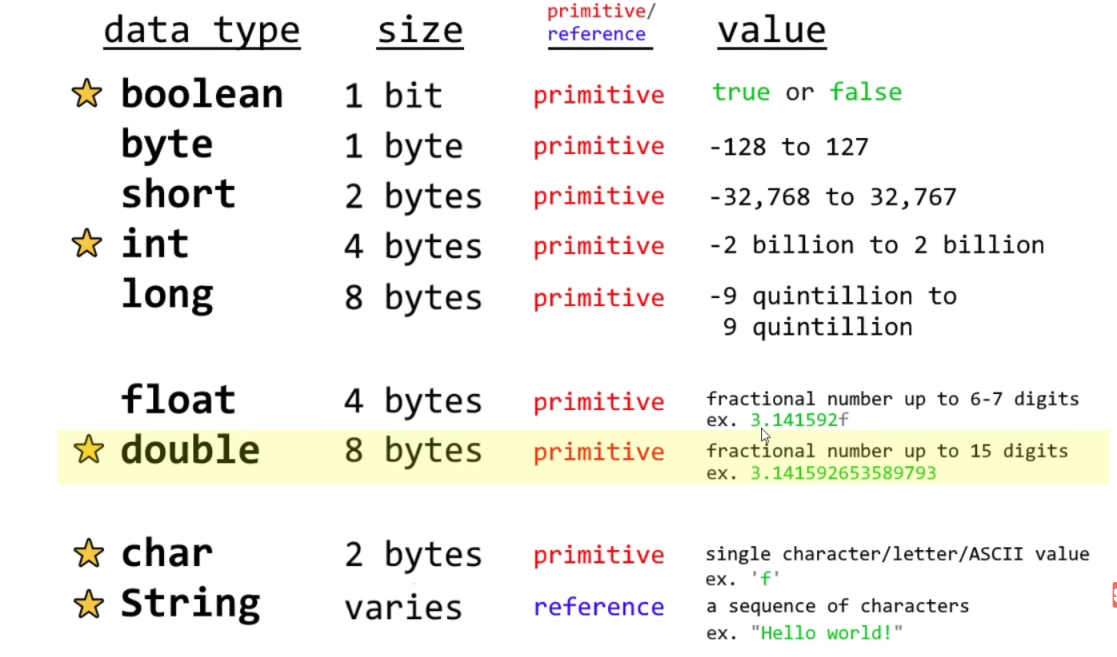
System.out.print();

Text -> inside double quotes

Number -> no need doueble quotes

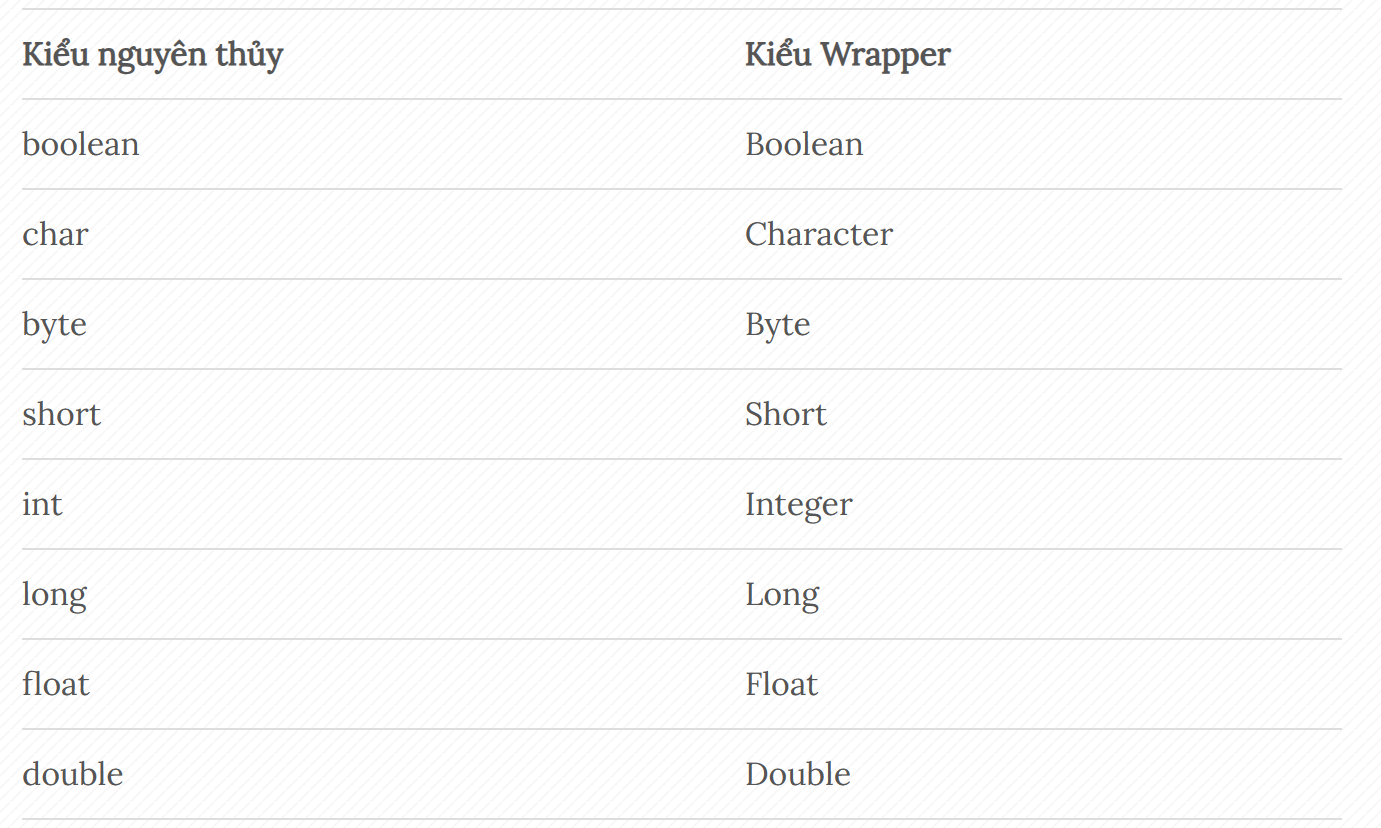
System.out.println(); -> with end line

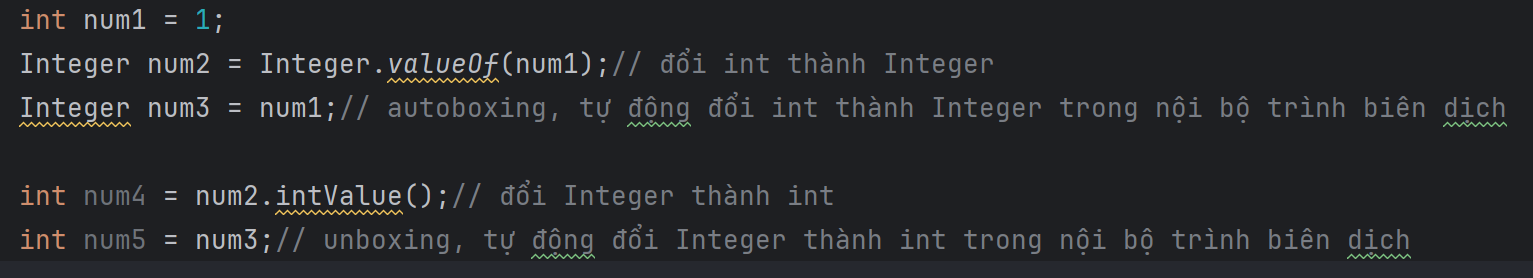
**Data type**

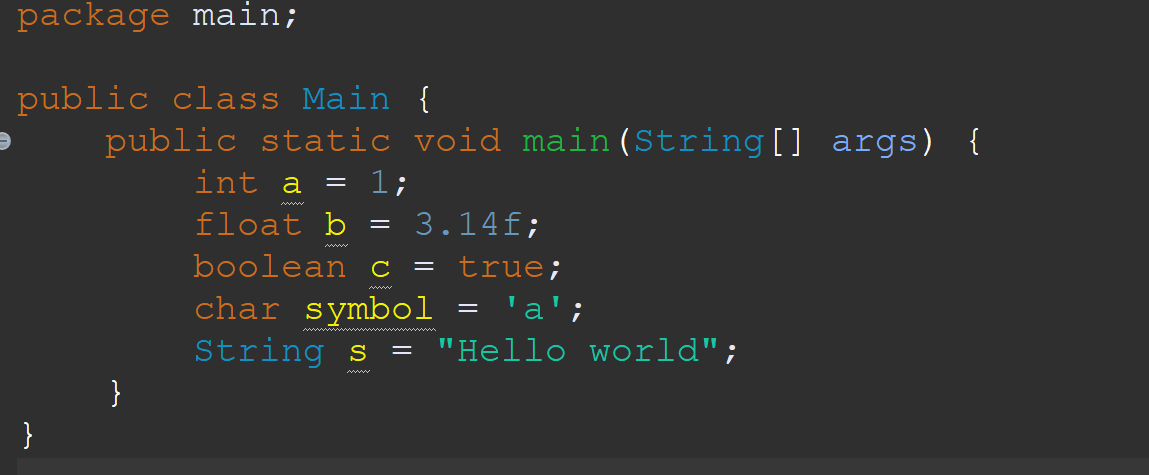


**Lớp wrapper trong Java**

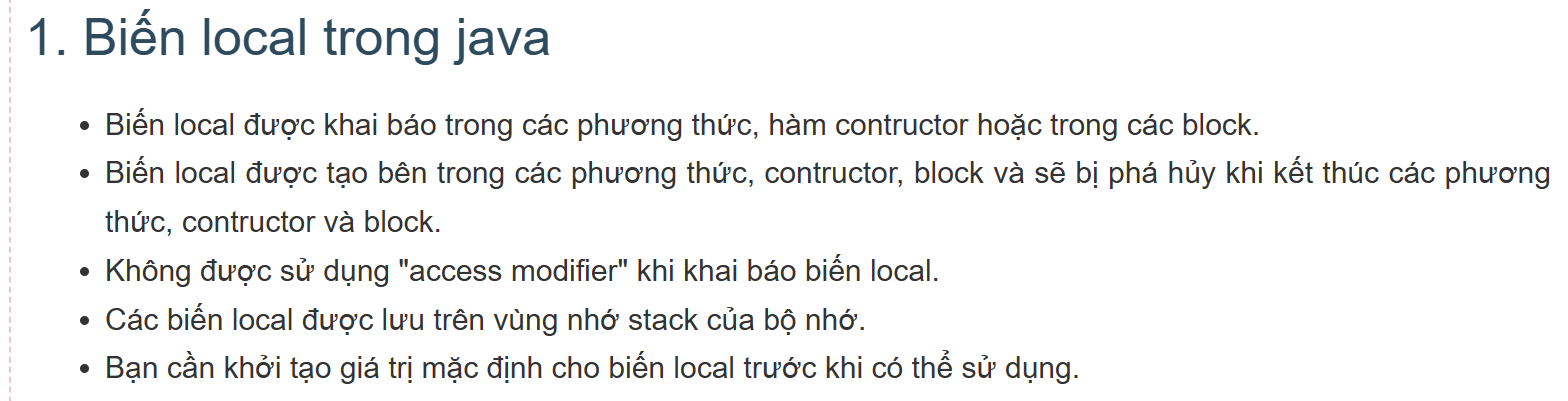
Chuyển kiểu dữ liệu primitive(nguyên thủy) sang non-primitive(đối tượng)

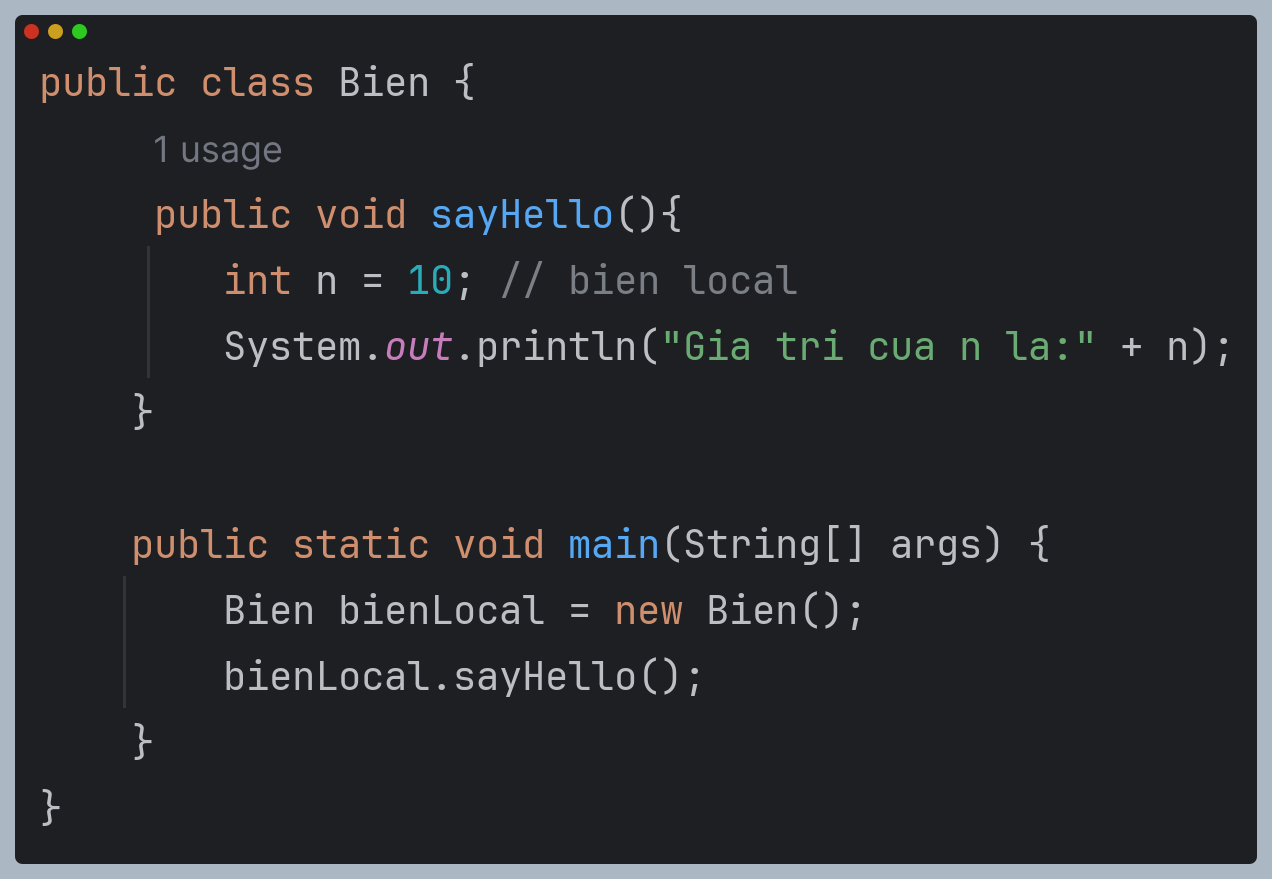


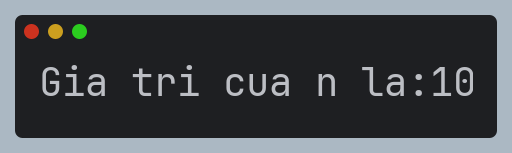


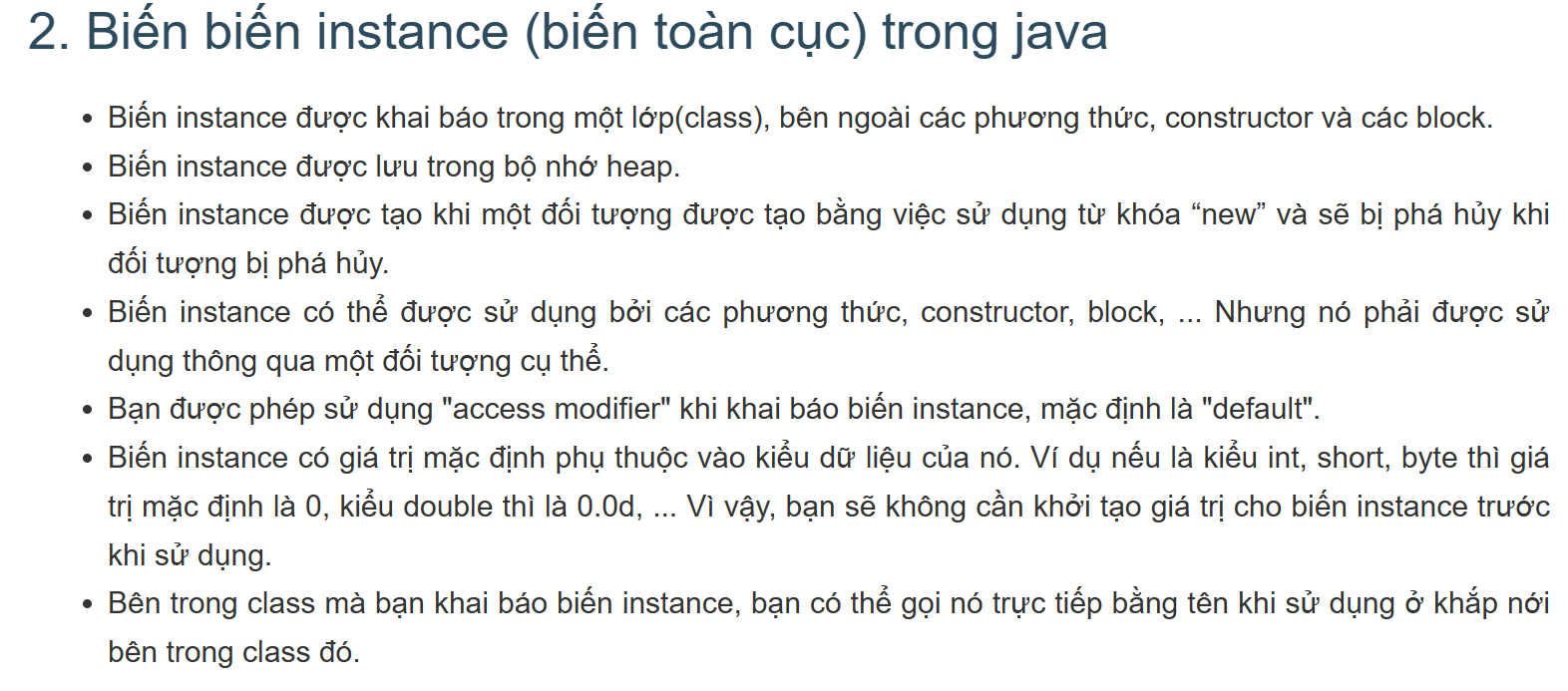


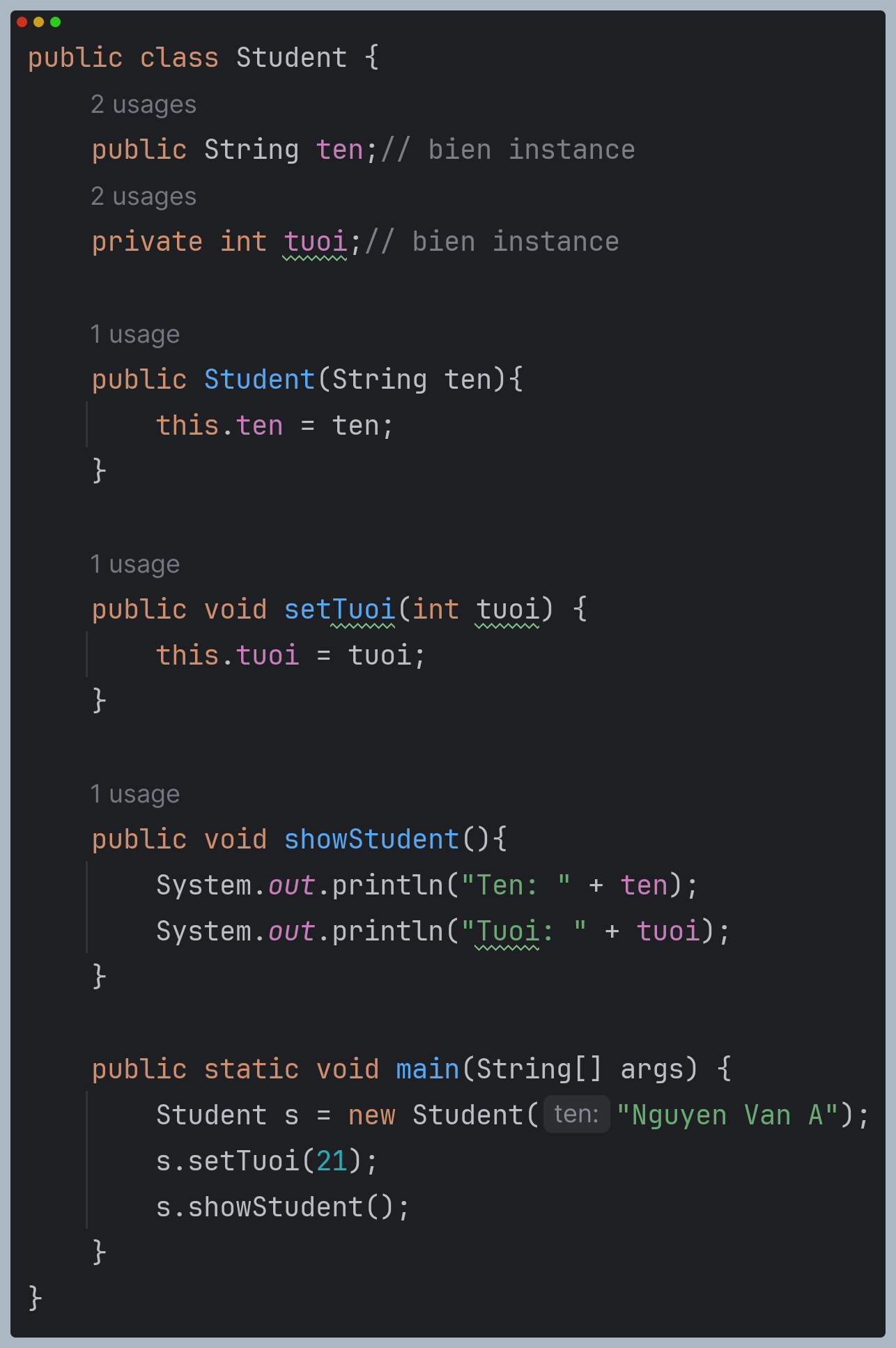
**Biến trong Java**

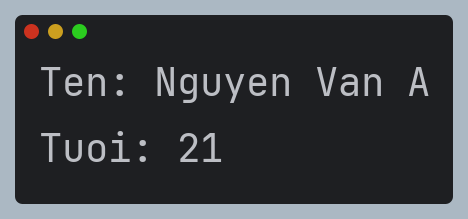


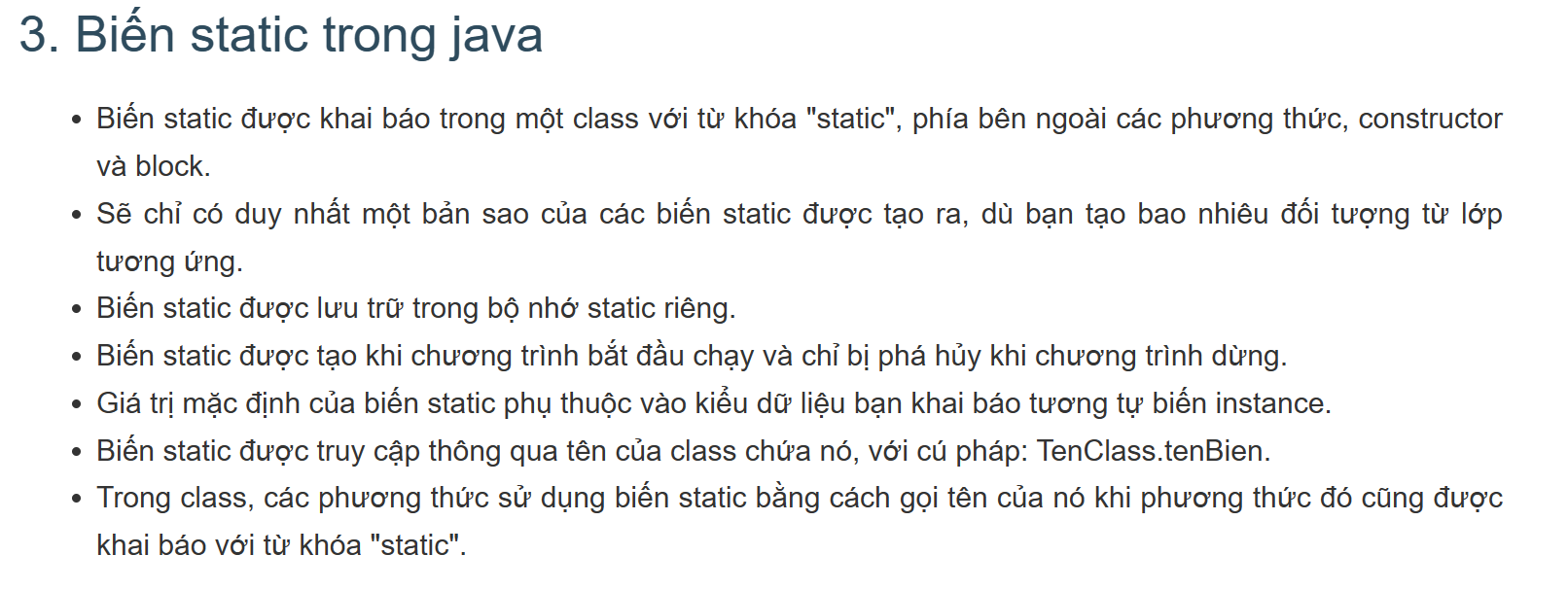


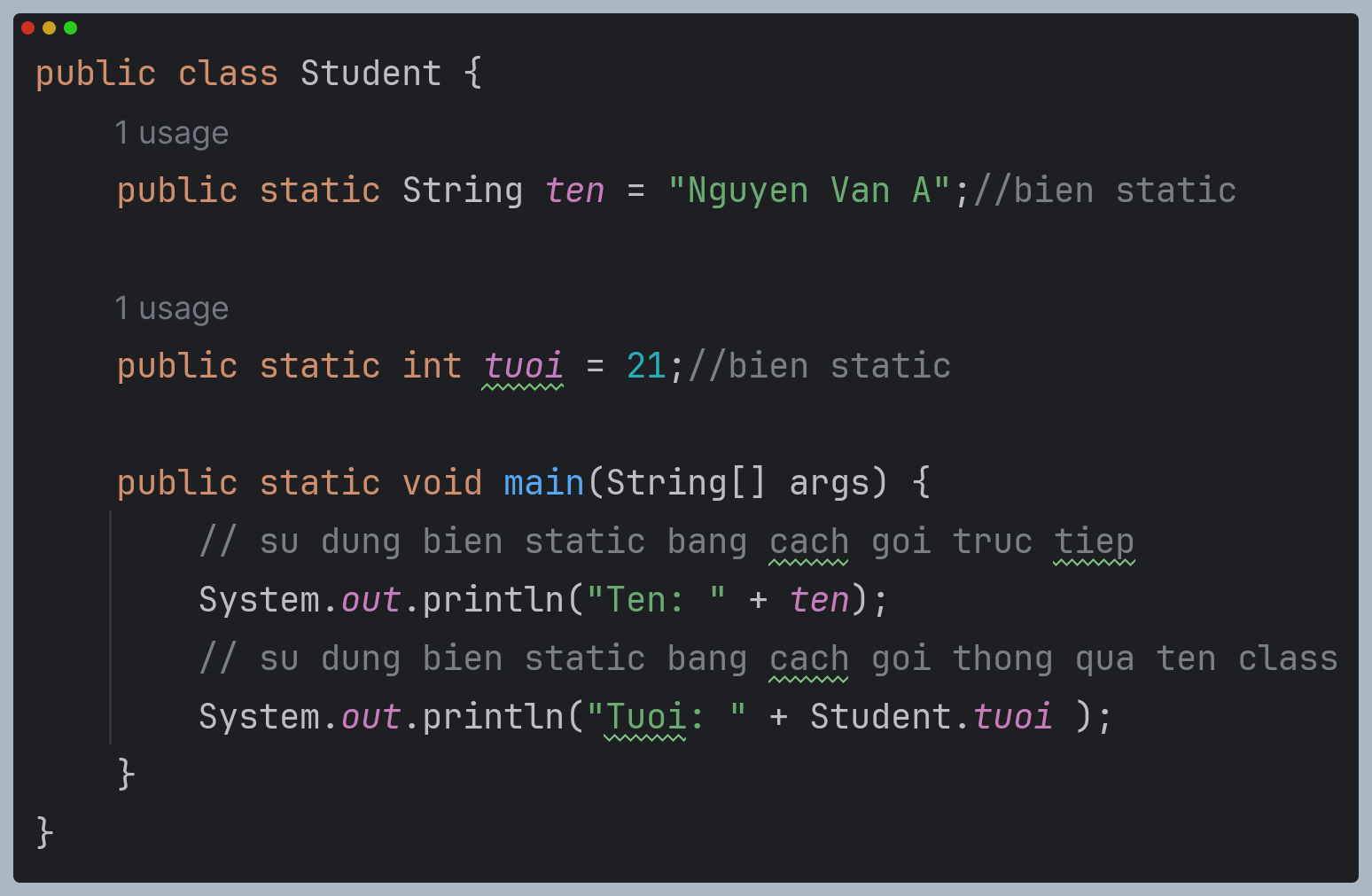


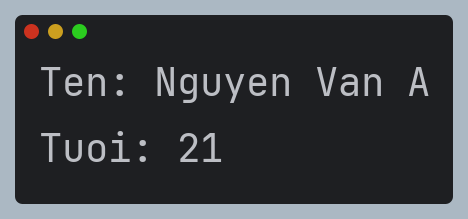












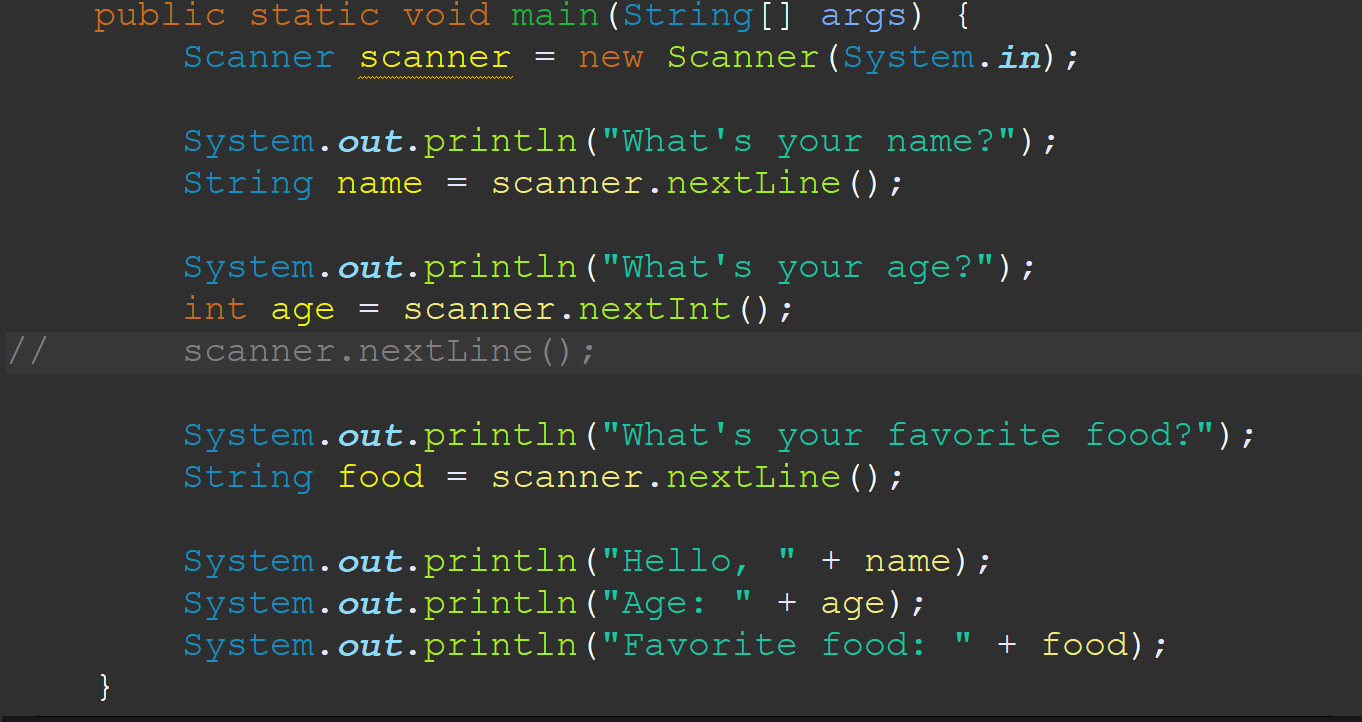
**User input**

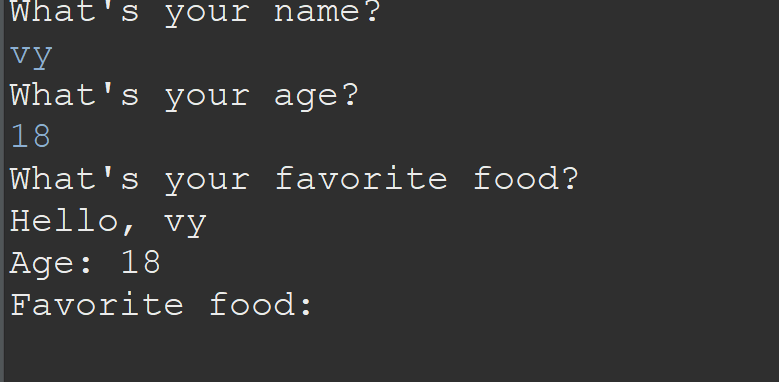




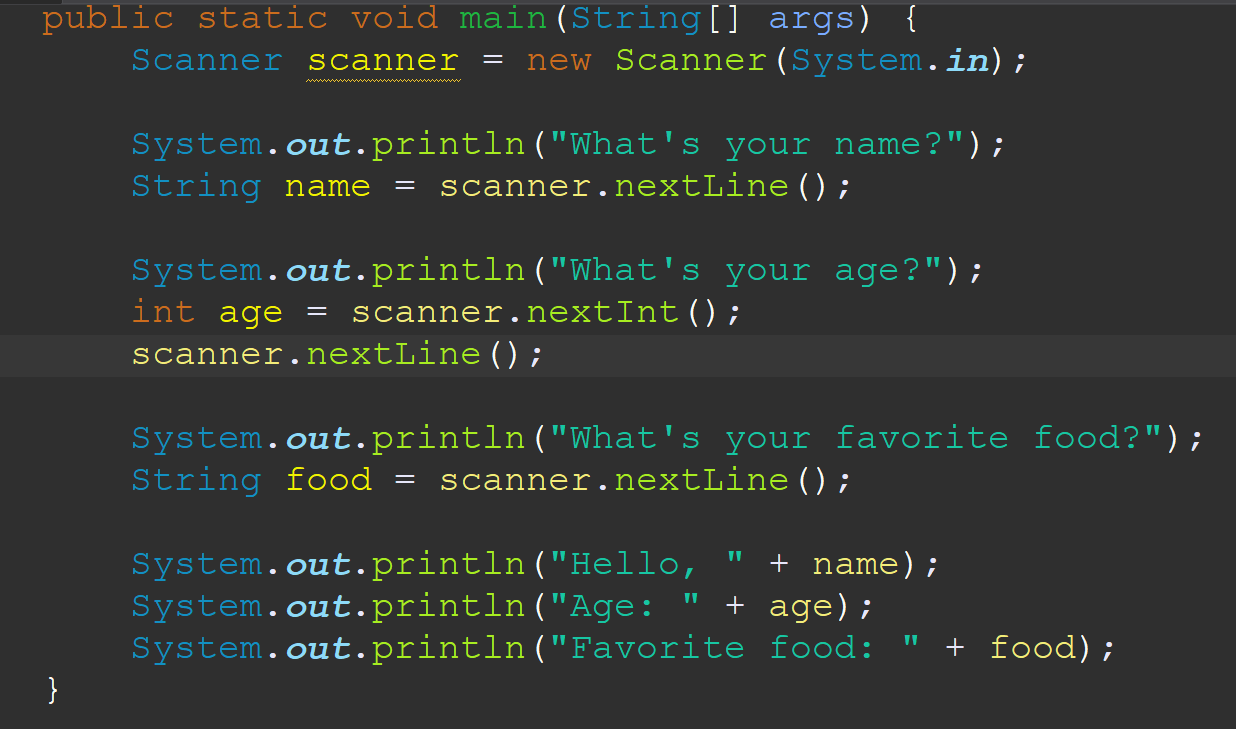
Scanner là lớp dùng để đọc dữ liệu vào

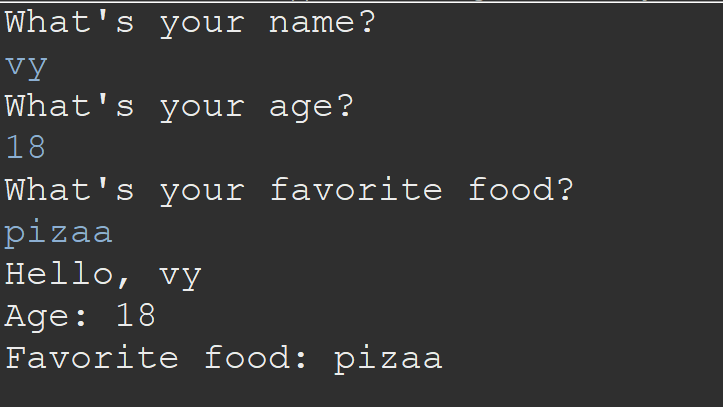
nextLine() -> đọc dữ liệu kể cả '\n'





Favorite food không in ra gì cả.





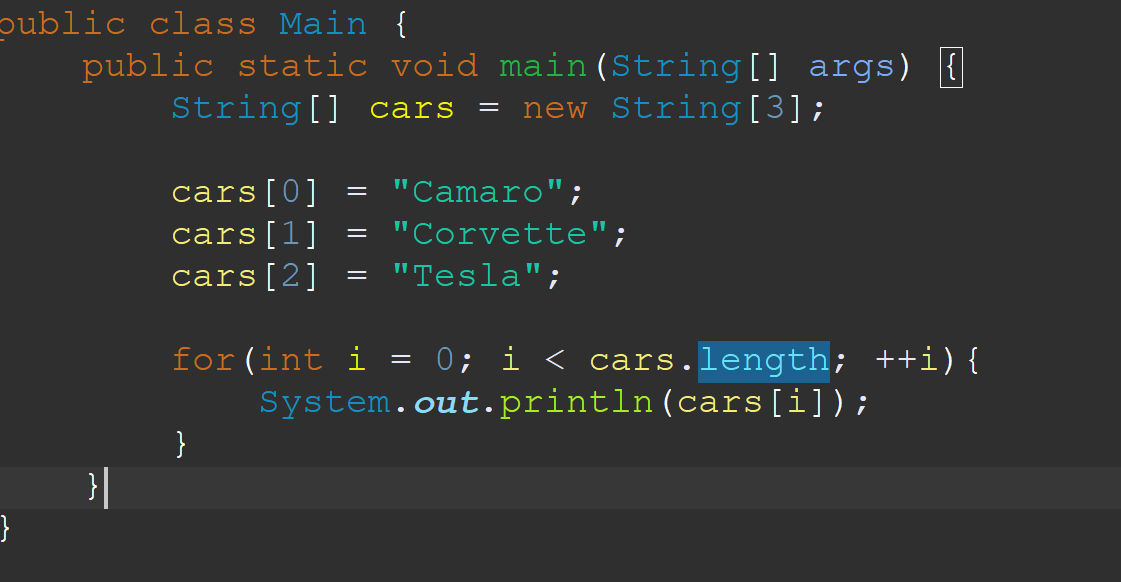
Fixed

**Câu lệnh điều kiện**

If statement, switch, logical operators, while, do…while, for loop giống C\C++

**Java String**



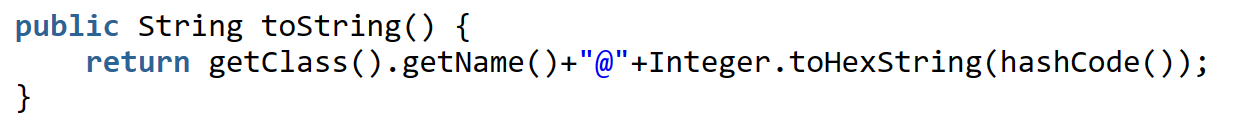


**ToString**

Phương thức toString() trả về chuỗi đại diện của đối tượng

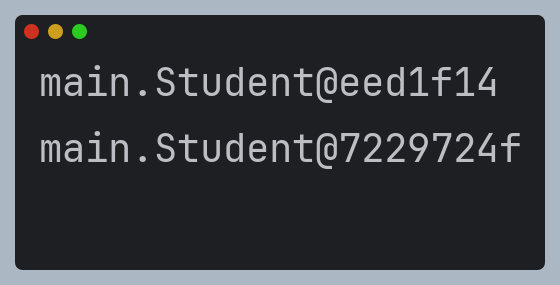
Khi in đối tượng, nội bộ trình biên dịch java sẽ gọi phương thức toString() từ đối tượng đó và trả kết quả mong muốn. Nó có thể là trạng thái của một đối tượng,… phụ thuộc vào việc cài đặt.

Là phương thức của lớp Object, có cú pháp:



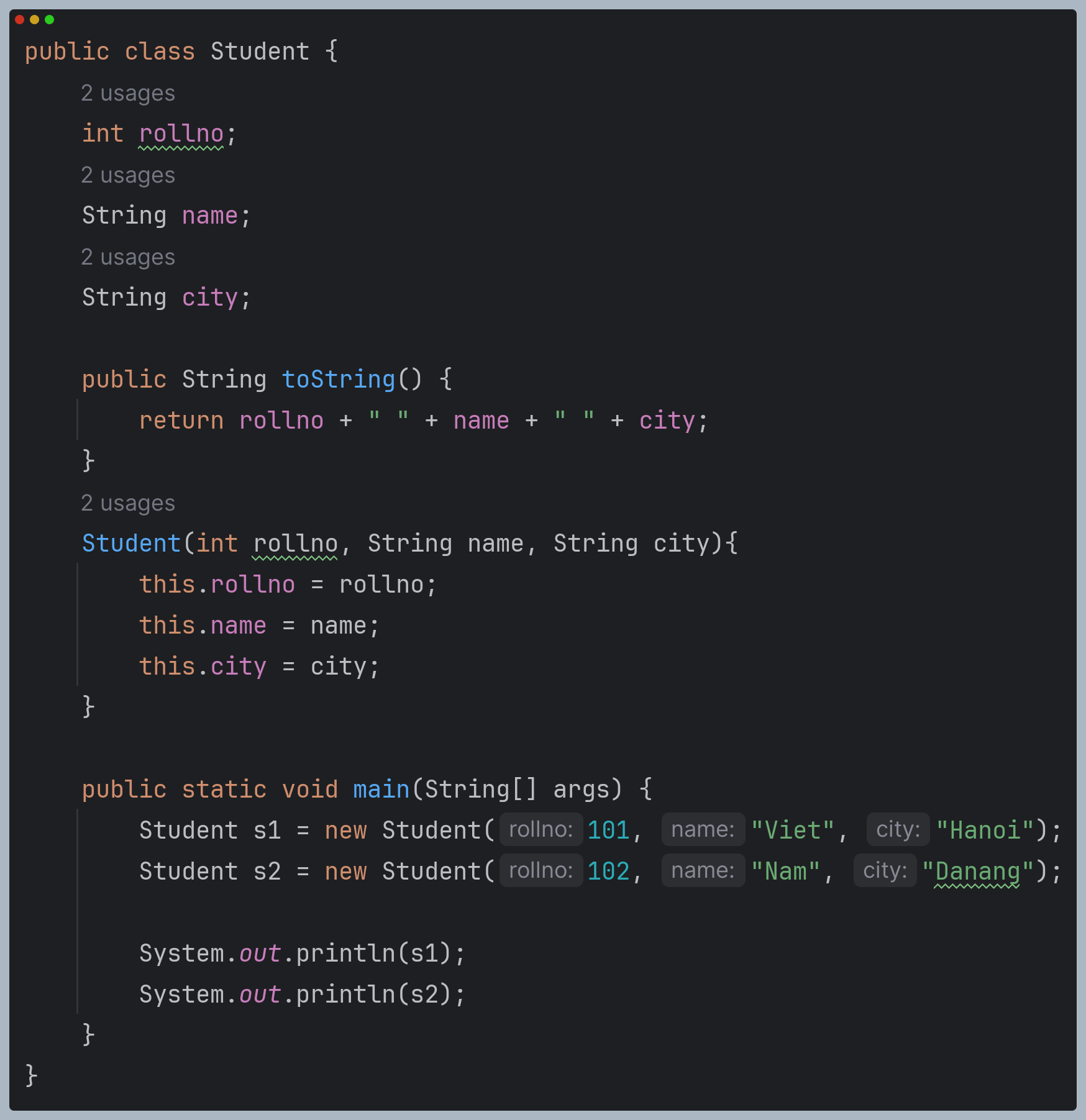
Bằng việc ghi đè phương thức toString() của lớp Object, chúng ta có thể trả về giá trị của đối tượng





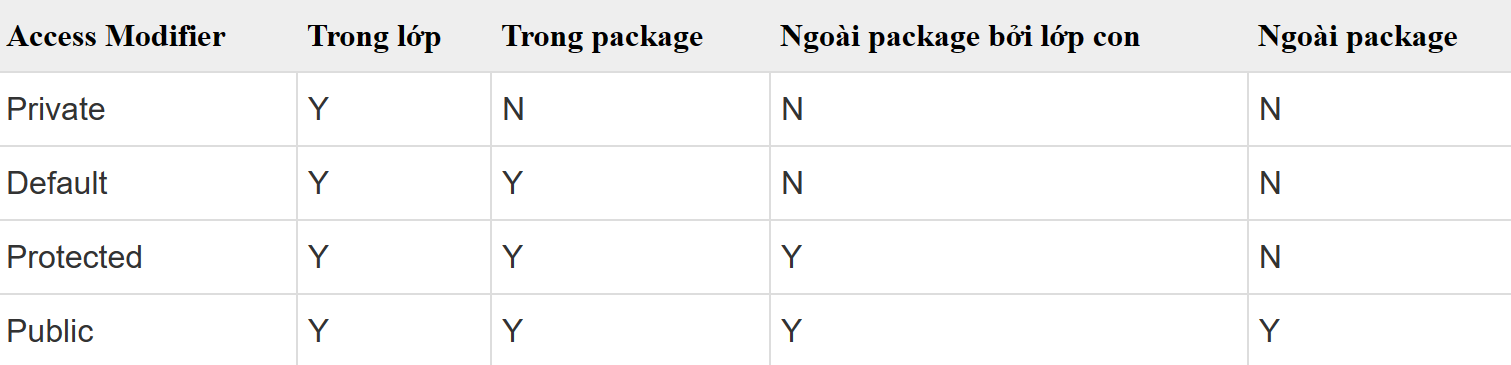
Trình biên dịch sẽ ghi đè s1.toString() thay vì s1, có nghĩa là khi khai báo s1 thì phương thức s1.toString() được gọi, tương tự đối với s2

VD thêm





**Phạm vi truy cập**



Private: truy cập nội bộ class

Default: Phạm vi mặc định, truy cập nội bộ trong package

Protected: phạm vi từ trong và ngoài package, phải thông qua tính kế thừa

Public: có thể truy cập trong Project, khác package phải import

Từ khóa Statisc

Biến của lớp:

Biến Static dùng để tham chiếu thuộc tính chung của tất cả các đối tượng, vd: tên công ty của nhân viên

Hàm Static

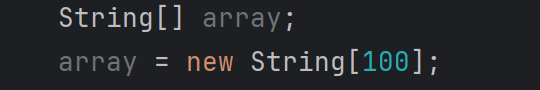
Tộn tại độc lập với các đối tượng, có thể gọi thẳng từ lớp mà không cần đến một tham chiếu đối tượng nào

**Mảng:**

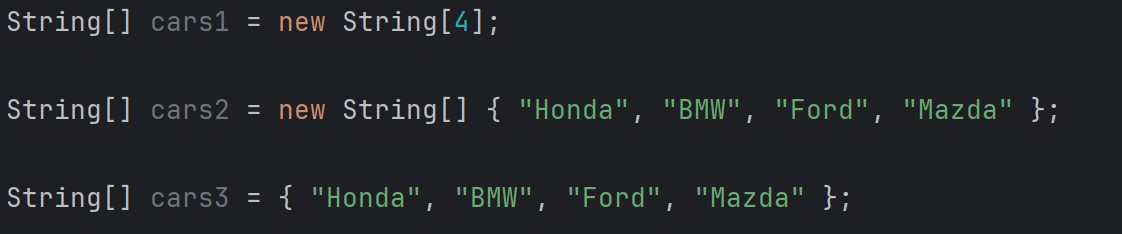
Khai báo: <kiểu dữ liệu>[] <tên mảng>;



Cấp phát bộ nhớ cho mảng:<tên mảng> = new <kiểu dữ liệu>[];

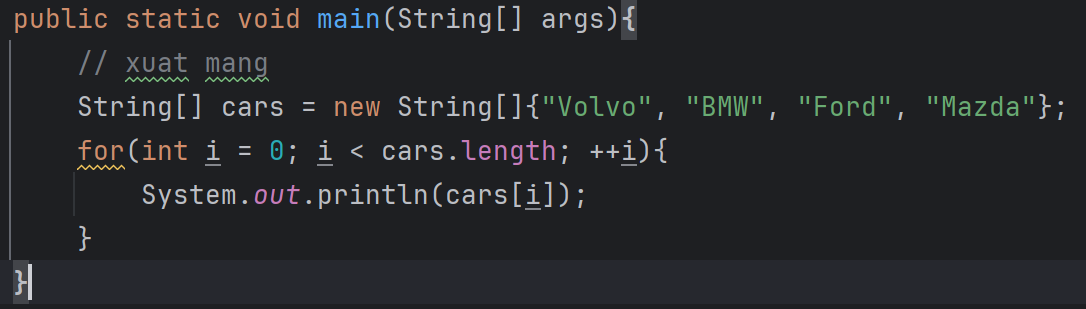


Các kiểu khai báo



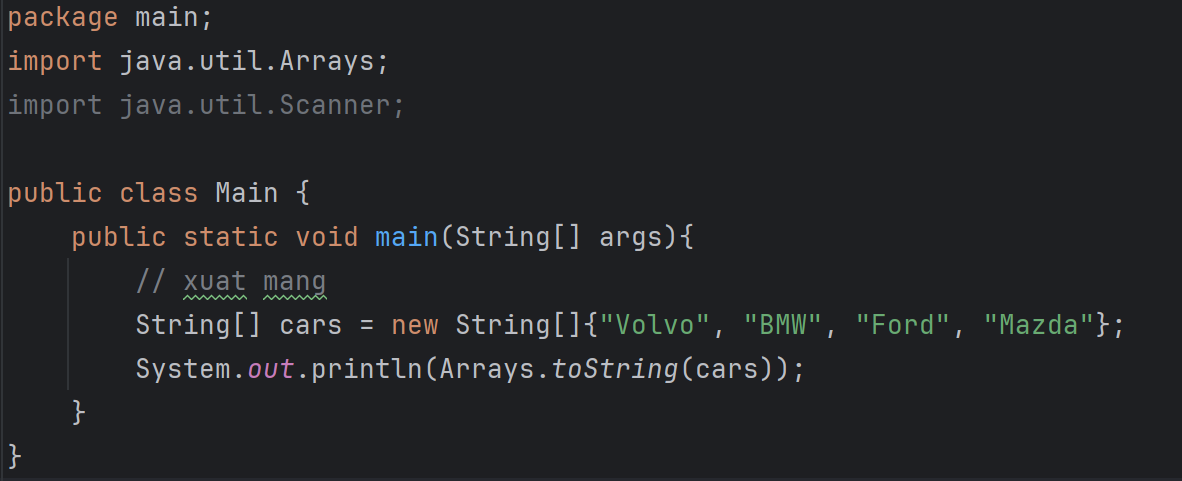
Thao tác trên mảng:

Xuất: bằng duyệt vòng lặp for





Bằng toString

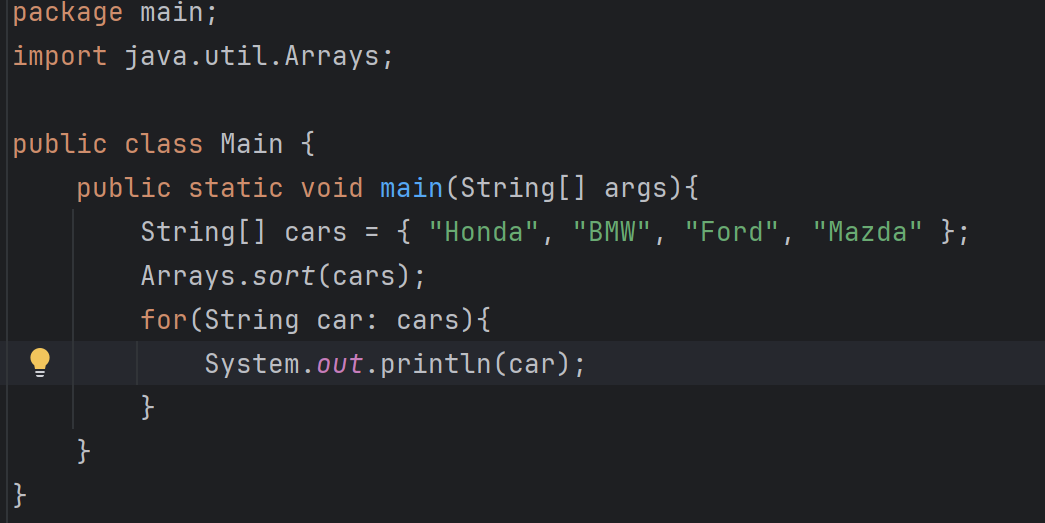




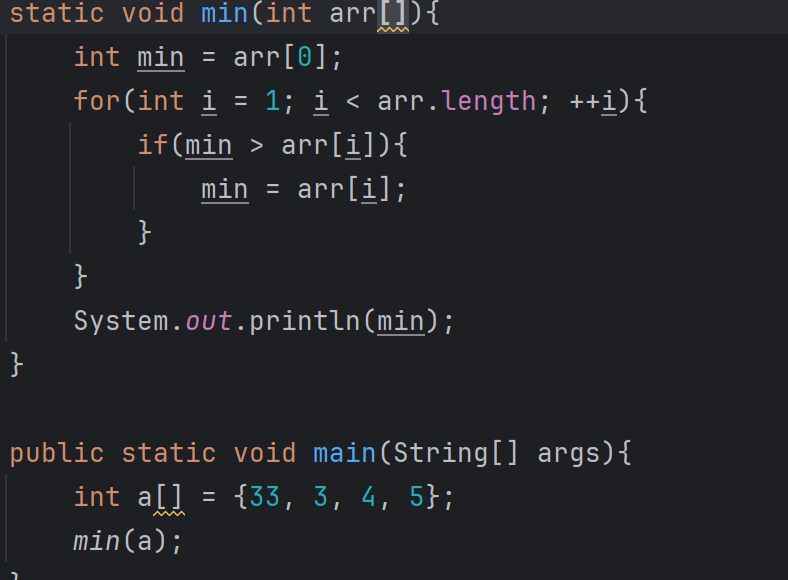
Duyệt mảng bẳng For-each:



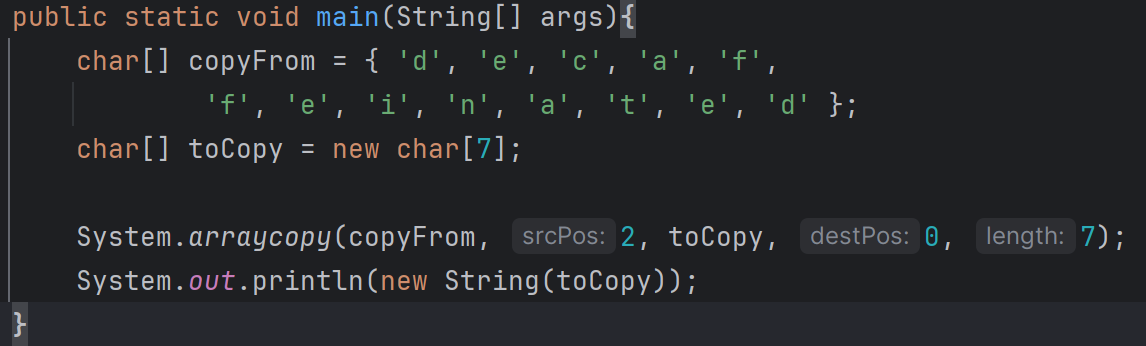
Sắp xếp mảng



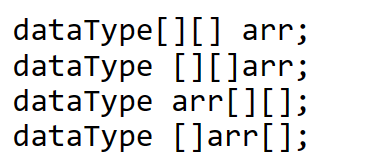
Truyền mảng vào phương thức



Sao chép mảng



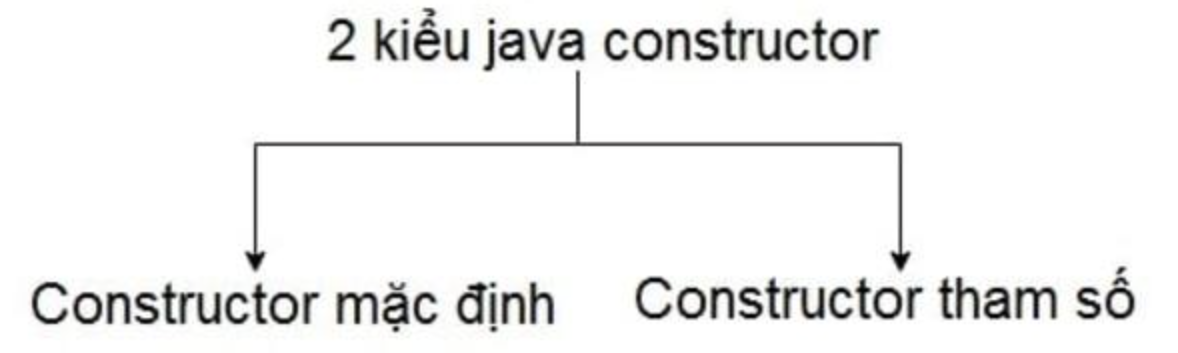
Khai báo mảng đa chiều



**Constructor**

Phương thức khỏi tạo và trả về một đối tượng của lớp mà nó định nghĩa

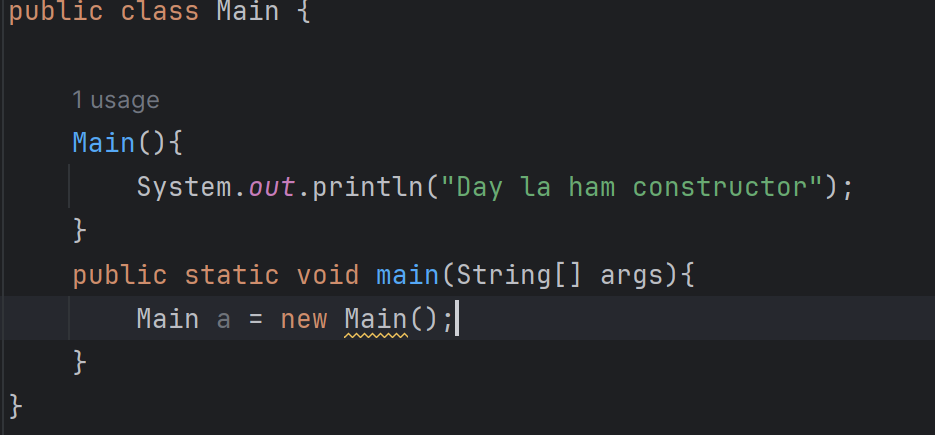
Tên constructor phải giống tên lớp chứa nó



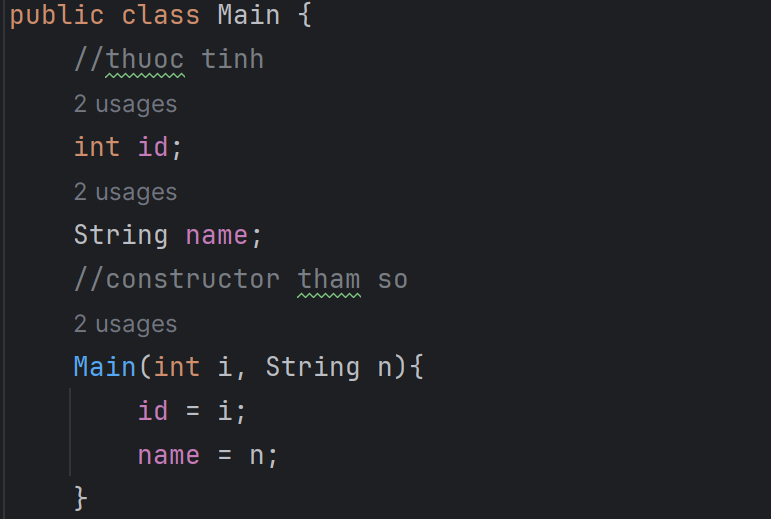
Constructor mặc định: hàm khởi tạo không có tham số

Cú pháp:

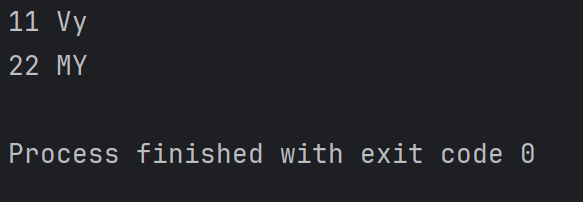
<class\_name>() {}



Constructor tham số: cung cấp giá trị riêng biệt đối với các đối tượng java khác nhau, cung cấp giá trị giống nhau đối với các đối tượng java khác nhau

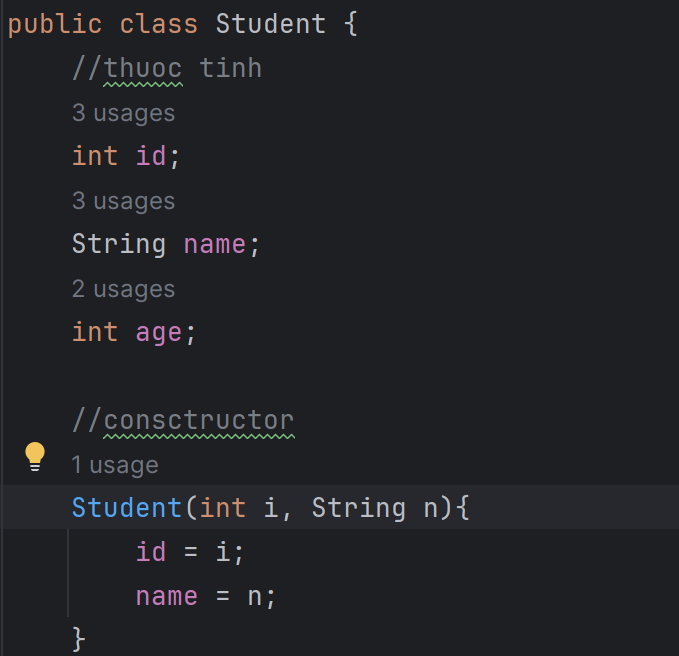


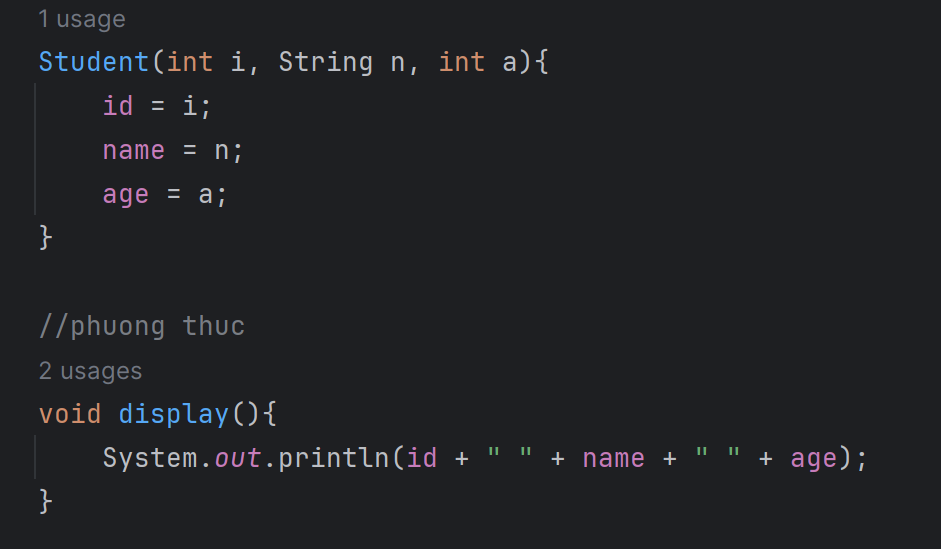


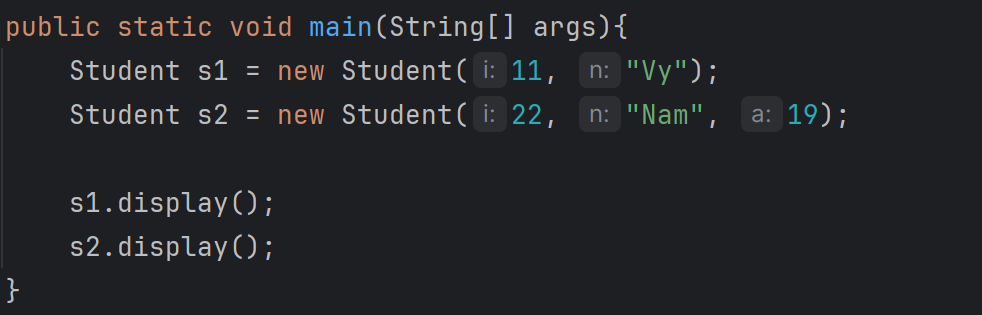


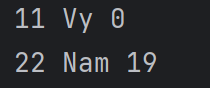
Constructor overloading

Tạo nhiều constructor trong cùng một lớp với danh sách tham số truyền vào khác nhau

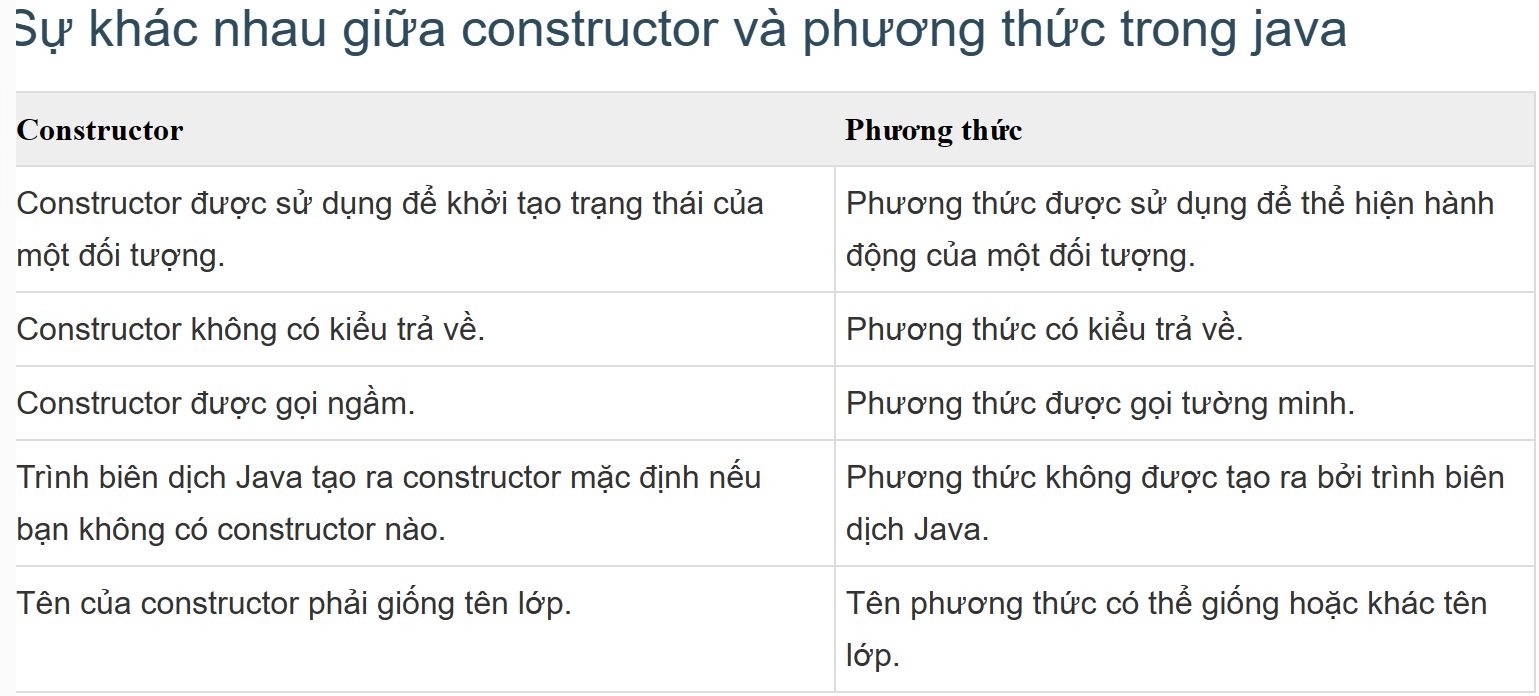






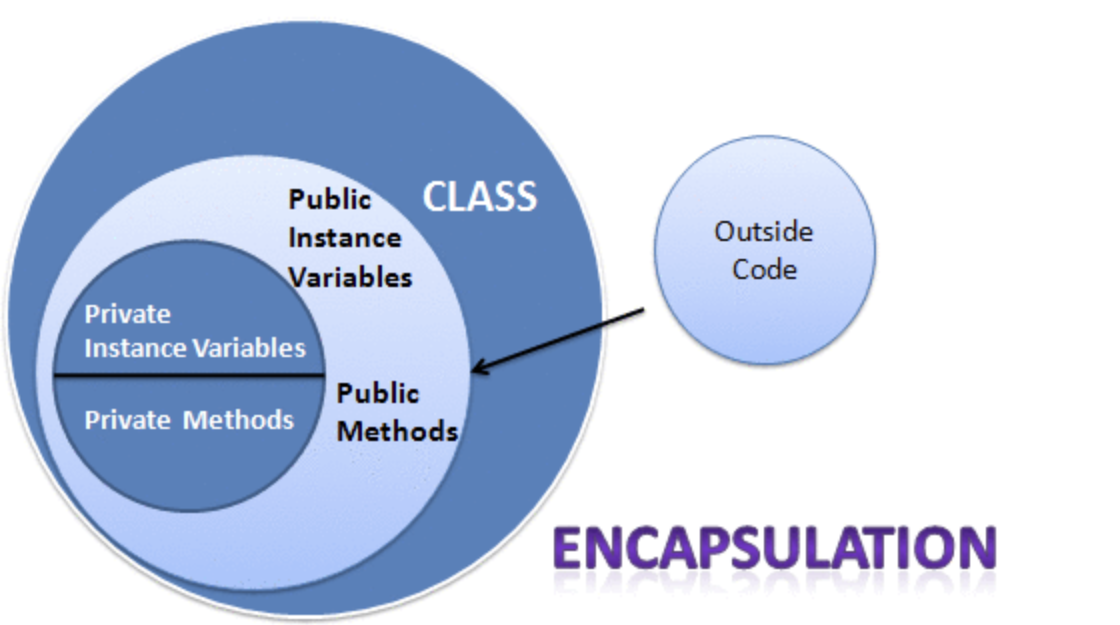


(xem thêm kết hợp giữa constructor mặc định và constructor có tham số)



**Các tính chất của Java OOPs**

*Tính đóng gói*: là kỹ thuật giấu thông tin ko liên quan và hiển thị ra thông tin liên quan

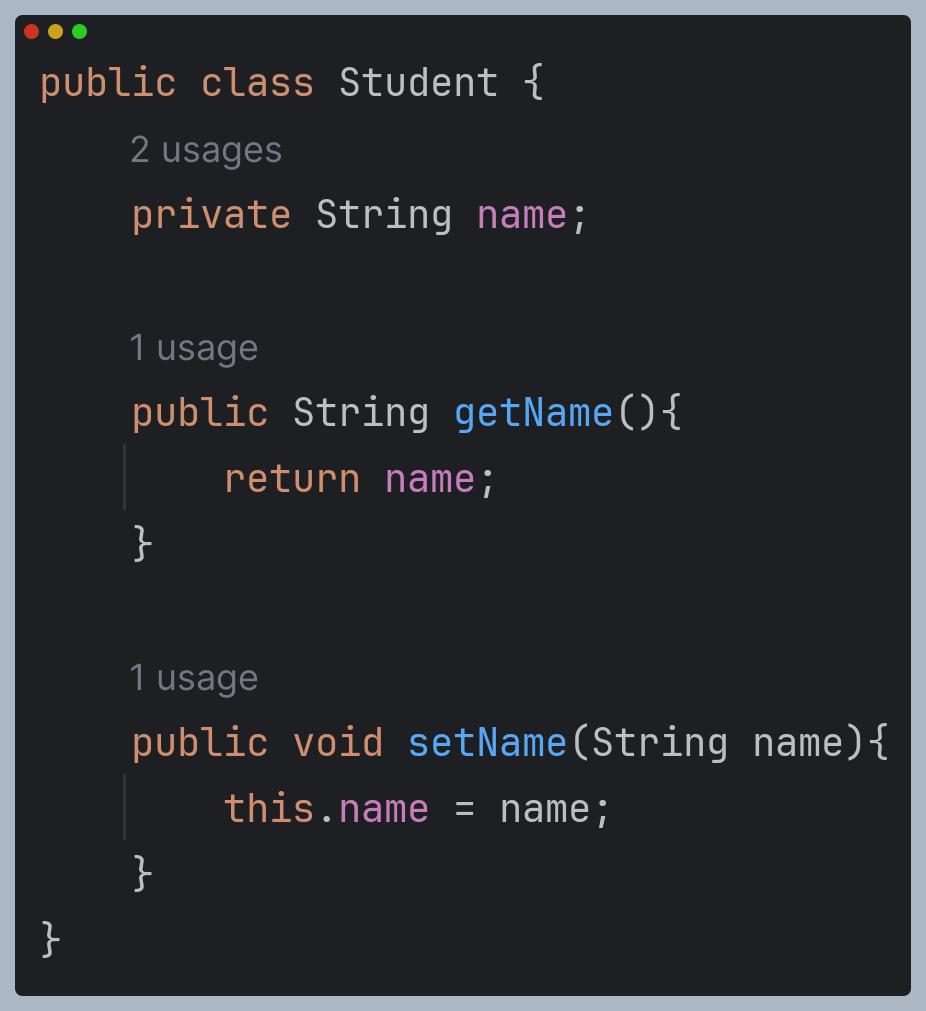


Instance value là biến khai báo bên trong class nhưng bên ngoài method

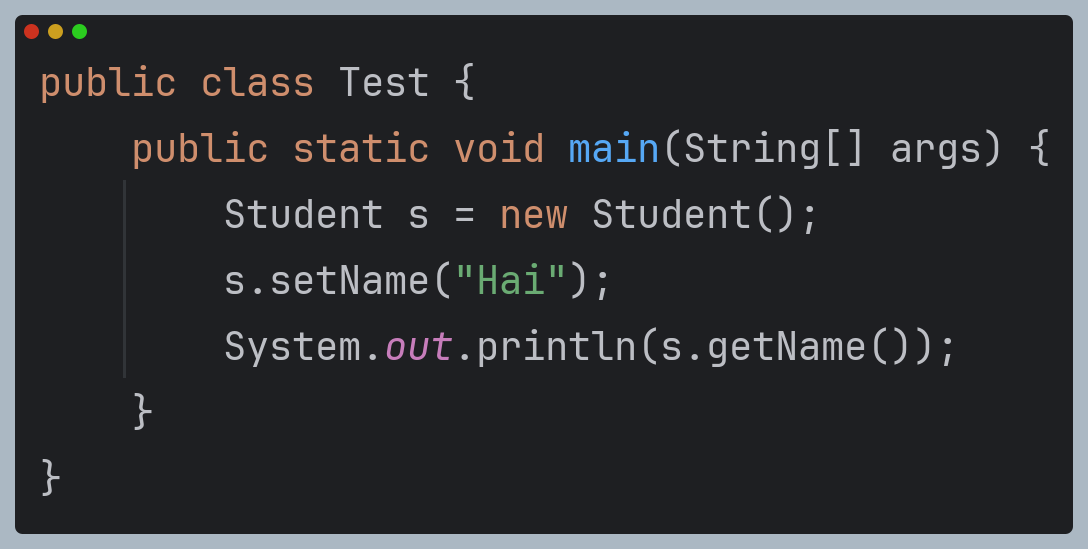
Đóng gói được sử dụng để bảo vệ trạng thái bên trong của một đối tượng. Bởi việc giấu các biến biểu diễn trạng thái của đối tượng. Việc chỉnh sửa đối tượng được thực hiện, xác định thông qua các phương thức. Hơn nữa, việc giấu các biến thì các lớp sẽ không chia sẽ thông tin với nhau. Điều này làm giảm số lượng khớp nối có trong một ứng dụng.

Có thể tạo lớp read-only hoặc write-only bằng phương thức setter hoặc getter

File Student.java



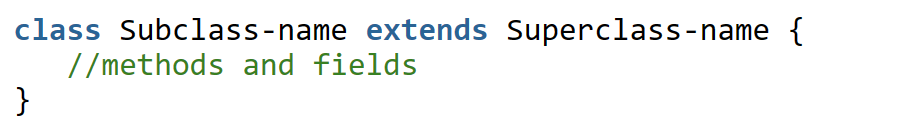
File Test.java

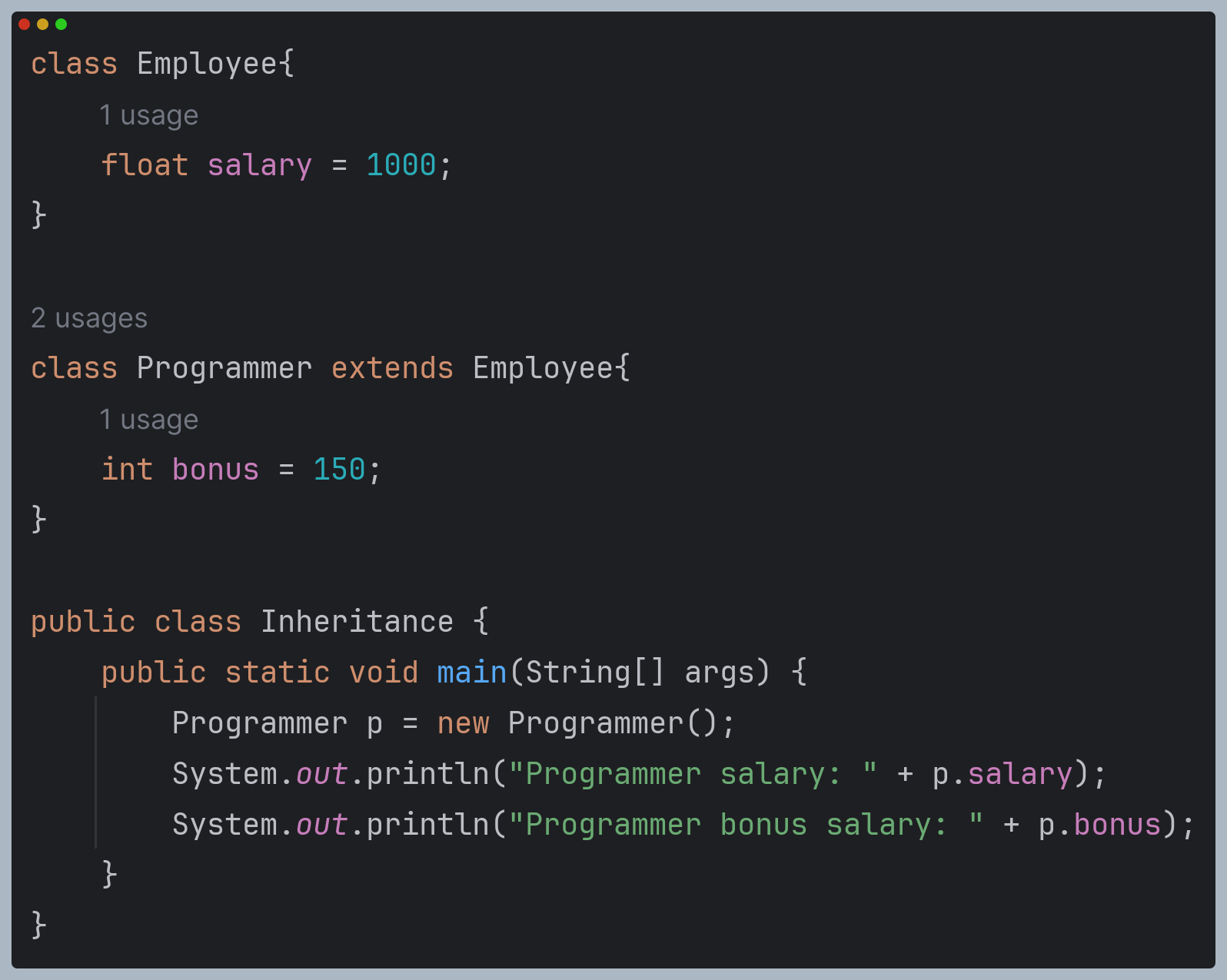


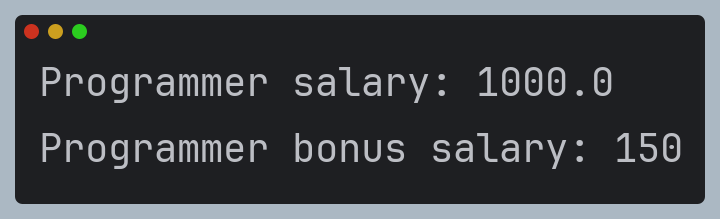


*Tính kế thừa:* sự liên quan giữa 1 class với nhau: superclass(class cha) và subclass(class con). Khi kế thừa subclass được hưởng tất cả phương thức và thuộc tínhcủa superclass, nhưng chỉ được truy cập vào các thành viên public và protected, chứ không được truy cập vào private

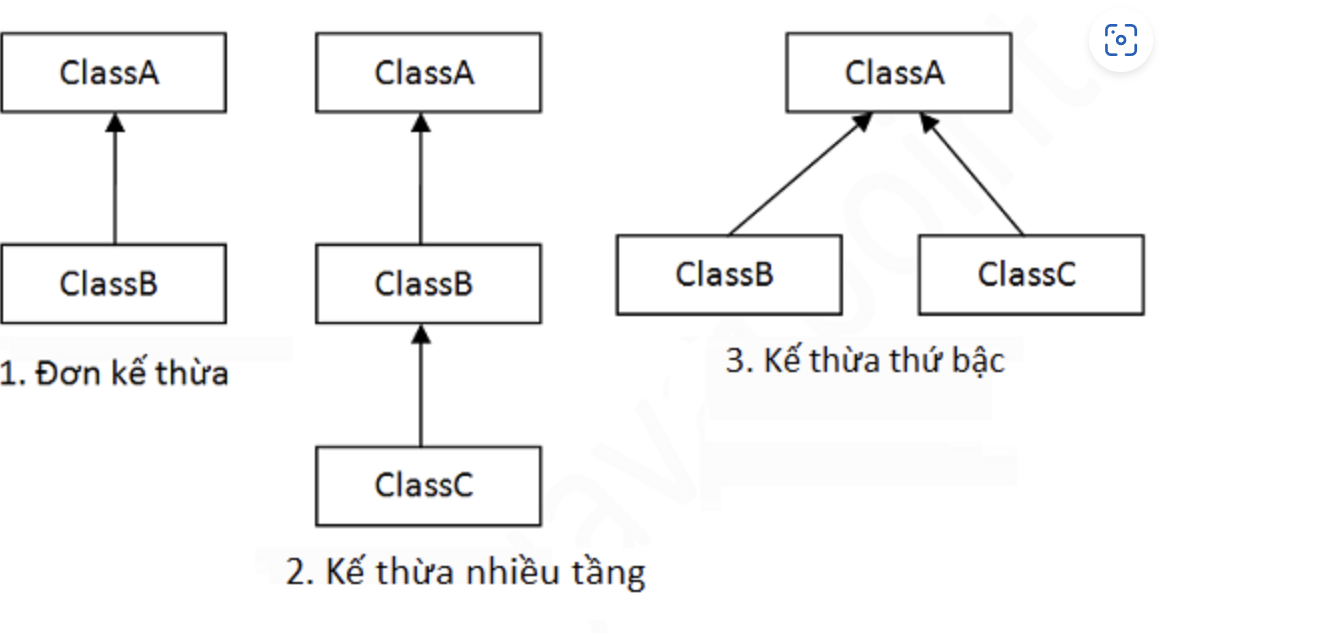
Cú pháp:





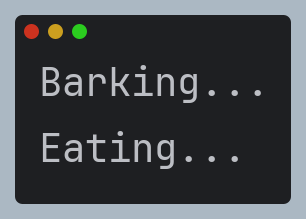


Các kiểu kể thừa

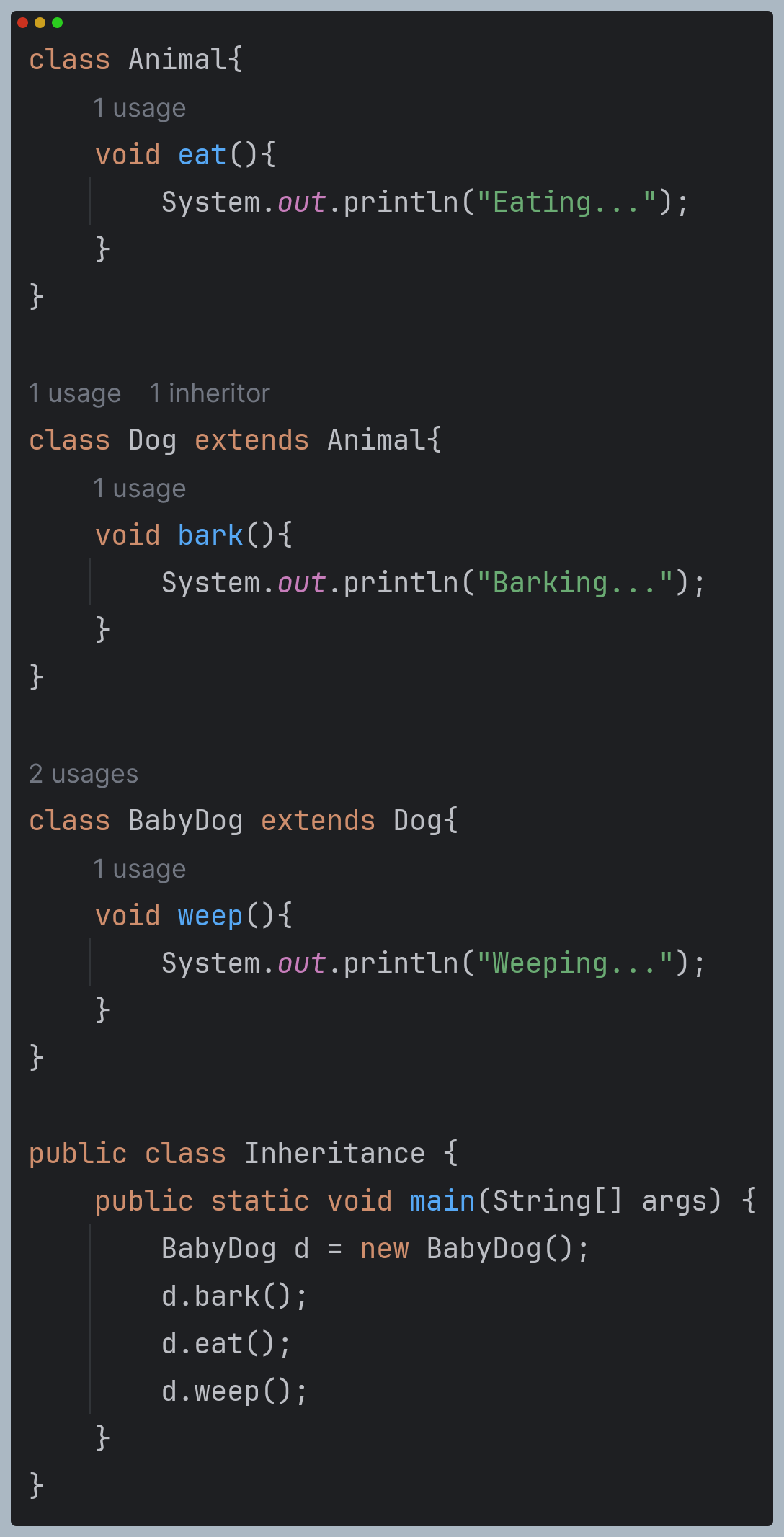


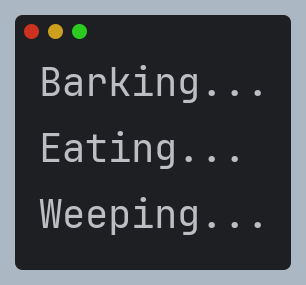
Đơn kế thừa



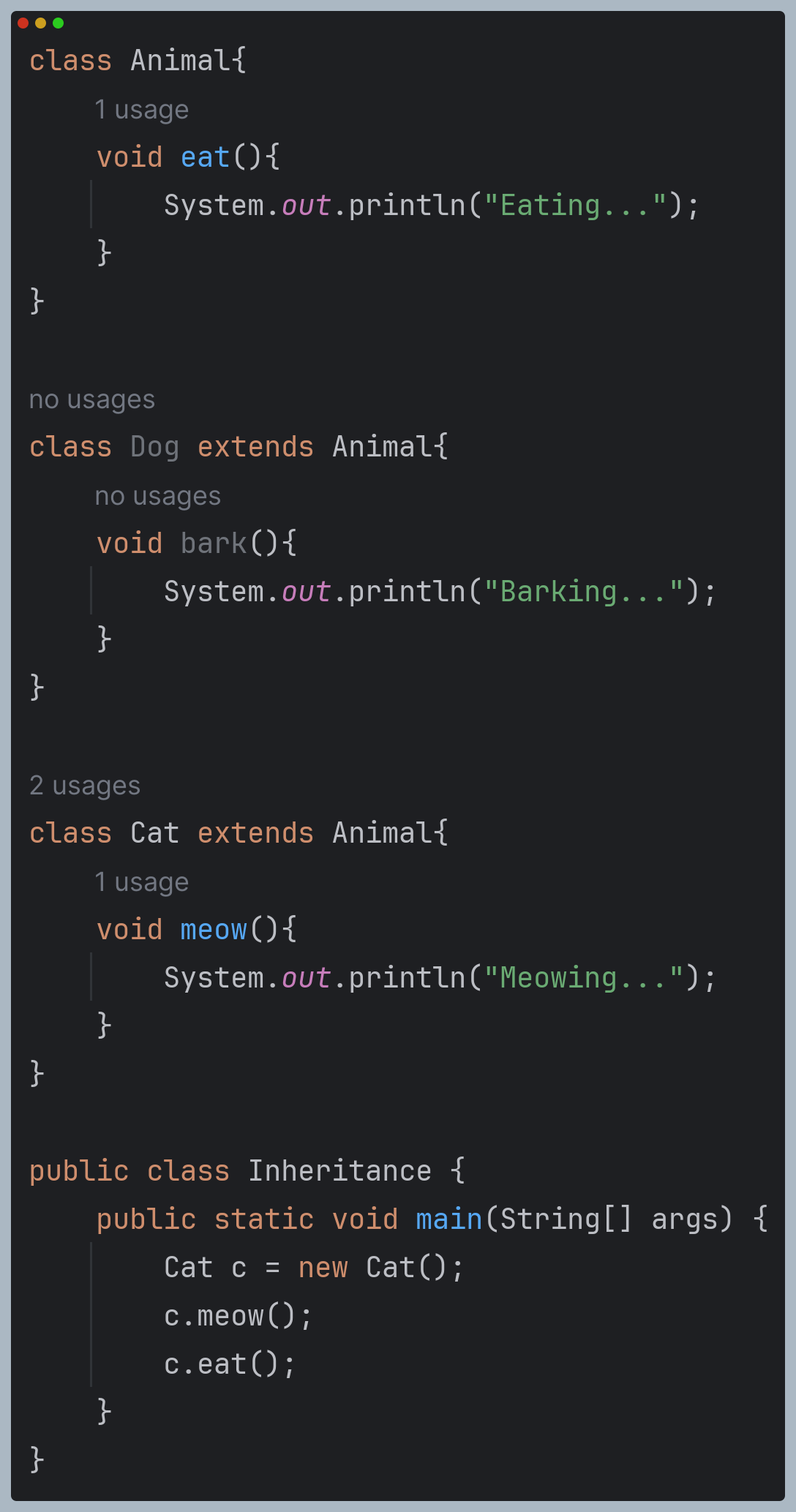


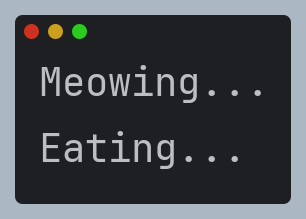
Kế thừa nhiều tầng





Kế thừa thứ bậc

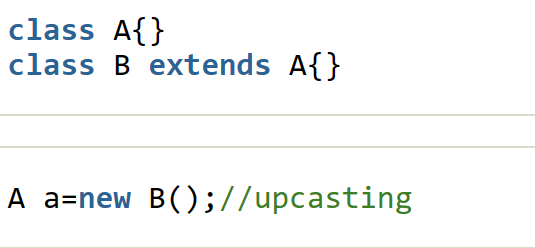


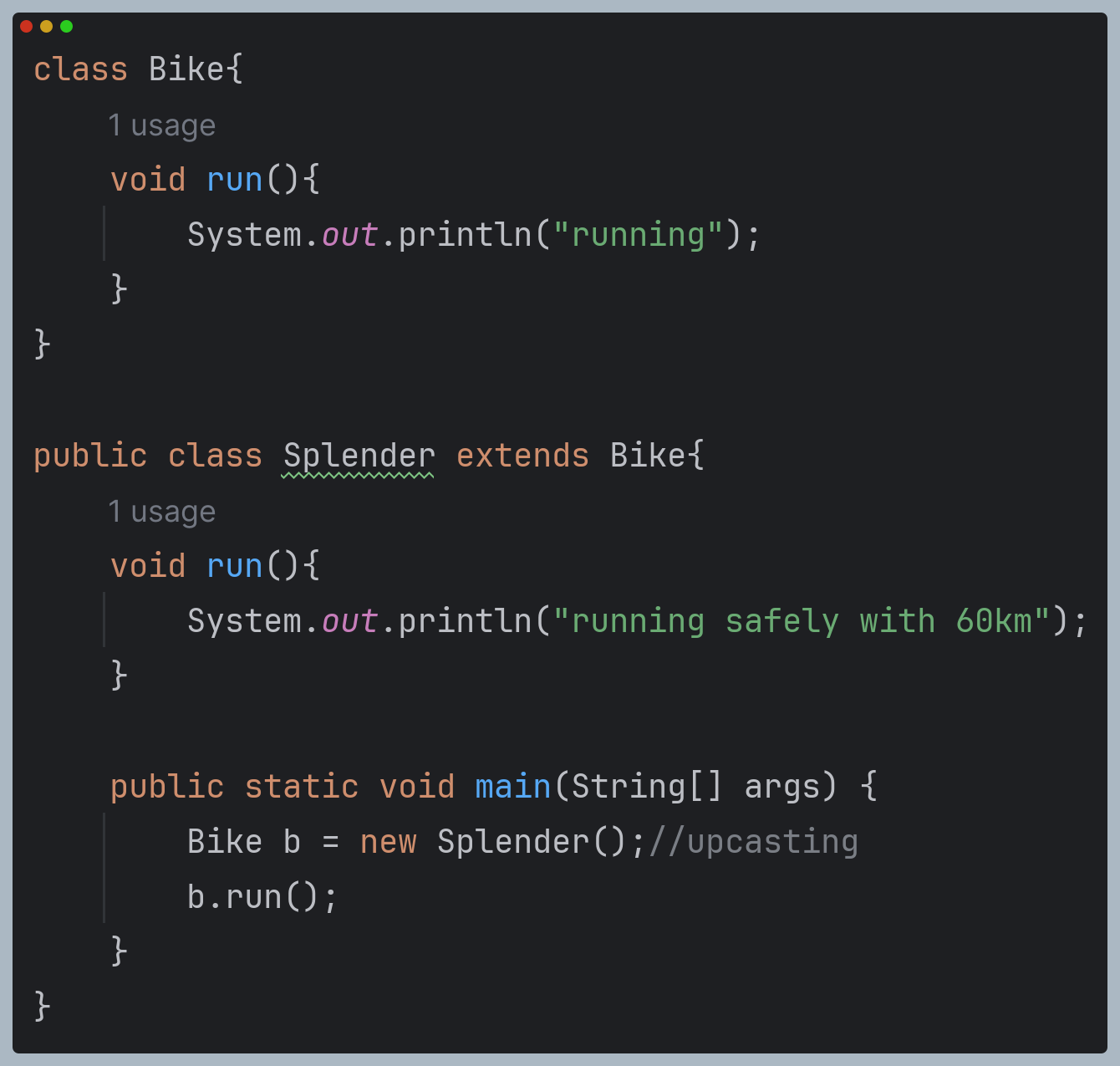


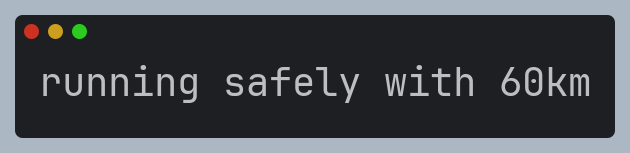
*Tính đa hình*: có 2 loại: đa hình lúc biên dịch và đa hình lúc thực thi

Đa hình lúc runtime: quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình. Phương thức được ghi đè được gọi thông qua biến tham số của superclass

Upcasting: khi biến tham chiếu của superclass tới đối tượng của subclass







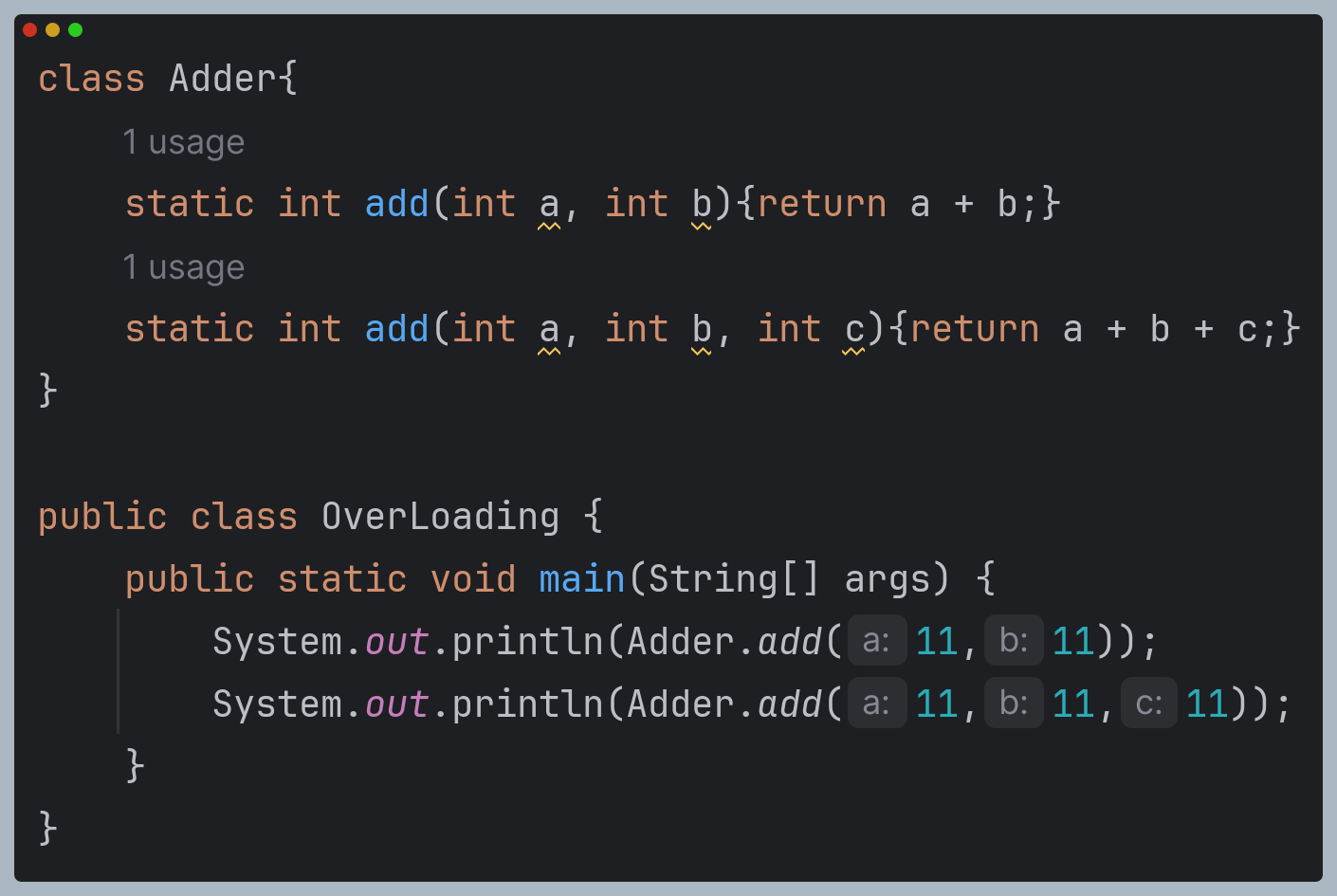
Lớp Splender kế thừa lớp Bike và ghi đè phương thức run() của nó. Gọi phương thức run() bởi tham chiếu của superclass. Khi nó tham chiếu tới đối tượng của subclass và phương thức subclass ghi đè phương thức của superclass, subclass được gọi là runtime.

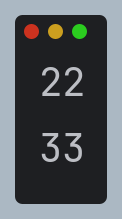
Đa hình lúc runtime: phương thức bị ghi đè không là thành viên dữ liệu, vì thế đa hình tại runtime không thể có được bởi thành viên dữ liệu.

*Nạp chồng phương thức(Overloading)*: nạp chồng xảy ra nếu một lớp có nhiều phương thức có tên giống nhau nhưng khác nhau về kiểu dữ liệu hoặc số lượng tham số.

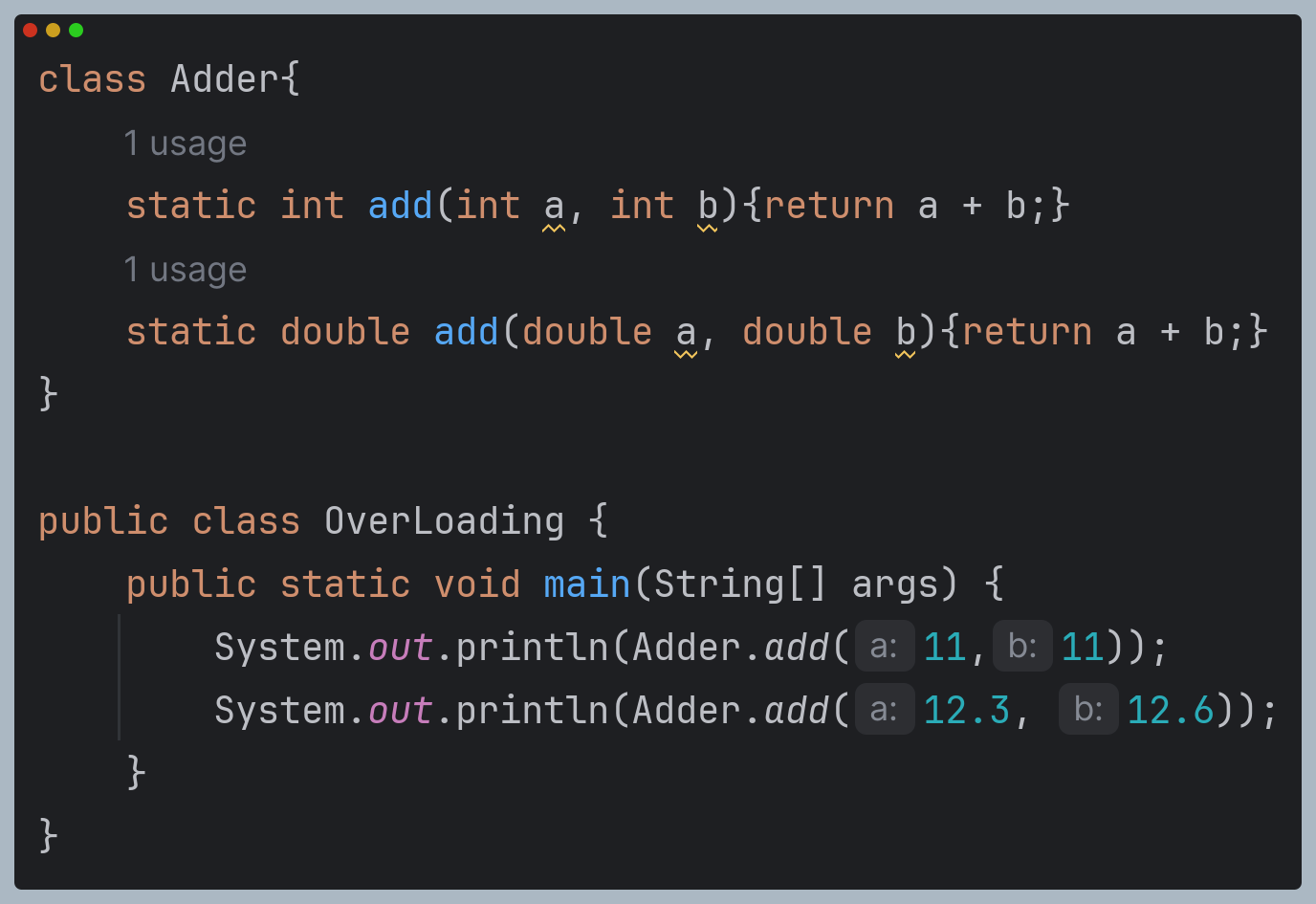
Các cách nạp chồng:

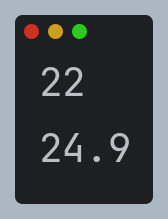
Thay đổi số lượng các tham số



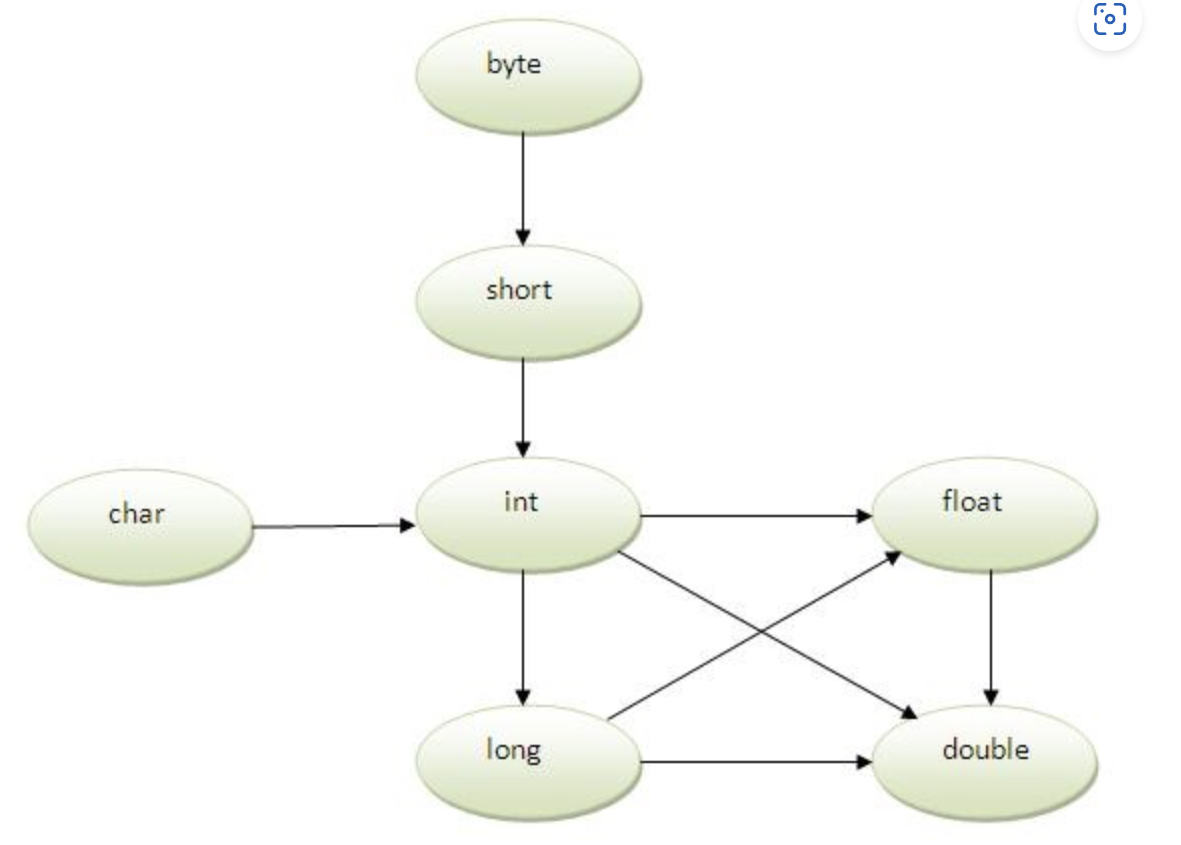


Thay đổi kiểu dữ liệu của tham số

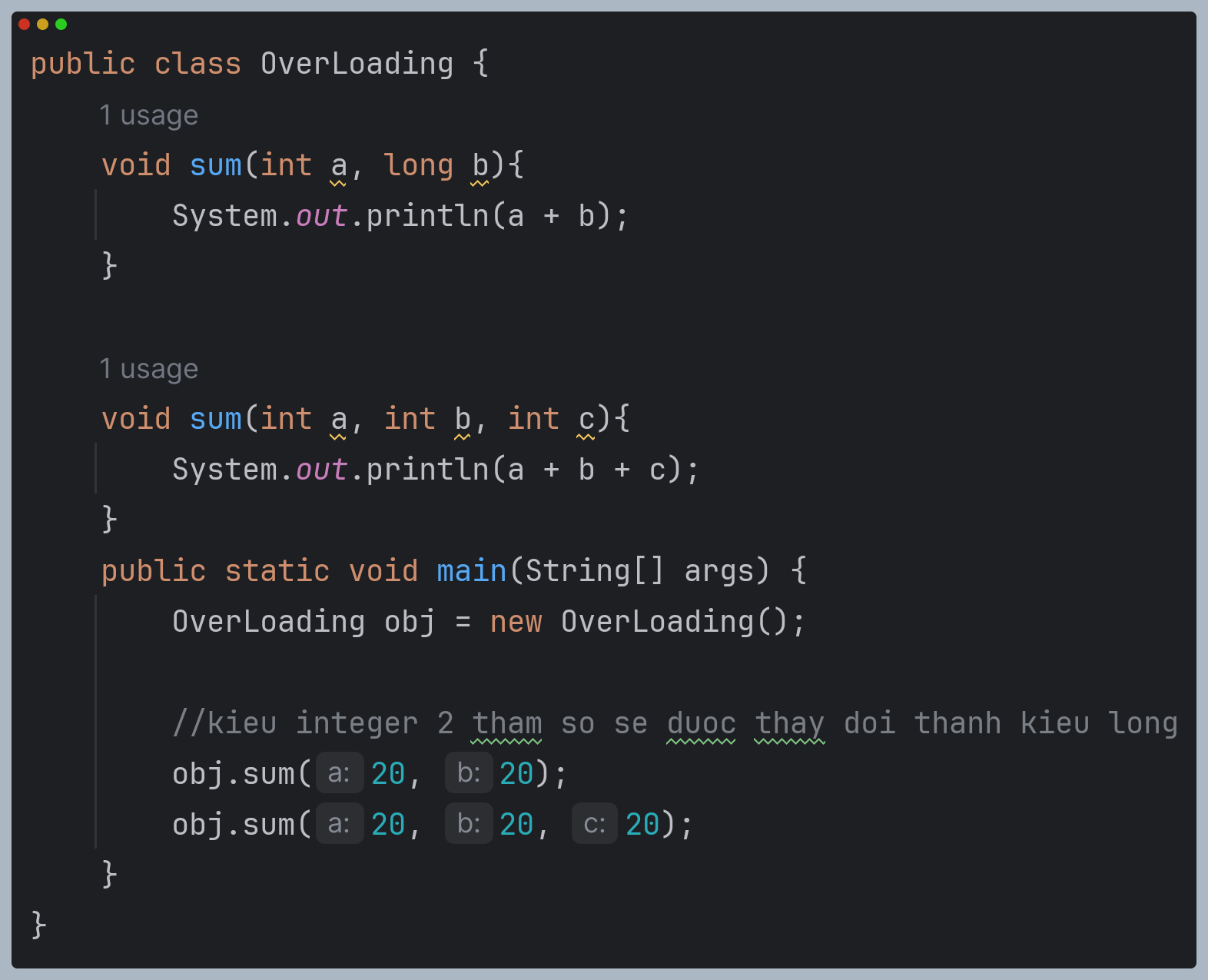


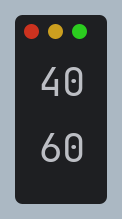


Nạp chồng phương thức và sự thay đổi về kiểu giá trị: kiểu dữ liệu của đối số truyền vào được thay đổi sang kiểu dữ liệu khác(tự động ép kiểu) nếu giá trị của đối số đó không phù hợp với kiểu dữ liệu của tham số đã được định nghĩa



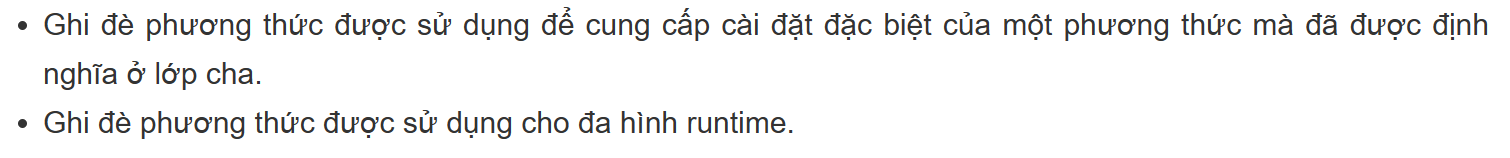
Kiểu byte có thể ép sang kiểu short, int, long,…



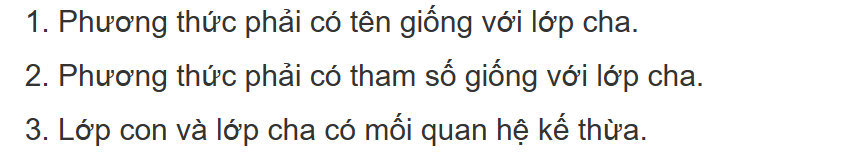


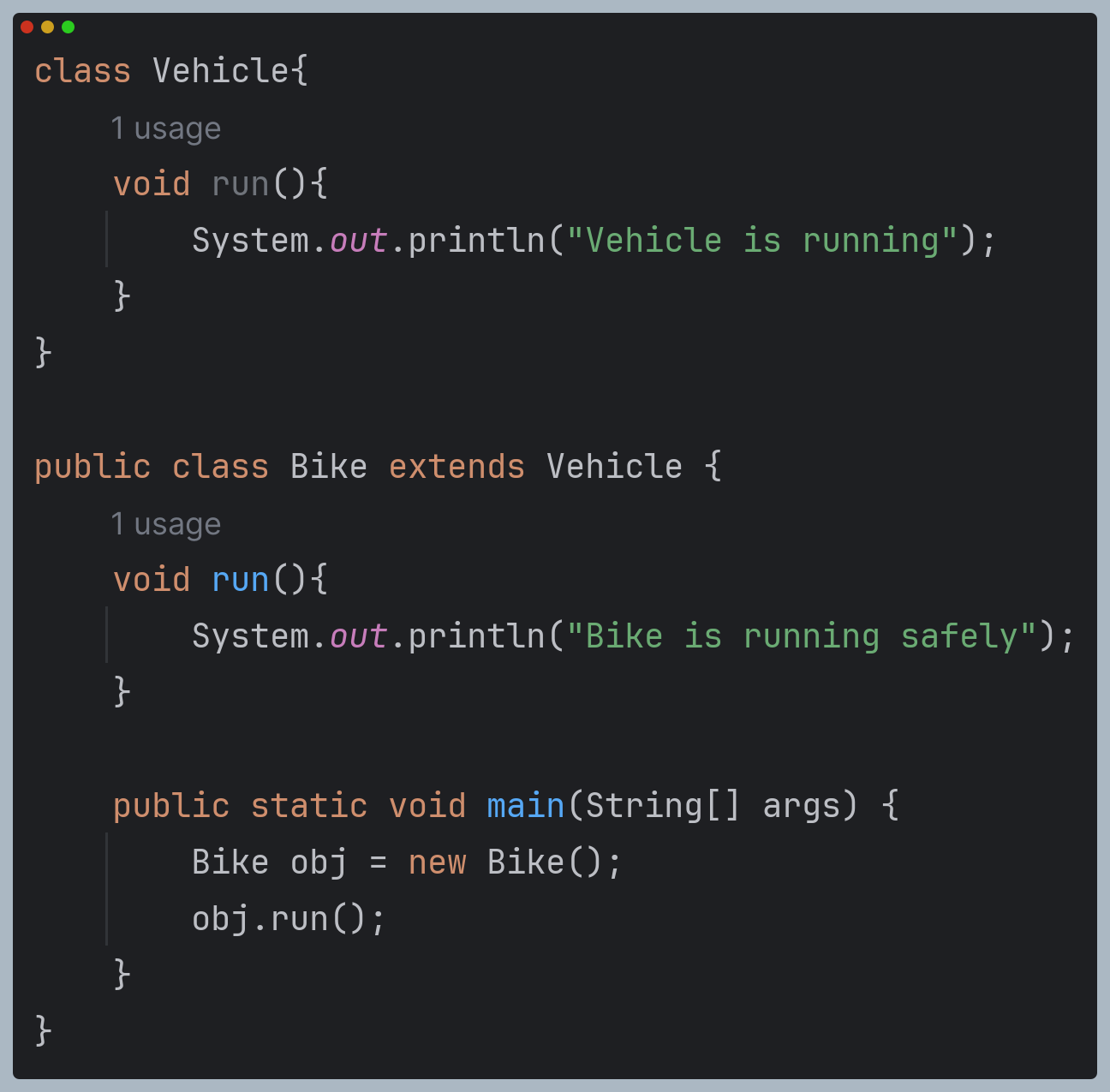
*Ghi đè phương thức:* xảy ra nếu subclass có phương thức giống superclass. Nếu subclass cung cấp sự cài đặt cụ thể cho phương thức đã được cung cấp bởi superclass được gọi là method overriding.

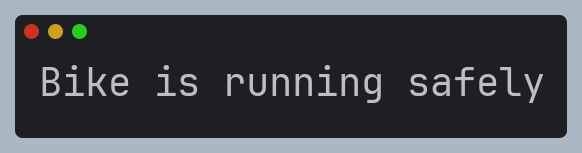
Sự ghi đè phương thức:



Nguyên tắc ghi đè phương thức

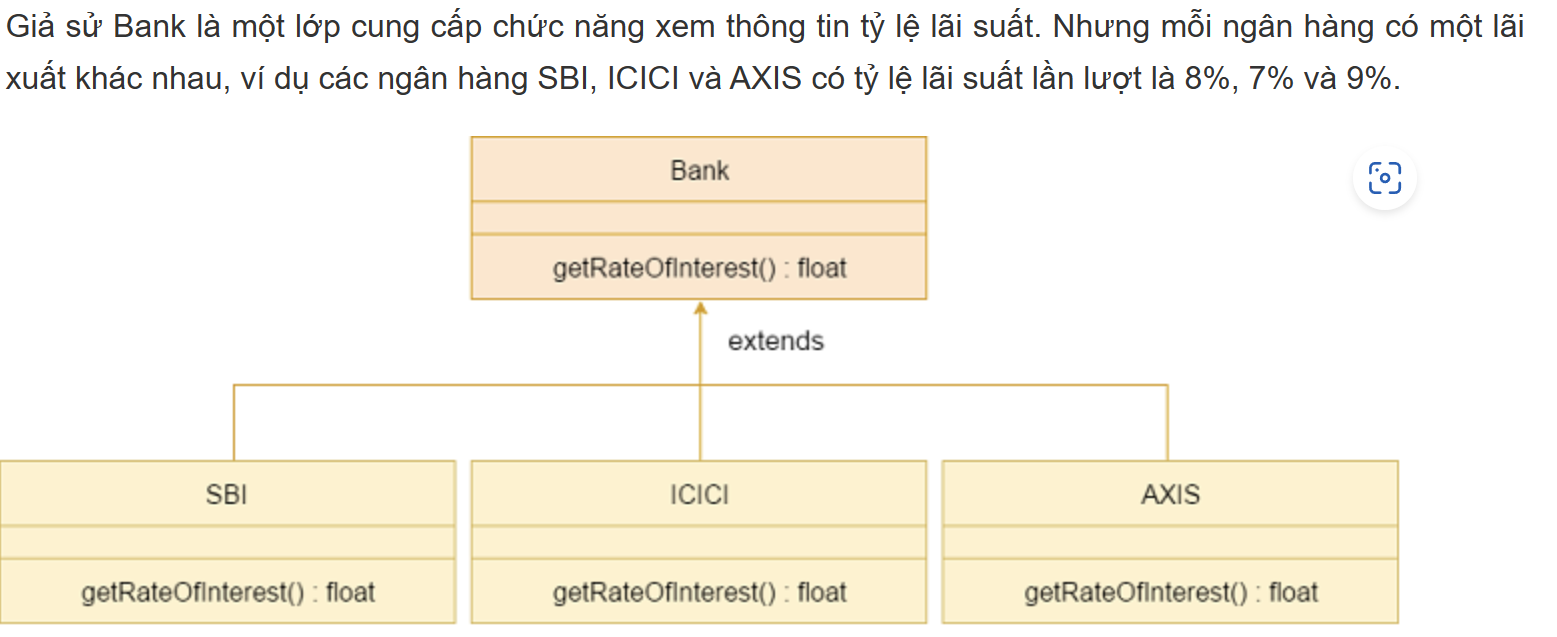






Phương thức run() trong subclass đã được định nghĩa trong superclass

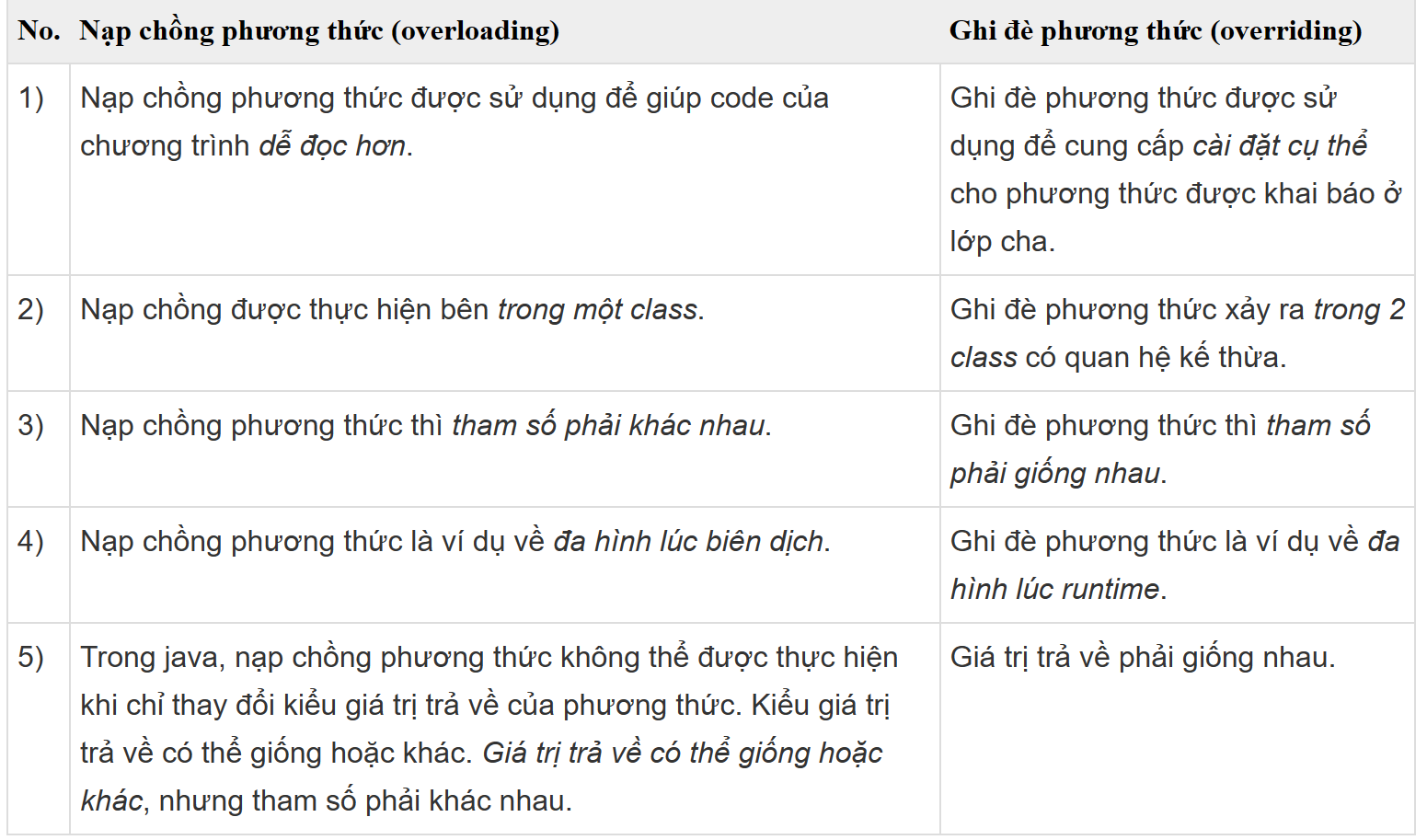
VD thực tế về ghi đè phương thức trong Java







*Overloading vs Overriding*

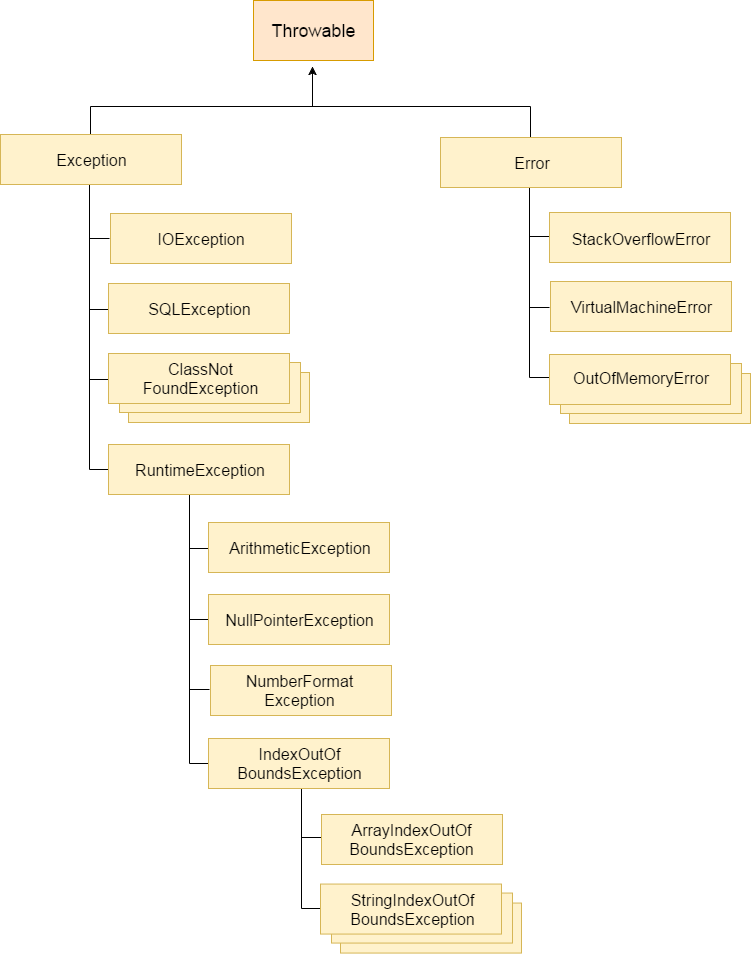


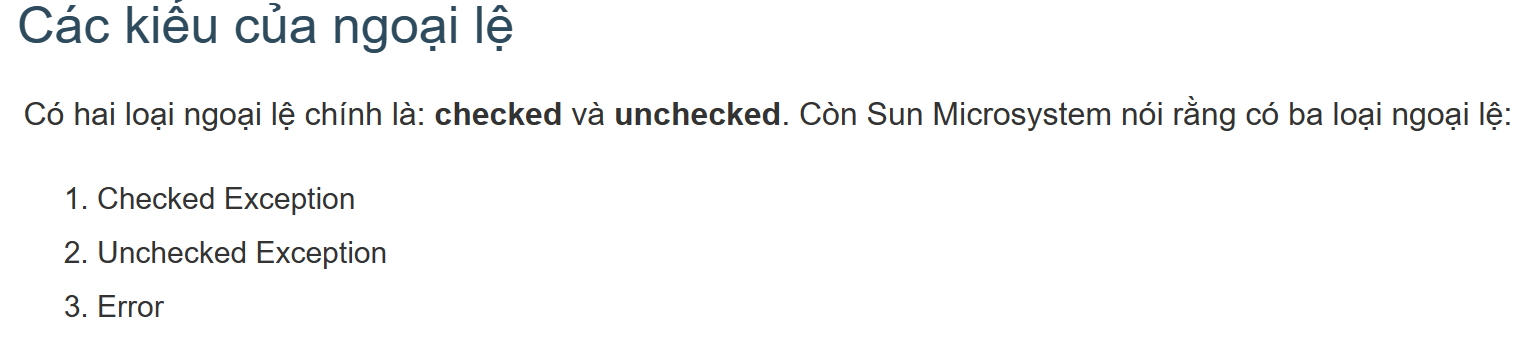
**Xử lí ngoại lệ**

*Exception là gì?* Ngoại lệ là một sự kiện làm gián tiếp luồng bình thường trong chương trình. Là một đối tượng được ném ra tại runtime.

Exception handling: là cơ chế xử lý các lỗi runtime như ClassNotFound, IO, …

Hệ thống cấp bậc các lớp ngoại lệ trong Java



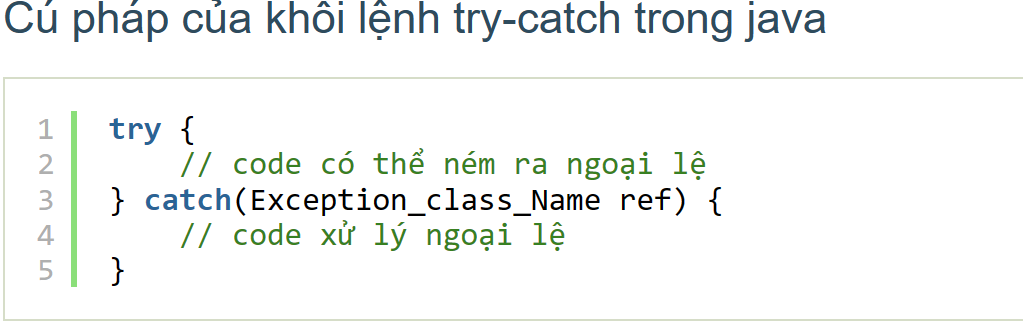


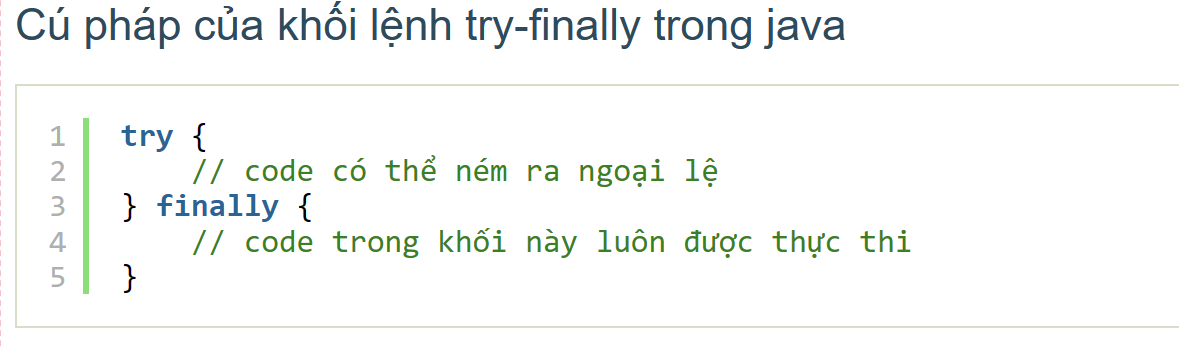
Sự khác nhau giữa Checked Exception và Unchecked Exception

Checked Exception: các extends từ Throwable ngoại trừ RuntimeException và Error là Checked Exception, được kiểm tra tại compile-time

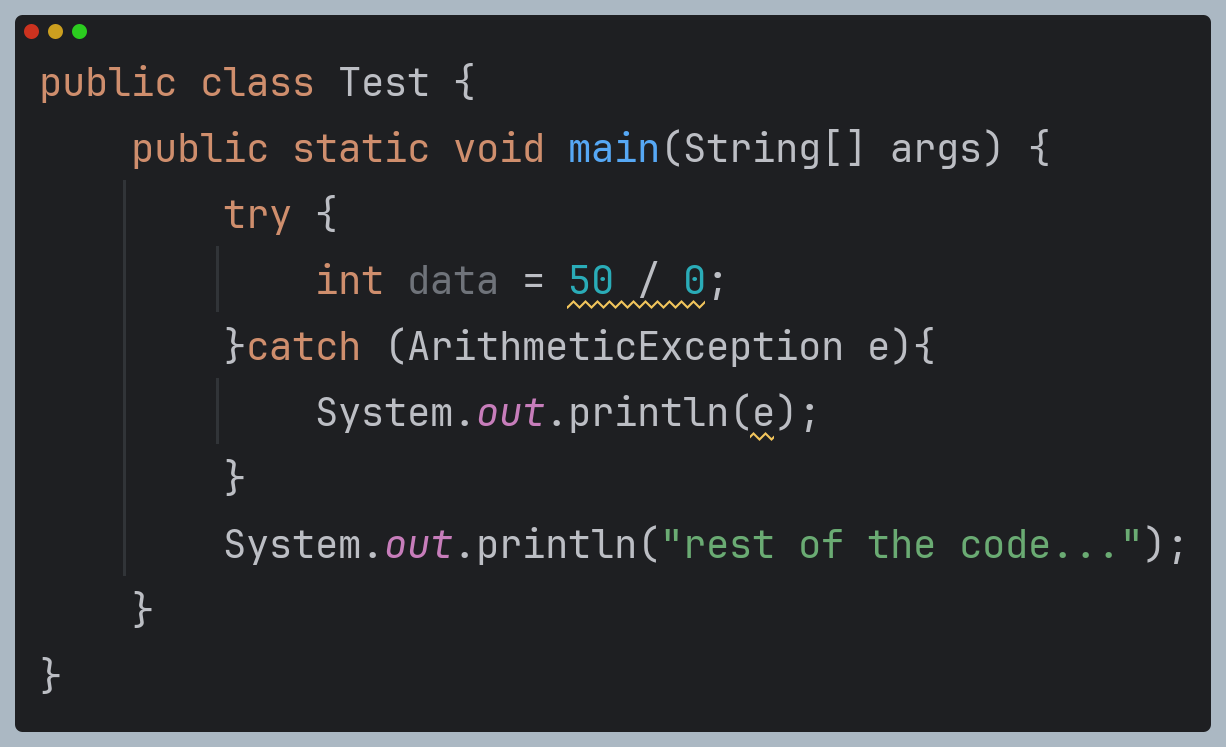
Unchecked Exception: các extends từ RuntimeException, không được kiểm tra tại compile-time mà được kiểm tra tại runtime

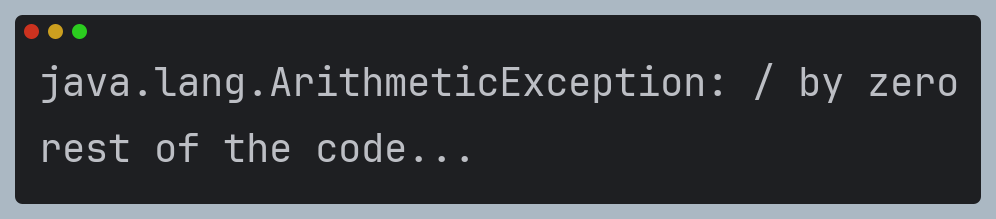
*Khối lệnh Try-Catch*



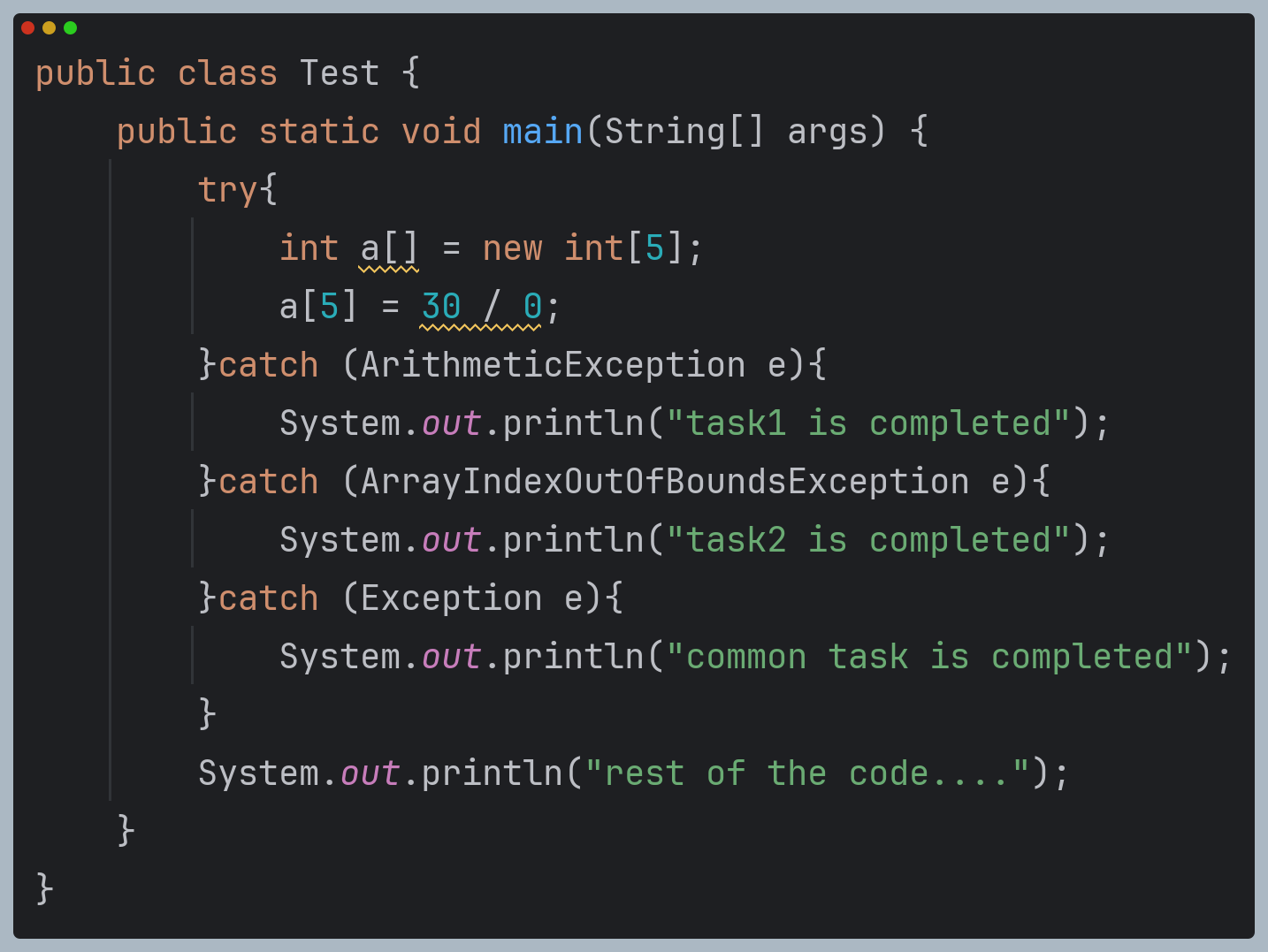


Khối Catch trong Java dùng để xử lý Exception, nó được sử dụng sau khối Try. Có thể sử dụng nhiều khối Catch với một khối Try duy nhất.



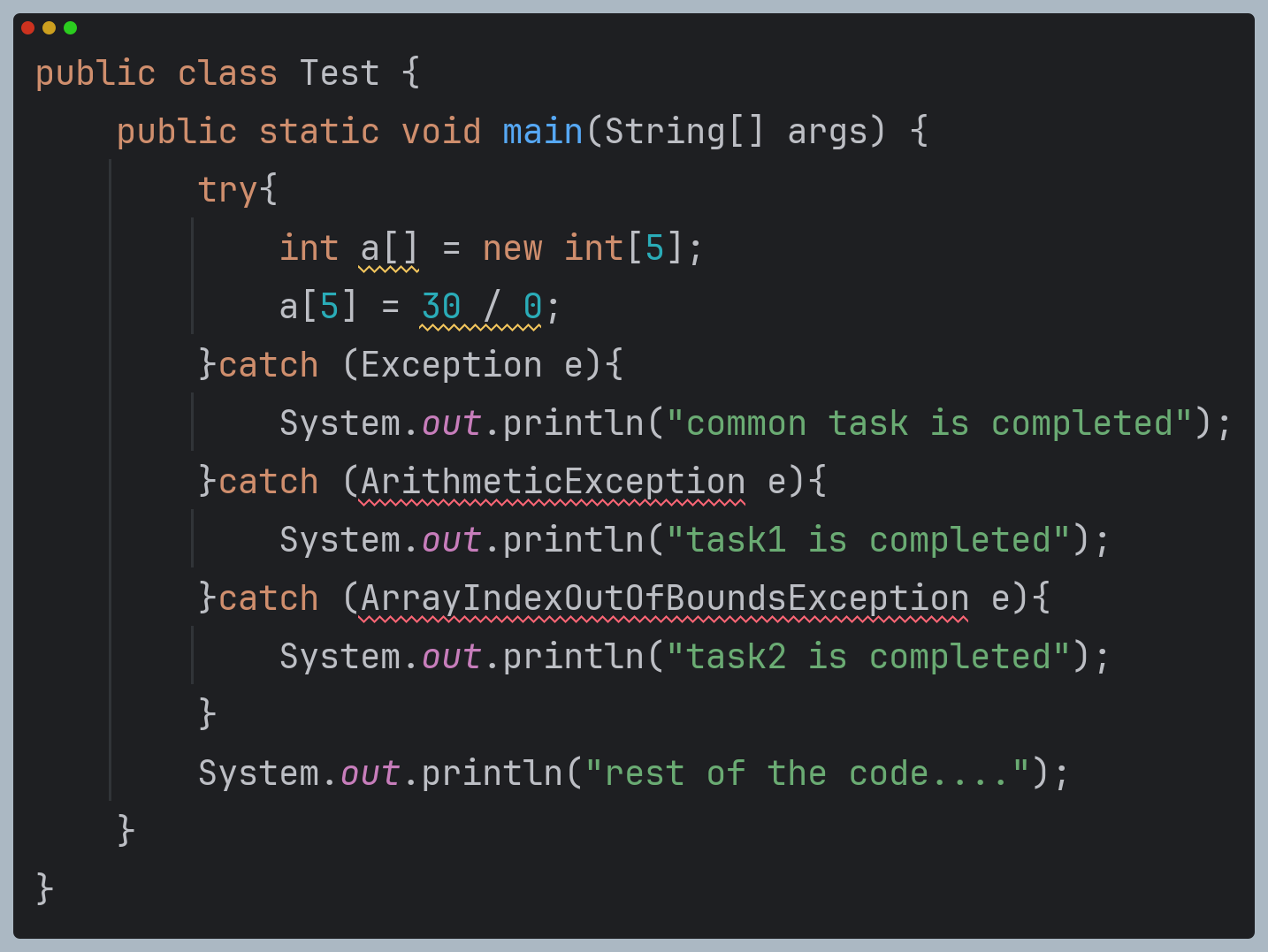


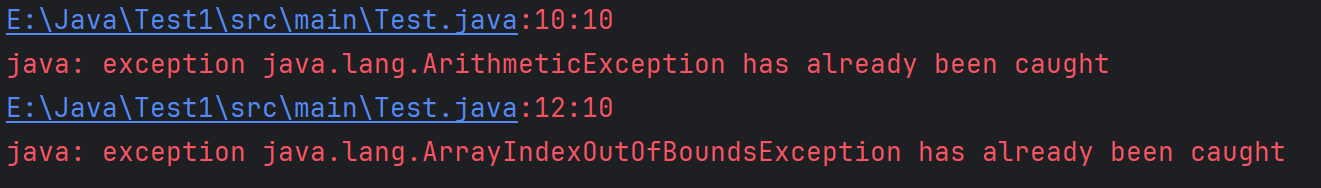
*Đa khối lệnh Catch*





Khi đổi khối Catch Exception lên đầu tiên

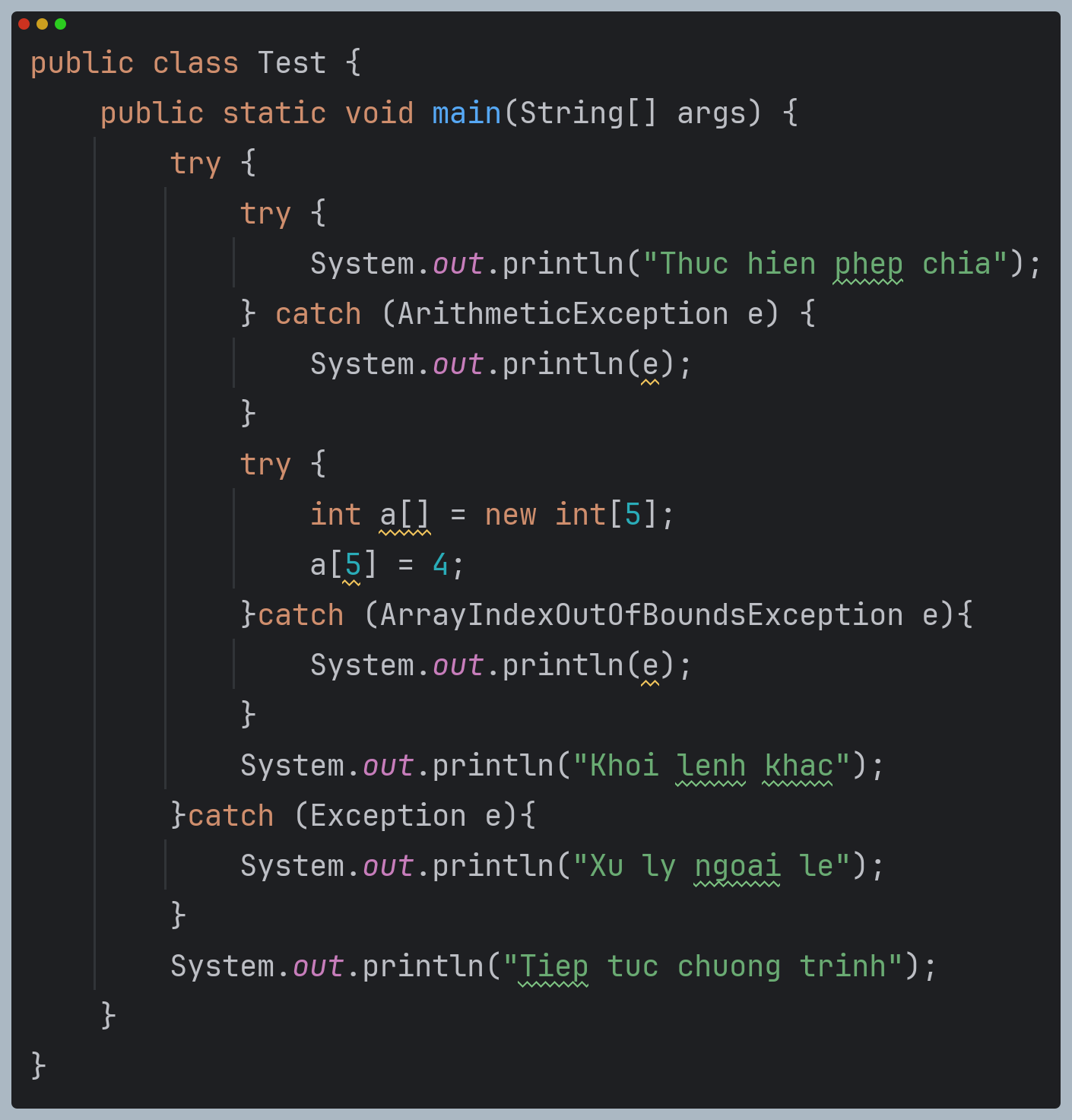




Chương trình sẽ bị lỗi vì ArithmeticException và ArrayIndexOutOfBoundsException không được thực thi vì Exception đã bắt hết lỗi

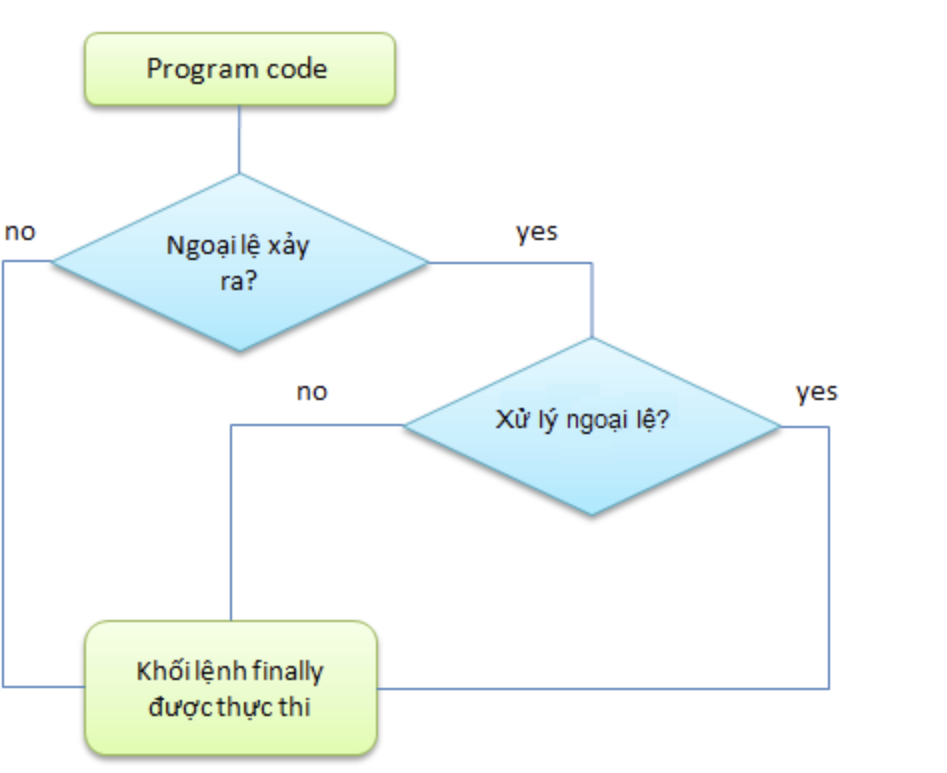
*Khối Try lồng nhau*

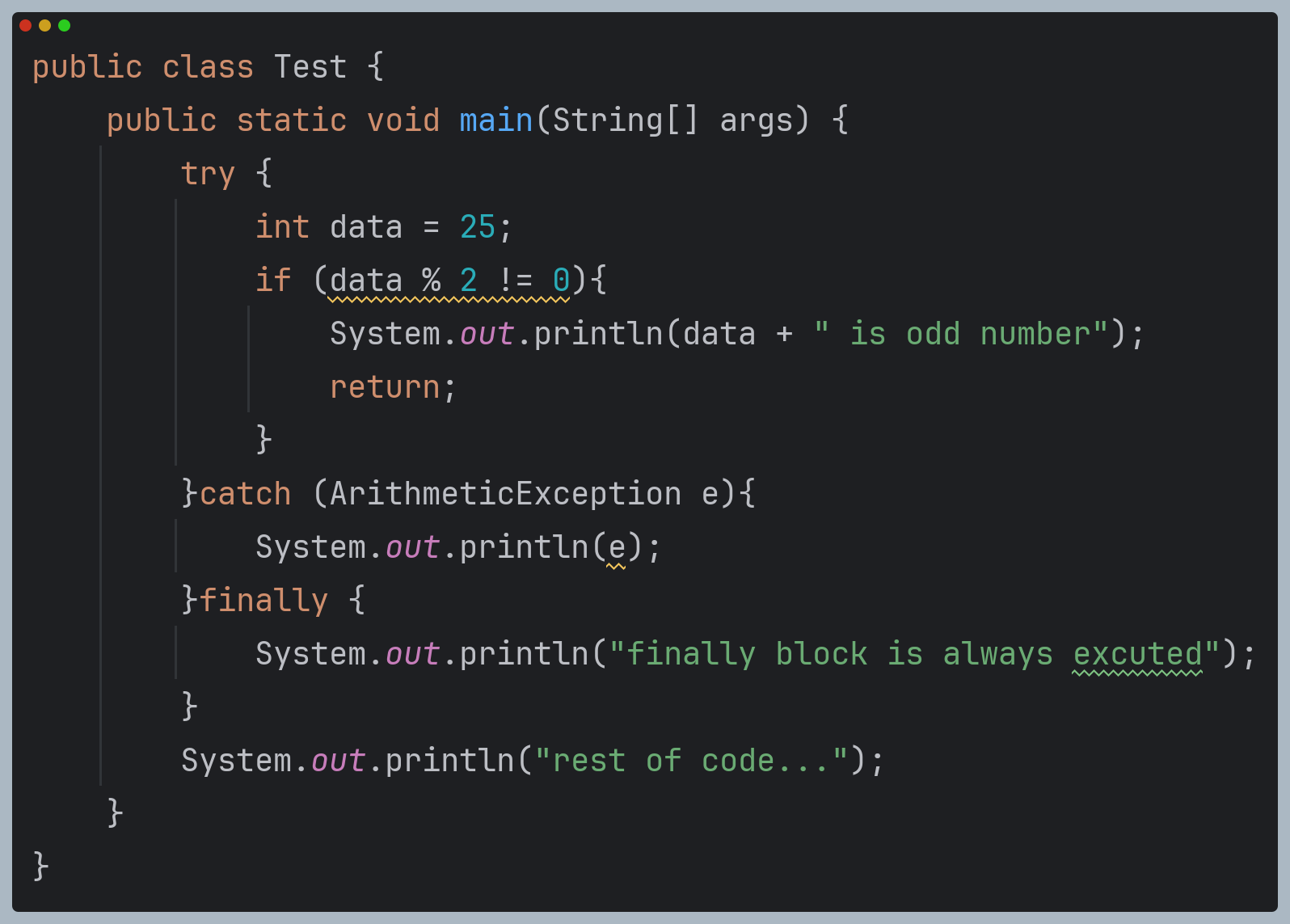
Tại sao? Đôi khi một tình huống có thể phát sinh khi một phần của khối lệnh có thể xảy ra một lỗi và toàn bộ khối lệnh chính có thể xảy ra lỗi khác. Trong những trường hợp đó, xử lý ngoại lệ phải được lồng nhau.

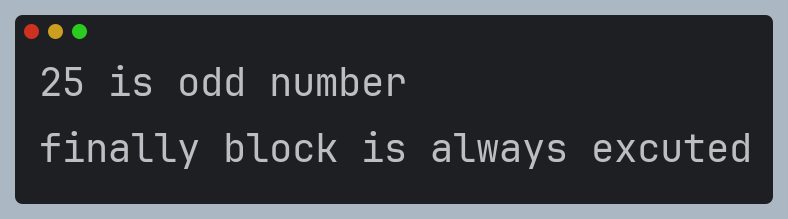




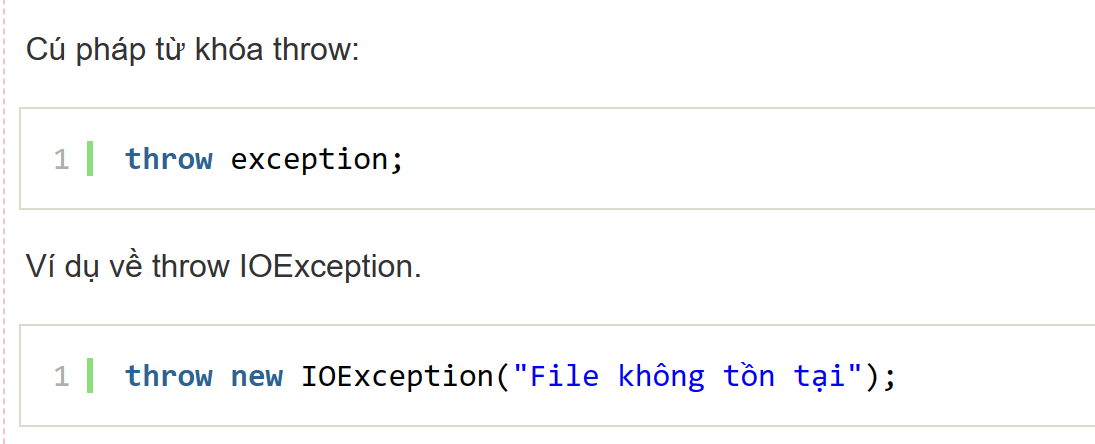
*Khối lệnh finally* được thực thi dù có ngoại lệ xảy ra hay không, hoặc gặp lệnh return trong khối Try







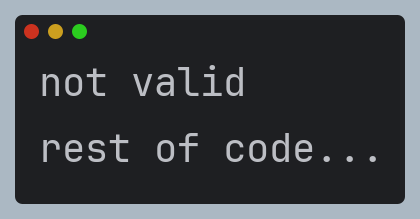
*Từ khóa Throw*  dùng để ném ra một ngoại lệ cụ thể, có thể ném một trong 2 ngoại lệ checked hoặc unchecked bẳng từ khóa Throw. Từ khóa Throw chủ yếu được dùng để ném ngoại lệ tùy chỉnh( ngoại lệ do người dùng định nghĩa)



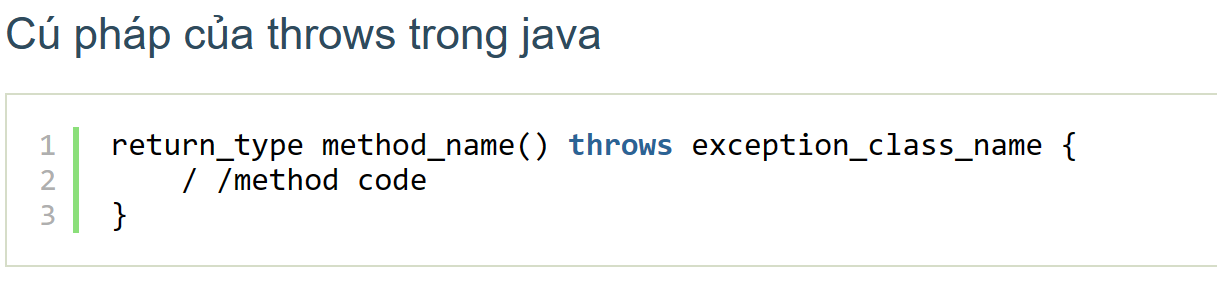


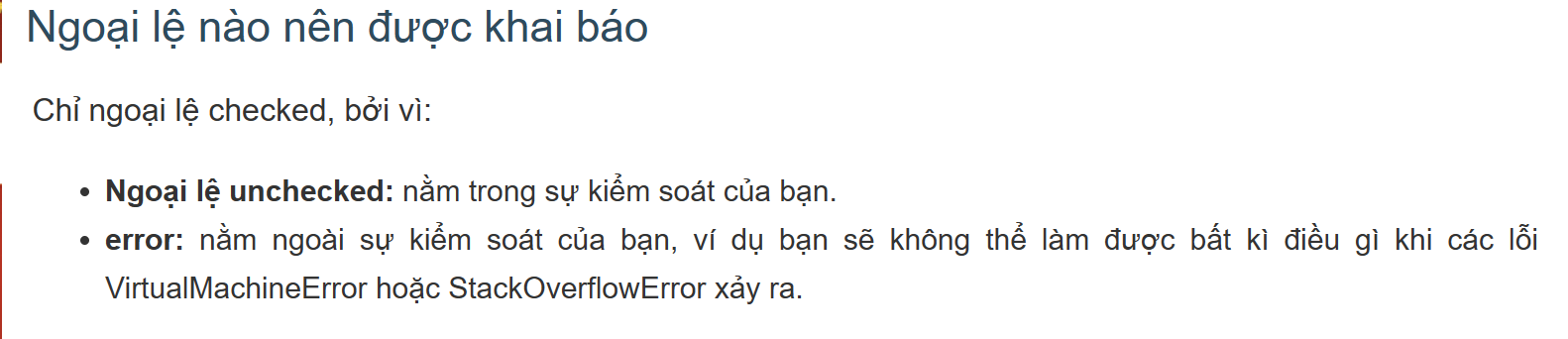




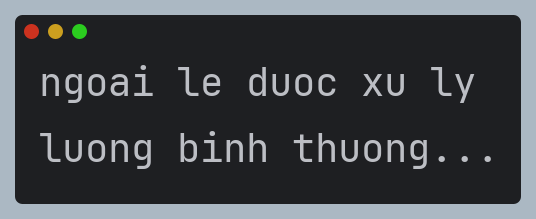


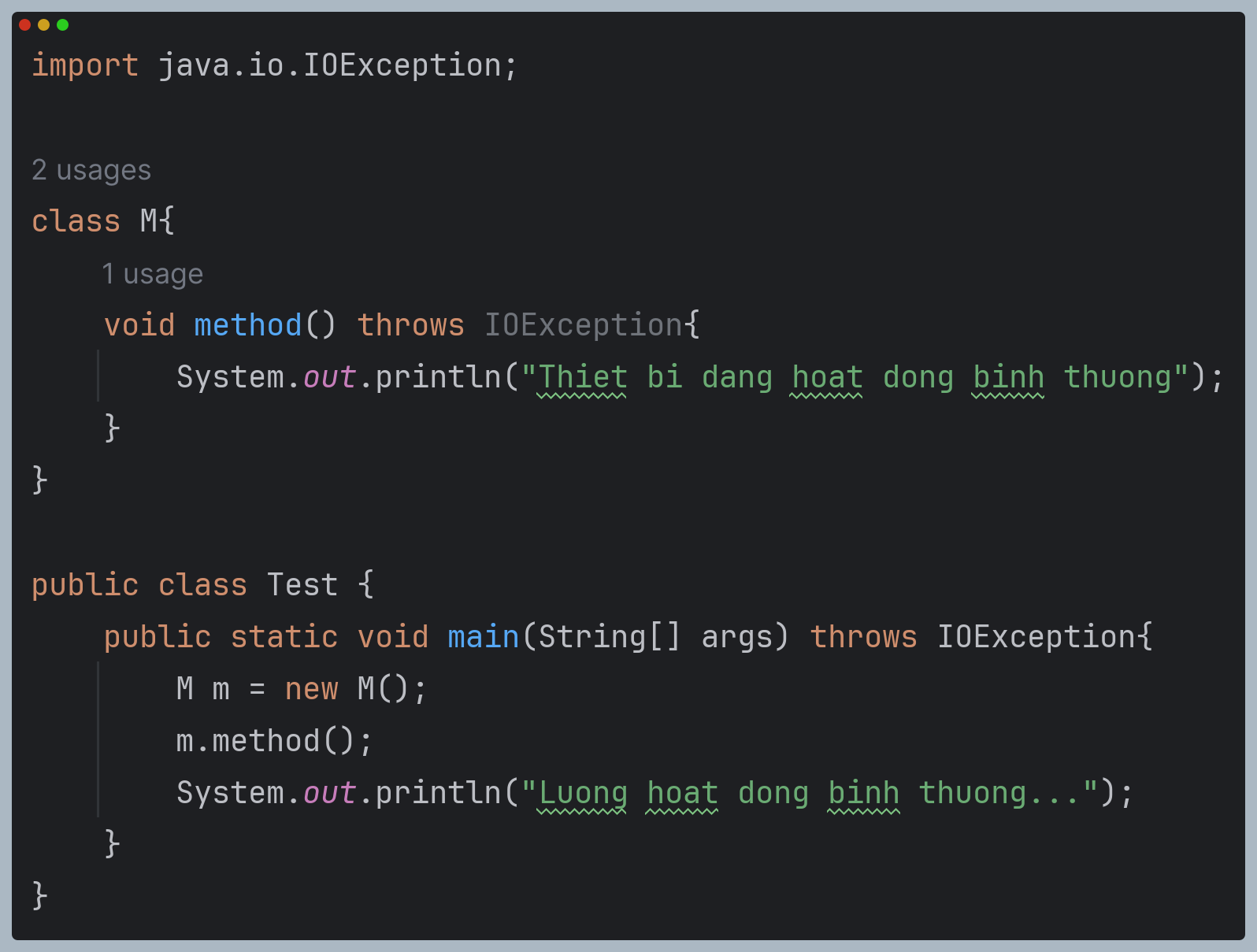
*Từ khóa Throws* sử dụng để khai báo ngoại lệ, hiện thông tin rằng có thể xảy ra một ngoại lệ, vì vậy

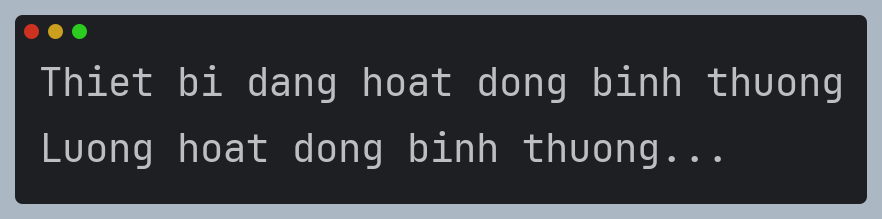




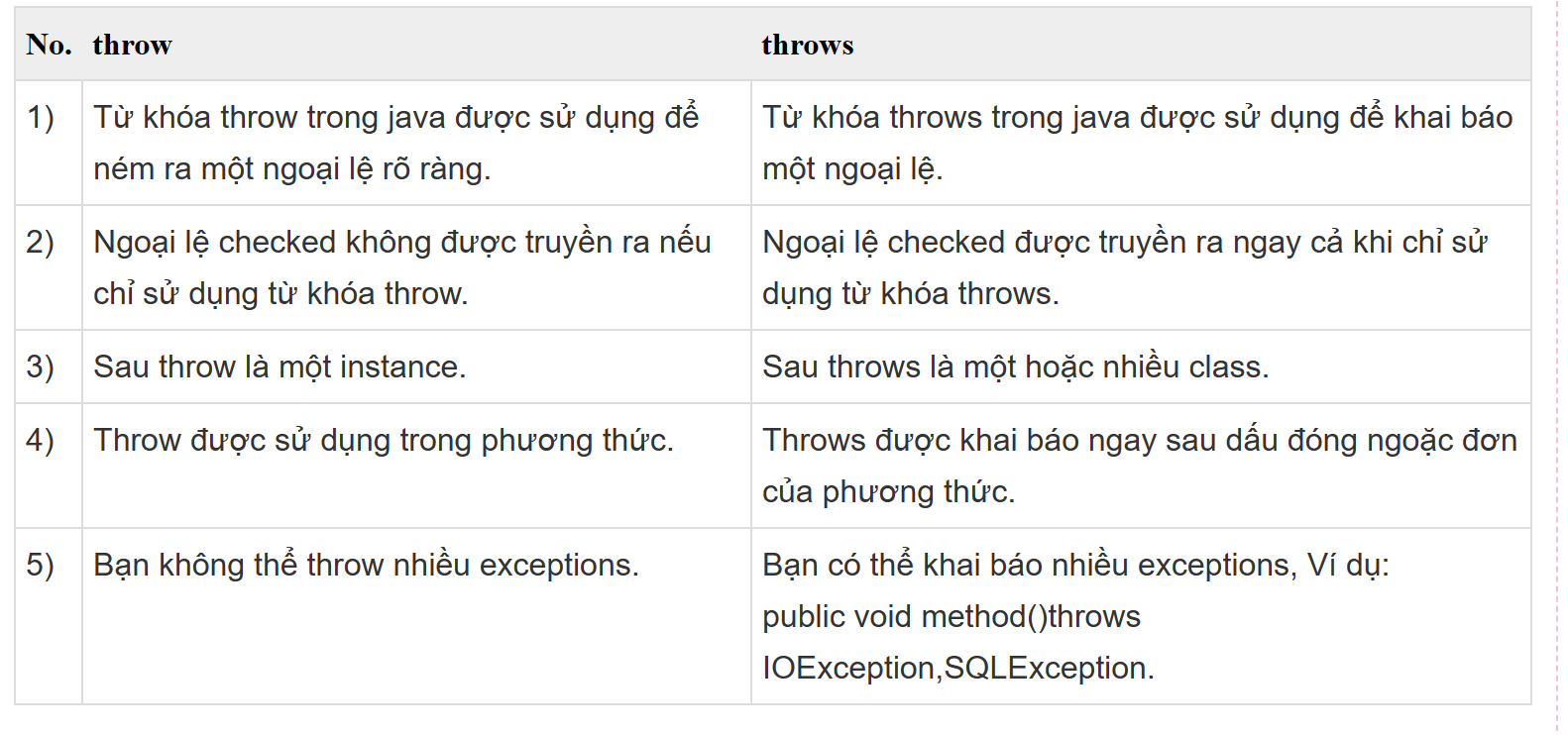








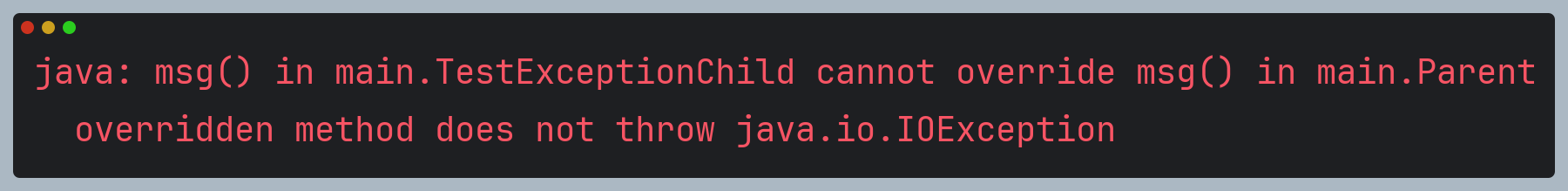
*Sự khác biệt giữa Throw và Throws*



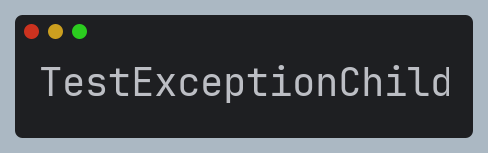
*Exception Handling với Overriding phương thức*

Nếu phương thức của superclass không khai báo ném ra exception: phương thức được ghi đè của superclass không thể khai báo ném ra ngoại lệ checked, nhưng ngoại lệ unchecked thì có thể









Nếu phương thức của superclass khai báo ném ra exception: phương thức được ghi đè của superclass có thể khai báo ném ra ngoại lệ tương tự, ngoại lệ subclass, nhưng không thể khai báo ném ra ngoại lệ superclass

**Interface - Abstract method**