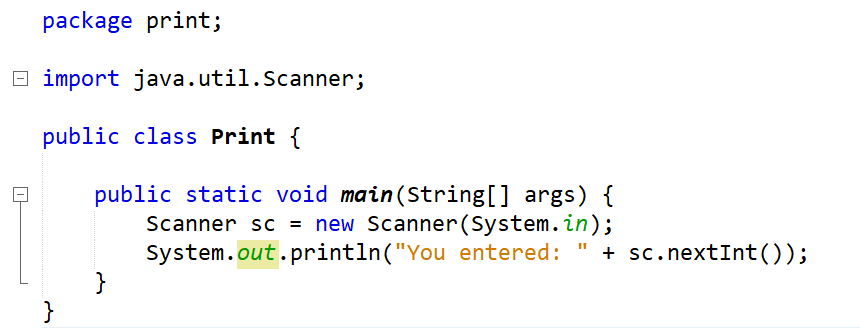
**JAVA - Tuần 1**

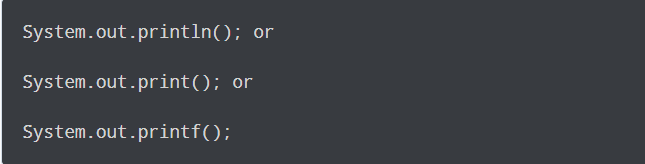
1. **Syntax cơ bản**
2. **Nhập xuất**
3. Nhập

* Sử dụng lớp BufferedReader
* Sử dụng lớp JOptionPane
* Sử dụng lớp Scanner (hay dùng): Nếu dùng BufferReader hoặc JOptionPane thì dữ liệu nhập vào chỉ có kiểu String. Scanner hơn hẳn những lớp trước là vì nó phân loại đc dữ liệu mà người dùng nhập vào cho nó: có thể đọc kiểu int hay float, double,..

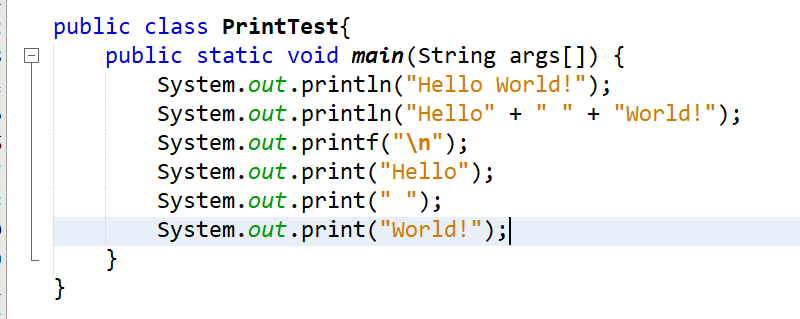


* + next(): Nhận vào một String token (nhận vào 1 từ đầu tiên thay cả câu)
  + nextInt(): Nhận vào một số int
  + nextLong(): Nhận vào một số long
  + nextFloat(): Nhận vào một số float
  + nextDouble(): Nhận vào một số double
  + sc.nextLine(): Nhận vào một chuỗi String (Cả 1 câu)
  + nextByte(): Nhận vào một byte
  + nextBoolean(): Nhận vào một boolean

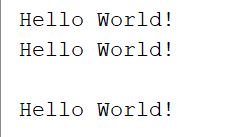
1. Xuất



* System.out.print(*aString*): in *aString* nhưng đặt con trỏ sau chuỗi in
* System.out.println(*aString*): in *aString* và đưa con trỏ đến đầu dòng tiếp theo
* printf(): cung cấp định dạng chuỗi tương tự như printf() trong C/C++
* Ta dùng toán tử **+** để nối 2 chuỗi

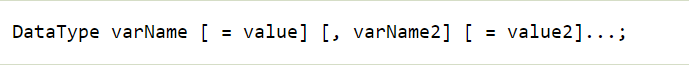


* Output

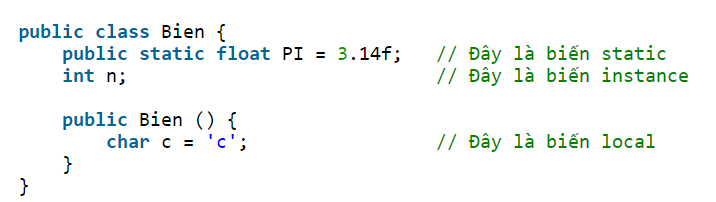


1. **Biến, toán tử**
2. Biến

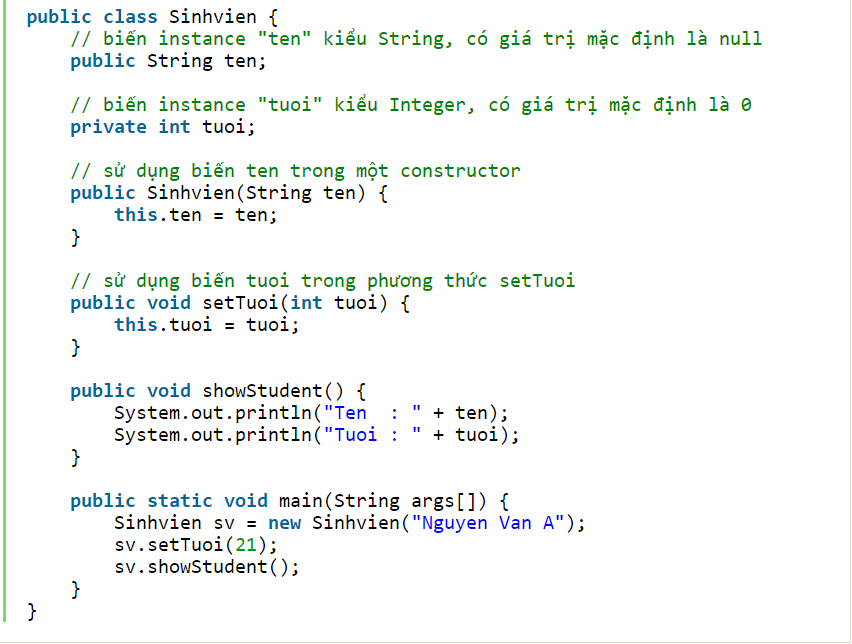
* Syntax:



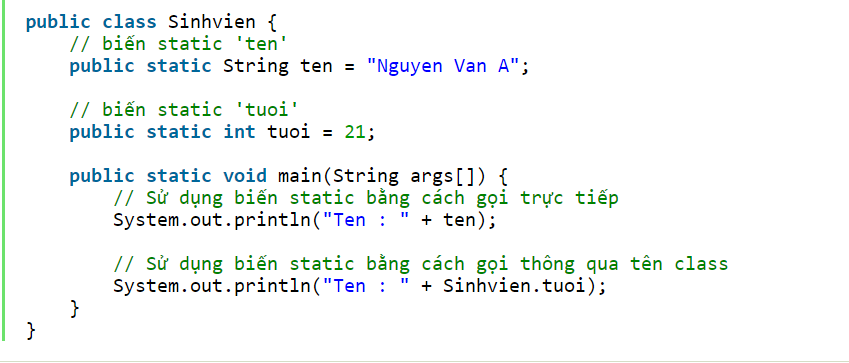
* Quy tắc đặt tên biến:
  + Chỉ được bắt đầu bằng một ký tự(chữ), hoặc một dấu gạch dưới(\_), hoặc một ký tự dollar($)
  + Tên biến không được chứa khoảng trắng
  + Bắt đầu từ ký tự thứ hai, có thể dùng ký tự(chữ), dấu gạch dưới(\_), hoặc ký tự dollar($)
  + Không được trùng với các từ khóa
  + Có phân biệt chữ hoa và chữ thường
* Có 3 kiểu biến trong java



* Biến local
  + Biến local được khai báo trong các phương thức, constructor hoặc trong các block.
  + Biến local được tạo bên trong các phương thức, constructor, block và sẽ bị phá hủy khi kết thúc các phương thức, constructor và block.
  + Không được sử dụng "access modifier" khi khai báo biến local.
  + Các biến local được lưu trên vùng nhớ stack của bộ nhớ.
  + Bạn cần khởi tạo giá trị mặc định cho biến local trước khi có thể sử dụng.
* Biến instance
  + Biến instance được khai báo trong một lớp(class), bên ngoài các phương thức, constructor và các block.
  + Biến instance được lưu trong bộ nhớ heap.
  + Biến instance được tạo khi một đối tượng được tạo bằng việc sử dụng từ khóa “new” và sẽ bị phá hủy khi đối tượng bị phá hủy.
  + Biến instance có thể được sử dụng bởi các phương thức, constructor, block, ... Nhưng nó phải được sử dụng thông qua một đối tượng cụ thể.
  + Bạn được phép sử dụng "access modifier" khi khai báo biến instance, mặc định là "default".
  + Biến instance có giá trị mặc định phụ thuộc vào kiểu dữ liệu của nó. Ví dụ nếu là kiểu int, short, byte thì giá trị mặc định là 0, kiểu double thì là 0.0d, ... Vì vậy, bạn sẽ không cần khởi tạo giá trị cho biến instance trước khi sử dụng.
  + Bên trong class mà bạn khai báo biến instance, bạn có thể gọi nó trực tiếp bằng tên khi sử dụng ở khắp nơi bên trong class đó.

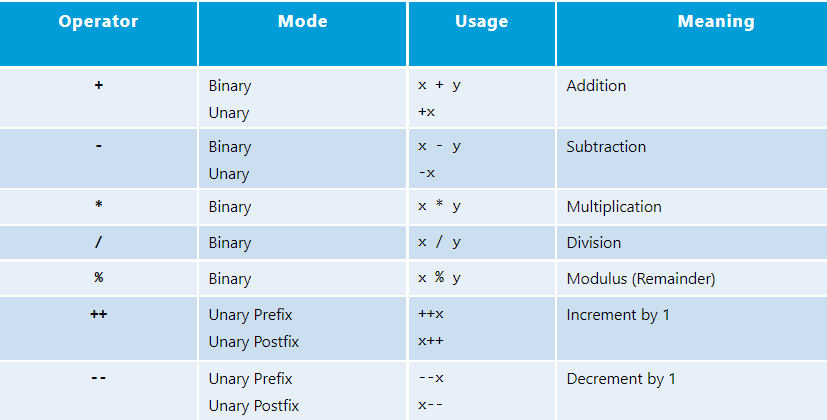


* Biến static
  + Biến static được khai báo trong một class với từ khóa "static", phía bên ngoài các phương thức, constructor và block.
  + Sẽ chỉ có duy nhất một bản sao của các biến static được tạo ra, dù bạn tạo bao nhiêu đối tượng từ lớp tương ứng.
  + Biến static được lưu trữ trong bộ nhớ static riêng.
  + Biến static được tạo khi chương trình bắt đầu chạy và chỉ bị phá hủy khi chương trình dừng.
  + Giá trị mặc định của biến static phụ thuộc vào kiểu dữ liệu bạn khai báo tương tự biến instance.
  + Biến static được truy cập thông qua tên của class chứa nó, với cú pháp: TenClass.tenBien
  + Trong class, các phương thức sử dụng biến static bằng cách gọi tên của nó khi phương thức đó cũng được khai báo với từ khóa "static".



1. Toán tử

* Toán tử số học



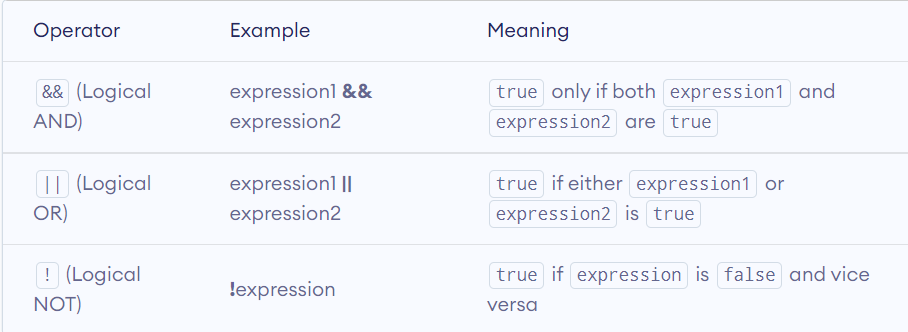
* Toán tử gán



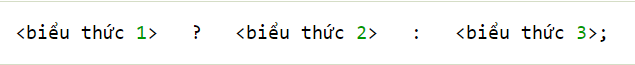
* Toán tử quan hệ



* Toán tử logic



* Toán tử 3 ngôi

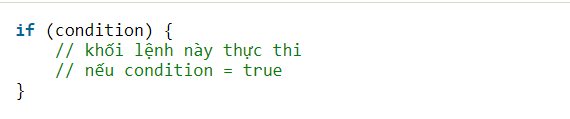


* Toán tử bit

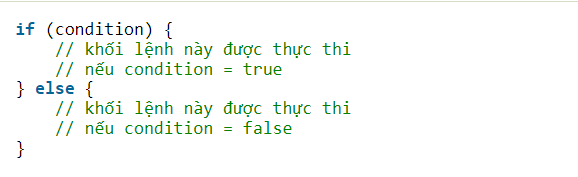


1. **If else**

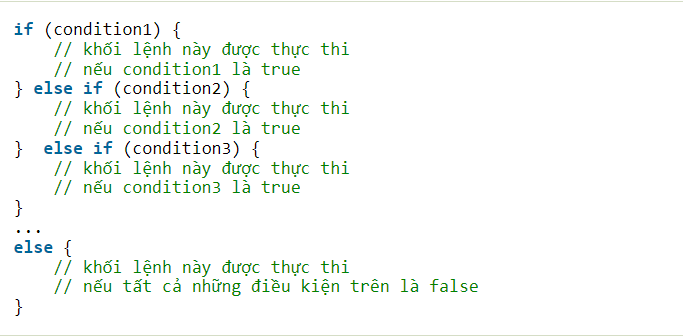
* if (if-then)



* if-else

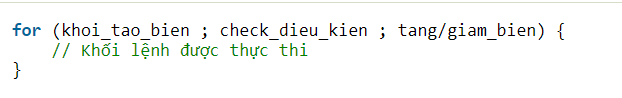


* if…else…if

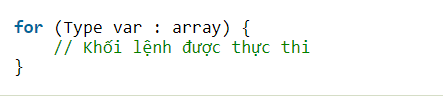


1. **Vòng lặp**
2. for

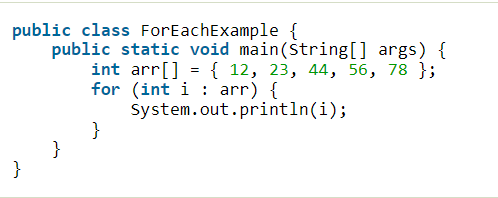
* Vòng lặp for đơn giản



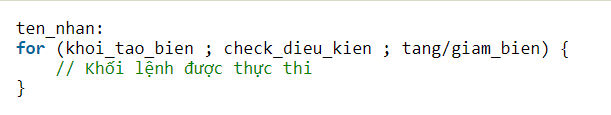
* Vòng lặp for cải tiến (for-each): được sử dụng để lặp mảng(array) hoặc collection trong java. Bạn có thể sử dụng nó dễ dàng, dễ hơn cả vòng lặp for đơn giản. Bởi vì bạn không cần phải tăng hay giảm giá trị của biến rồi check điều kiện, bạn chỉ cần sử dụng ký hiệu hai chấm ":"
  + Syntax



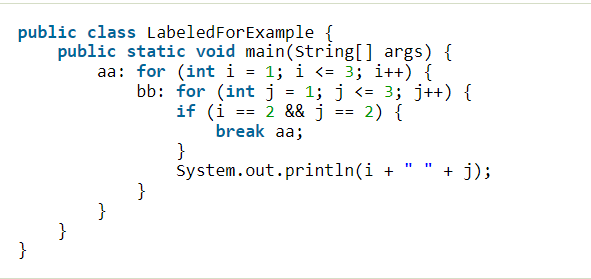
* + Ví dụ



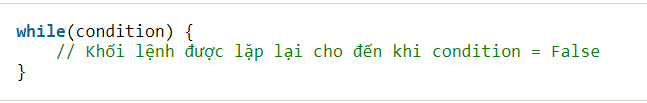
* Vòng lặp for dán nhãn: Chúng ta có để đặt tên cho mỗi vòng lặp for bằng cách gán nhãn trước vòng lặp for. Điều này rất hữu dụng khi chúng ta muốn thoát/tiếp tục(break/continues) chạy vòng lặp for.
  + Syntax



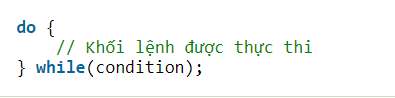
* + Ví dụ



1. while

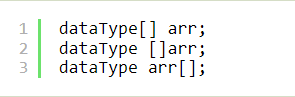


1. do-while

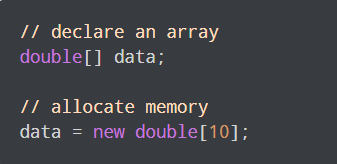


1. break: kết thúc vòng lặp ngay lập tức và chuyển sang câu lệnh tiếp theo sau vòng lặp
2. continue: Từ khóa continue trong java được sử dụng để tiếp tục vòng lặp tại điều kiện đã được xác định, với điều kiện đó khối lệnh phía sau từ khóa continue sẽ không được thực thi. Đối với vòng lặp bên trong một vòng lặp khác, continue chỉ có tác dụng với vòng lặp bên trong đó.
3. **Array**
4. Mảng 1 chiều

* Khai báo

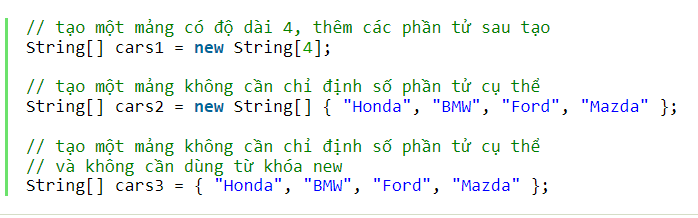


* Cấp phát bộ nhớ

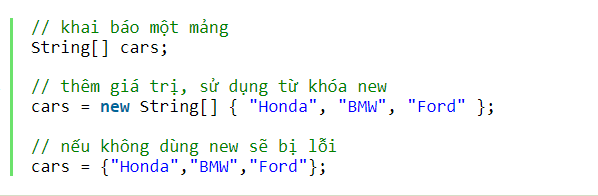


* Khởi tạo mảng

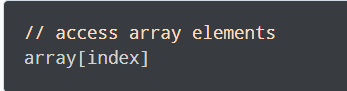




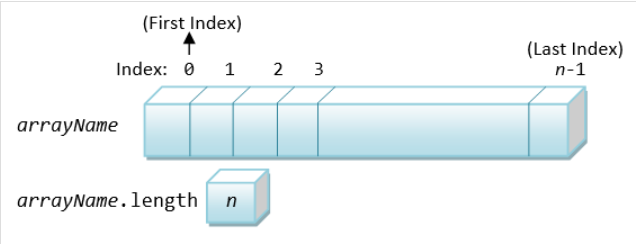
* + Nếu khai báo một mảng và khởi tạo nó sau, ban phải sử dụng từ khóa new



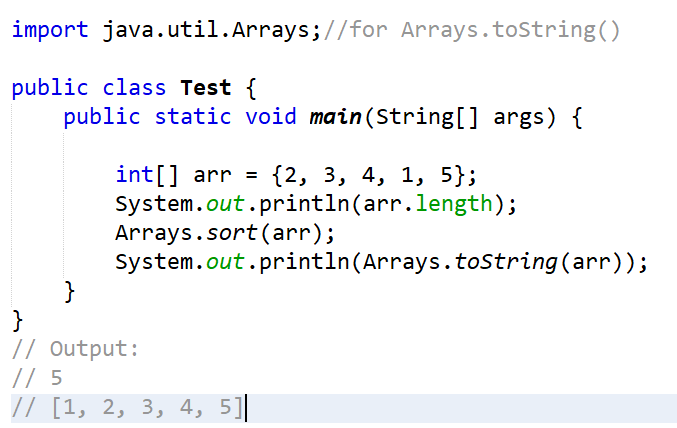
* Truy cập



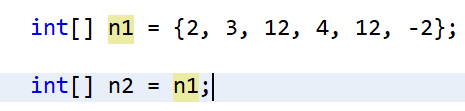
* Độ dài của mảng: array.length



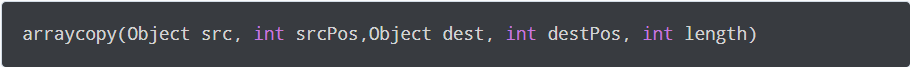
* Arrays.toString()
  + Dùng Arrays.toString() để in mảng dưới dạng [a1, a2, ..., an]
  + Cần import java.util.Arrays để dùng Arrays.toString()
* Arrays.sort(int/float/long… a[])
  + Cần import java.util.Arrays để dùng Arrays.sort(int a[])



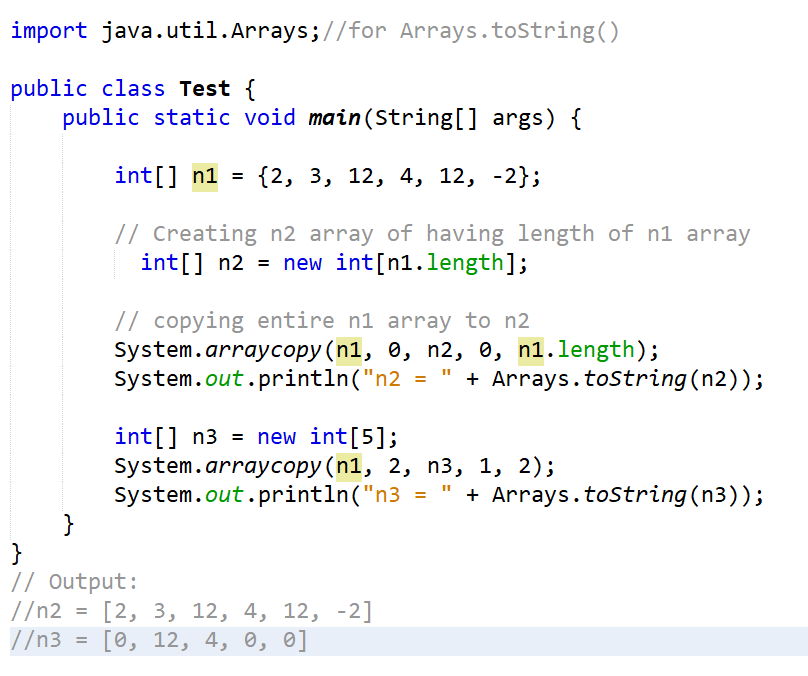
* Copy
  + Sử dụng toán tử gán



* + Dùng arraycopy()

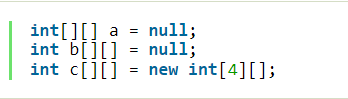


* src: mảng nguồn bạn muốn copy
* srcPos: vị trí bắt đầu (chỉ mục) trong mảng nguồn
* dest: mảng đích nơi các phần tử sẽ được sao chép từ mảng nguồn
* destPos: vị trí bắt đầu (chỉ mục) trong mảng đích
* length: số phần tử muốn copy

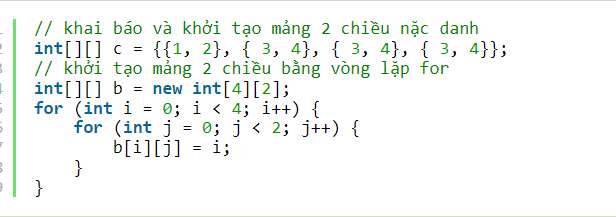


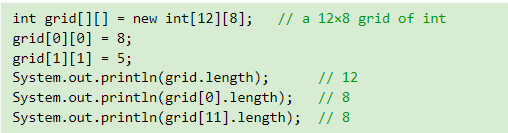
1. Mảng 2 chiều

* Khai báo mảng 2 chiều:

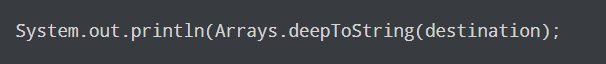


* Khởi tạo mảng 2 chiều

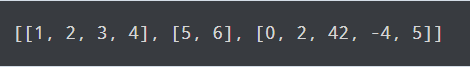




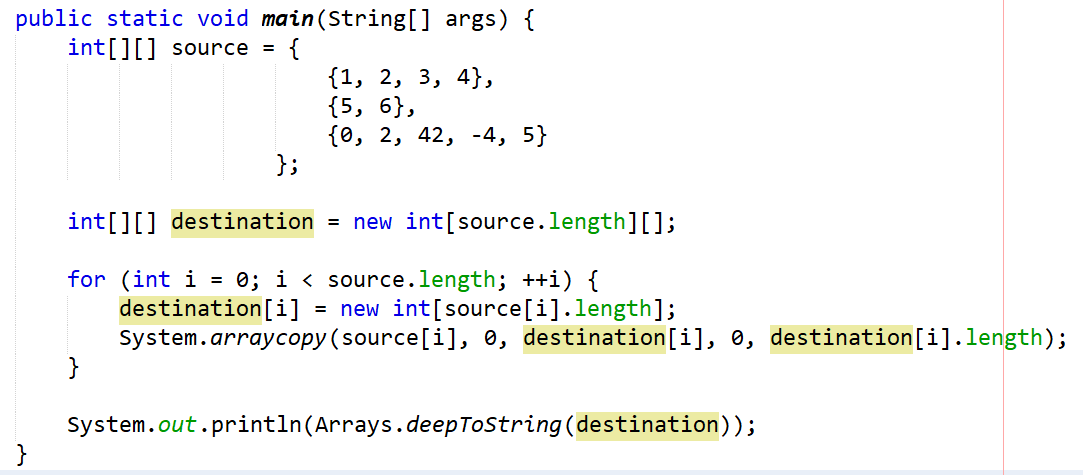
* Arrays.deepToString()



* + Ví dụ output:



* Copy
  + Dùng vòng lặp
  + Dùng arraycopy()

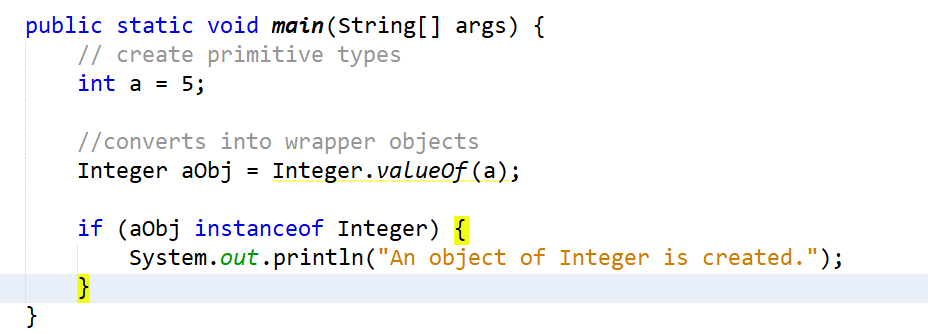


1. **Wrapper class**

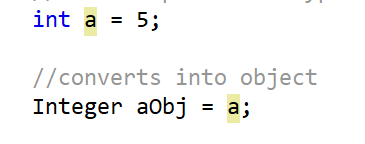
* Lớp Wrapper trong java cung cấp cơ chế để chuyển đổi kiểu dữ liệu nguyên thủy thành kiểu đối tượng và từ đối tượng thành kiểu dữ liệu nguyên thủy.
* Từ J2SE 5.0,
  + Tính năng autoboxing chuyển đổi tự động kiểu dữ liệu nguyên thủy thành kiểu đối tượng
  + Tính năng unboxing chuyển đổi tự động đối tượng thành kiểu dữ liệu nguyên thuỷ

## Chuyển đổi kiểu nguyên thủy thành đối tượng

* Chúng ta cũng có thể sử dụng phương thức valueOf() để chuyển đổi các kiểu nguyên thủy thành các đối tượng tương ứng.

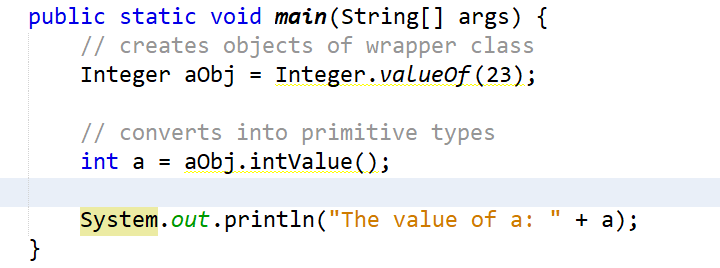


* + Ở đây, chúng ta đã sử dụng toán tử instanceof để kiểm tra xem các đối tượng được tạo có thuộc kiểu Integer hay không
* Tuy nhiên, trình biên dịch Java có thể chuyển đổi trực tiếp các kiểu nguyên thủy thành các đối tượng tương ứng. Ví dụ:

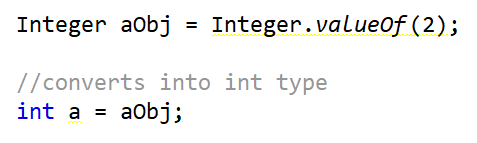


## Chuyển đổi các đối tượng thành các loại nguyên thủy

* Để chuyển đổi các đối tượng thành các kiểu nguyên thủy, chúng ta có thể sử dụng các phương thức giá trị tương ứng (intValue (), doubleValue (), v.v.) có trong mỗi lớp wrapper.

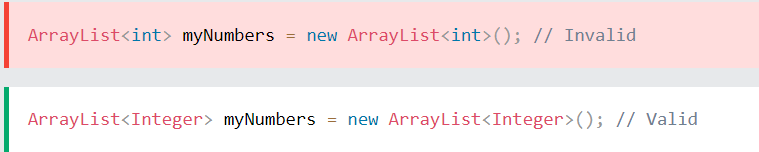


Tuy nhiên, trình biên dịch Java có thể tự động chuyển đổi các đối tượng thành các kiểu nguyên thủy tương ứng. Ví dụ:

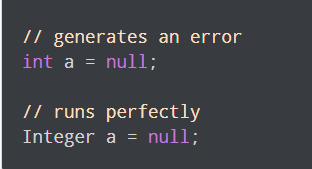


1. Lợi ích của Wrapper Classes

* Trong Java, đôi khi chúng ta có thể cần sử dụng các đối tượng thay vì các kiểu dữ liệu nguyên thủy. Ví dụ, trong khi làm việc với các collection.



* Chúng ta có thể lưu trữ giá trị null trong các đối tượng wrapper. Ví dụ:



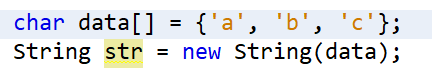
* Các kiểu nguyên thủy hiệu quả hơn các đối tượng tương ứng. Do đó chúng ta được khuyến khích sử dụng các kiểu nguyên thủy.

1. **String classes**

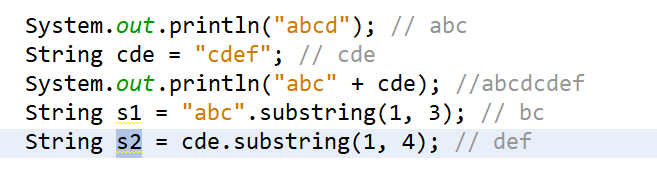
* Lớp String đại diện cho các chuỗi ký tự. Tất cả các ký tự chuỗi trong chương trình Java, chẳng hạn như "abc", được triển khai dưới dạng các thể hiện (instance) của lớp này.
* Chuỗi là bất biến
  + Chuỗi là hằng số; giá trị của chúng không thể thay đổi sau khi chúng được tạo
  + StringBuffer được sử dụng để tạo chuỗi có thể thay đổi (mutable)
  + Vì các đối tượng String là bất biến nên chúng có thể được chia sẻ. Ví dụ:



tương đương với



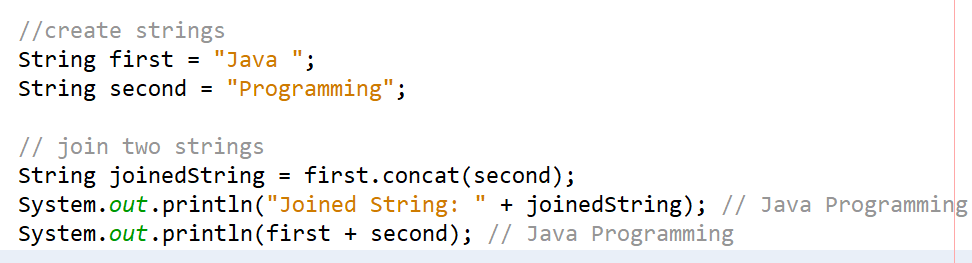
* Một số ví dụ về cách các chuỗi có thể được sử dụng



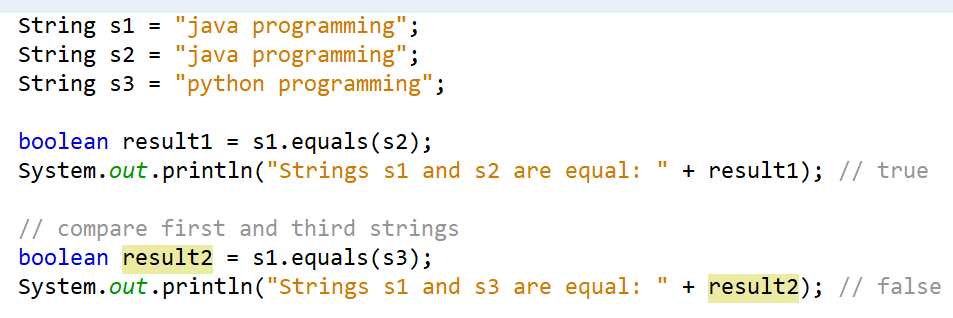
* Lớp String bao gồm các phương thức để kiểm tra các ký tự riêng lẻ của chuỗi, để so sánh các chuỗi, để tìm kiếm các chuỗi, để trích xuất các chuỗi con và để tạo bản sao của một chuỗi với tất cả các ký tự được dịch sang chữ hoa hoặc chữ thường.
  + length(): Lấy độ dài của một String



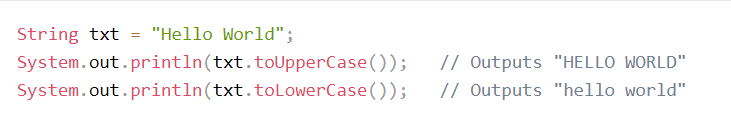
* + concat() hoặc + : Nối 2 String



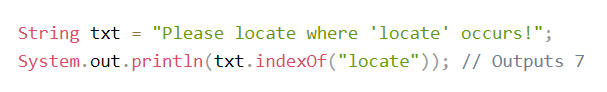
* + equals() : So sánh 2 String



* + toUpperCase() và toLowerCase()



* + indexOf(): Trả về index (vị trí) của lần xuất hiện đầu tiên của một text cụ thể được chỉ định trong chuỗi (bao gồm cả khoảng trắng)



1. Java Math Class

Lớp Math có nhiều phương thức cho phép bạn thực hiện các phép tính toán học trên các con số. Ví dụ:

* Math.max(x,y)
* Math.min(x,y)
* Math.sqrt(x)
* Math.abs(x)
* Math.random(): trả về 1 số ngẫu nhiên giữa 0.0 và 1.0

**II. RegEx (Regular Expression)**

1. **Khái niệm**

* Biểu thức chính quy là một chuỗi các ký tự tạo thành một mẫu tìm kiếm. Khi bạn tìm kiếm dữ liệu trong một văn bản, bạn có thể sử dụng mẫu tìm kiếm này để mô tả những gì bạn đang tìm kiếm.
* Biểu thức chính quy có thể là một ký tự đơn hoặc một mẫu phức tạp hơn.
* Biểu thức chính quy có thể được sử dụng để thực hiện tất cả các loại hoạt động tìm kiếm văn bản và thay thế văn bản.
* Java không tích hợp sẵn lớp RegEx, nhưng chúng ta có thể import gói java.util.regex để làm việc với biểu thức chính quy. Gói này bao gồm các lớp sau:
  + Pattern Class: Xác định một mẫu (được sử dụng trong tìm kiếm)
  + Matcher Class: Được sử dụng để tìm kiếm mẫu
  + PatternSyntaxException Class: Chỉ ra lỗi cú pháp trong một mẫu biểu thức chính quy

1. **Ví dụ**

****

* Pattern.compile(): phương thức dùng để tạo mẫu
  + “w3schools”: mẫu được tìm
  + Pattern.CASE\_INSENSITIVE: tìm kiếm không phân biệt chữ hoa chữ thường
* matcher(): phương thức dùng để tìm mẫu trong chuỗi. Nó trả về 1 đối tượng Matcher chứa thông tin về tìm kiếm đã được thực hiện
* find(): phương thức trả về true nếu mẫu được tìm thấy trong chuỗi và ngược lại

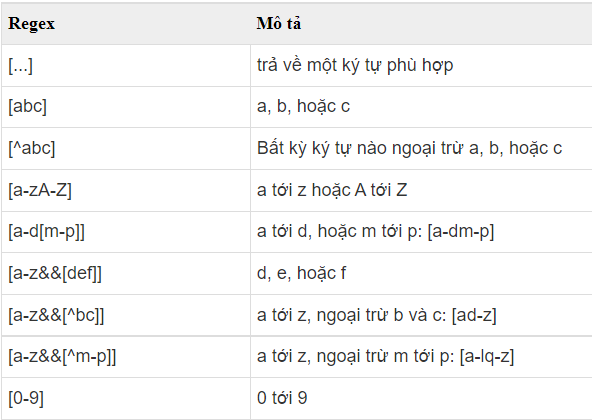
1. **Flags**

Flag trong phương thức compile () thay đổi cách thực hiện tìm kiếm.

* Pattern.CASE\_INSENSITIVE - Trường hợp các chữ cái sẽ bị bỏ qua khi thực hiện tìm kiếm.
* Pattern.LITERAL - Các ký tự đặc biệt trong mẫu sẽ không có bất kỳ ý nghĩa đặc biệt nào và sẽ được coi như các ký tự bình thường khi thực hiện tìm kiếm.
* Pattern.UNICODE\_CASE - Sử dụng nó cùng với flag CASE\_INSENSITIVE để bỏ qua trường hợp các chữ cái bên ngoài bảng chữ cái tiếng Anh

1. **RegEx Patterns**

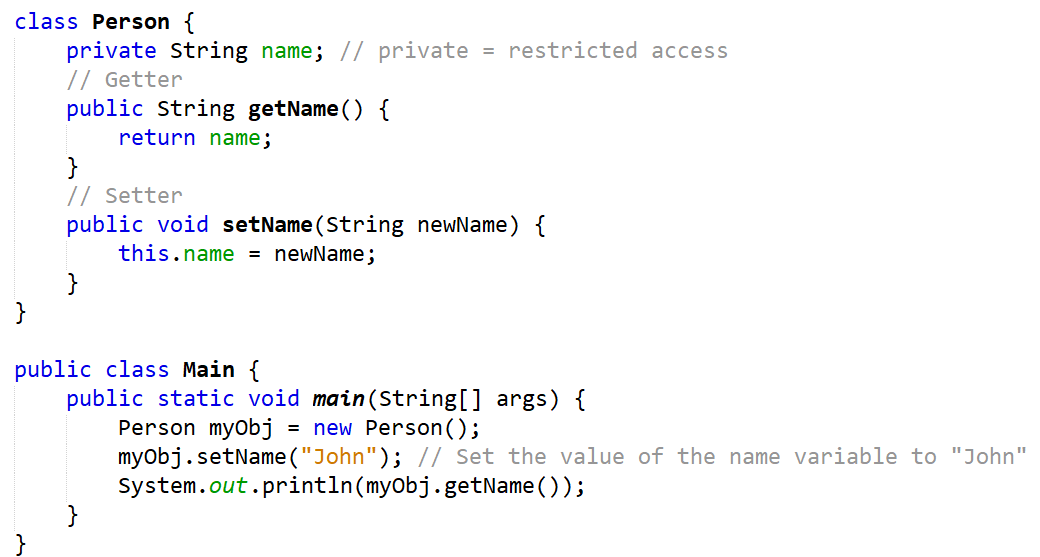
Tham số đầu tiên của phương thức Pattern.compile () là mẫu. Nó mô tả những gì đang được tìm kiếm.

****

**III. Class & Object**

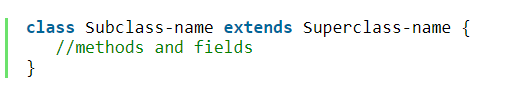
1. **Encapsulation (Tính đóng gói):** Đảm bảo rằng dữ liệu quan trọng được ẩn khỏi người dùng. Để đạt được điều này, bạn phải:

* Khai báo các biến/thuộc tính của lớp là **private**
* Cung cấp các phương thức **get** và **set** là **public** để truy cập và cập nhật giá trị của 1 biến private

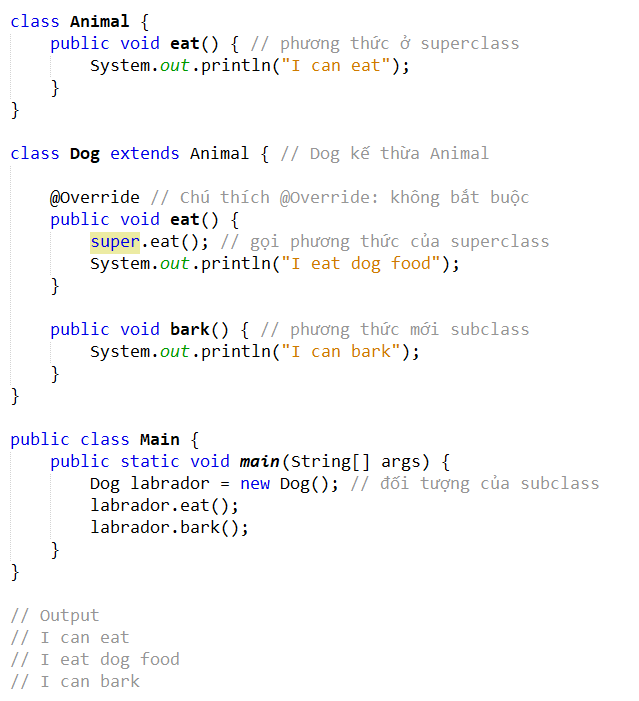


1. **Inheritance (Tính kế thừa)**

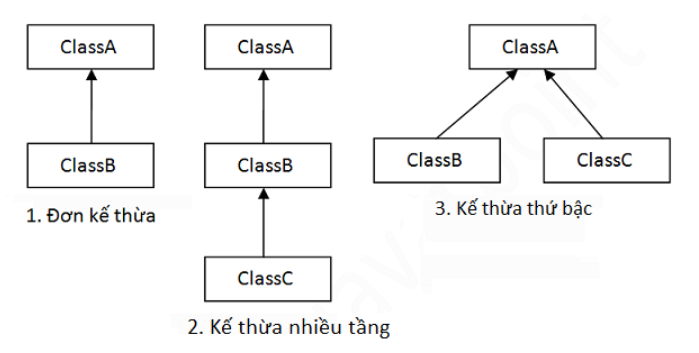
* Trong Java, có thể kế thừa các thuộc tính và phương thức từ lớp này sang lớp khác.
  + subclass (con): lớp kế thừa từ lớp khác
  + superclass (cha): lớp được kế thừa
* Syntax:



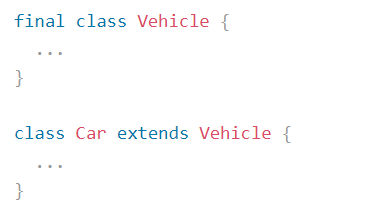
* Overriding



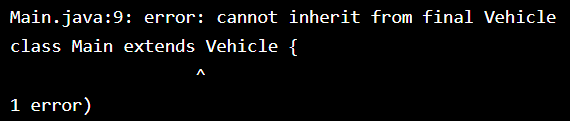
* Kế thừa là một mối quan hệ is-a. Chúng ta sử dụng kế thừa chỉ khi có một mối quan hệ is-a giữa hai lớp.
* Sử dụng kế thừa giúp
  + Tránh việc trùng lặp dữ liệu
  + Tăng tính tái sử dụng của code
* Các kiểu kế thừa trong java



* + Khi một class được kế thừa từ nhiều class được gọi là đa kế thừa. Trong java, đa kế thừa chỉ được support thông qua interface.
* Nếu không muốn các lớp khác kế thừa từ 1 lớp, sử dụng keyword final

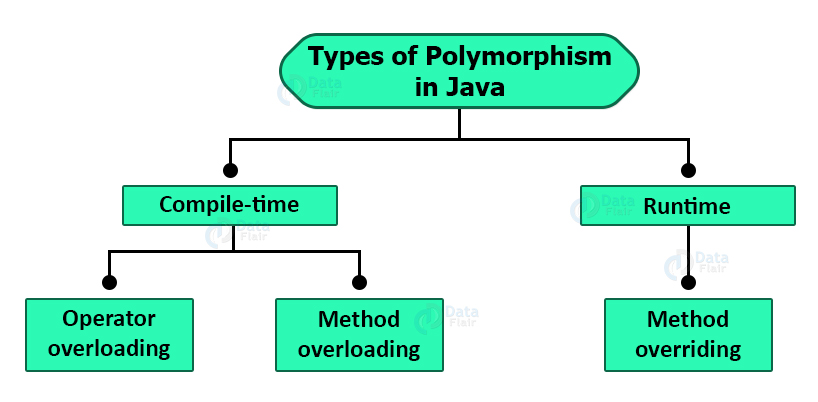


* + Output (Nếu cố truy cập final class)

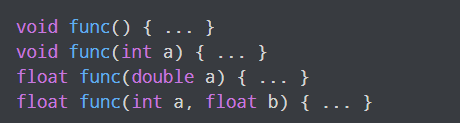


1. **Polymorphism**

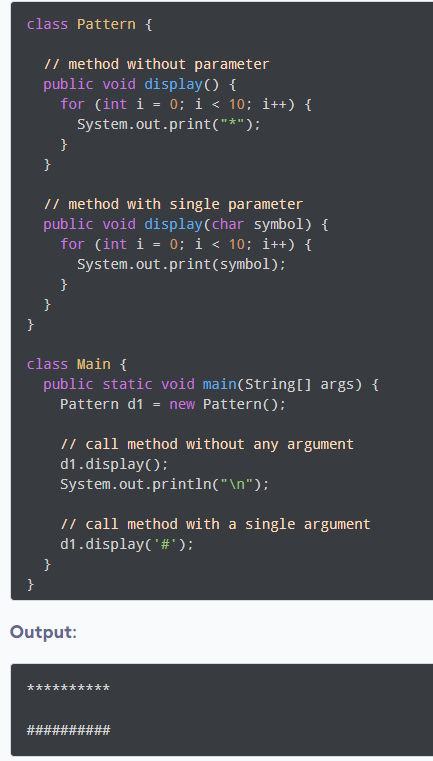
* Tính đa hình cho phép các chức năng (method) giống nhau được thực thi khác nhau trên các đối tượng khác nhau.
* Các kiểu đa hình



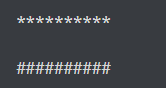
* + Đa hình thời gian biên dịch (Compile-time)
    - Operator overloading
* Một số toán tử trong Java hoạt động khác nhau với các toán hạng khác nhau. Ví dụ:
* Toán tử **+** được nạp chồng để thực hiện phép cộng số cũng như nối chuỗi
* Các toán tử như **&**, **|** , và **!** được nạp chồng cho các hoạt động logic và theo bit
* Tuy nhiên Java không hỗ trợ nạp chồng toán tử do người dùng xác định.
  + - Method overloading: Trong một Java class, chúng ta có thể tạo các phương thức có cùng tên nếu chúng khác nhau về tham số. Điều này được gọi là nạp chồng phương thức trong Java. Ở đây, cùng một phương thức sẽ thực hiện các thao tác khác nhau dựa trên tham số.



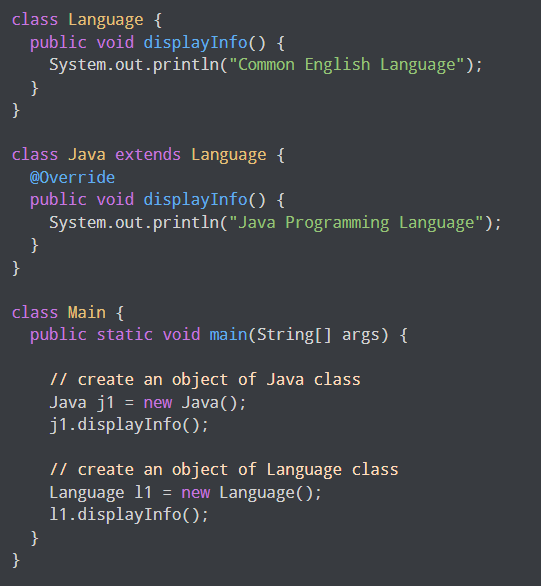
* Ví dụ:



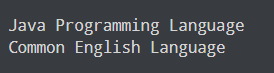
* + - * Output



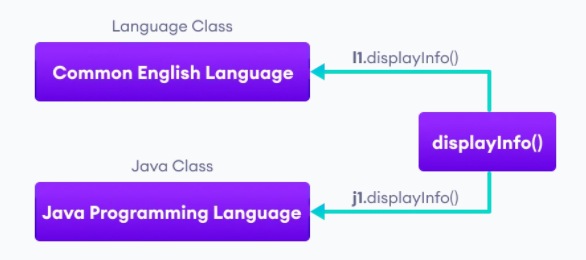
* + Đa hình thời gian thực thi (Runtime): là quá trình gọi phương thức đã được ghi đè trong thời gian thực thi chương trình
    - Method Overriding: Trong quá trình kế thừa trong Java, nếu cùng một phương thức được định nghĩa trong cả lớp cha và lớp con thì phương thức trong lớp con sẽ ghi đè phương thức tương tự trong lớp cha. Đây được gọi là ghi đè phương thức
    - Trong trường hợp này, cùng một phương thức sẽ thực hiện một thao tác trong lớp cha và một thao tác khác trong lớp con. Ví dụ:



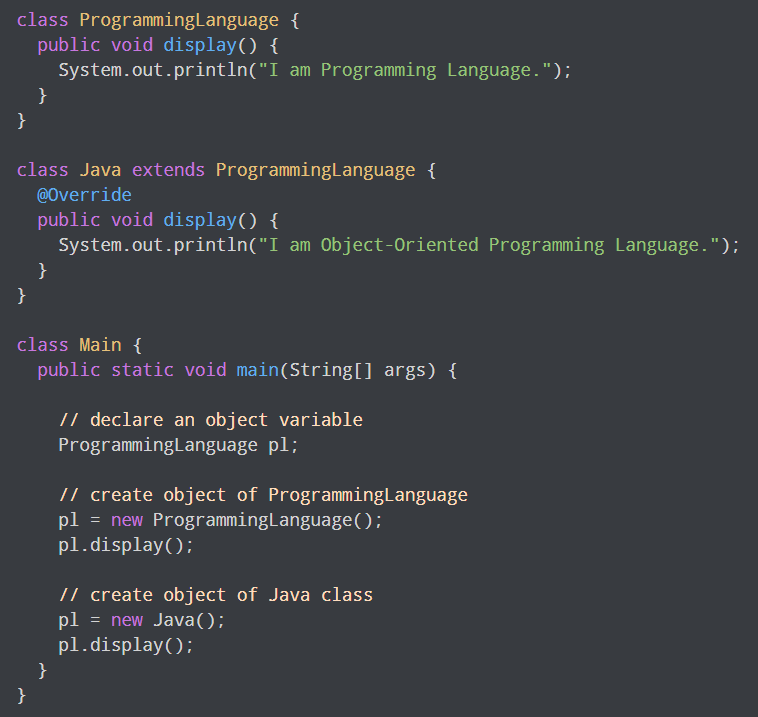
* + - * Output



* + - * Ở đây, phương thức displayInfo () có trong cả Ngôn ngữ và Java. Công dụng của displayInfo () là in thông tin. Tuy nhiên, nó đang in thông tin khác nhau bằng Ngôn ngữ và Java. Dựa trên đối tượng được sử dụng để gọi phương thức, thông tin tương ứng được in ra



* Phương thức được gọi xác định trong quá trình thực thi chương trình. Do đó, ghi đè phương thức là một đa hình thời gian chạy
  + - Biến đa hình: Một biến được gọi là đa hình nếu nó tham chiếu đến các giá trị khác nhau trong các điều kiện khác nhau. Các biến đối tượng (biến instance) đại diện cho hành vi của các biến đa hình trong Java. Đó là vì các biến đối tượng của một lớp có thể tham chiếu đến các đối tượng của lớp nó cũng như các đối tượng của các lớp con của nó.



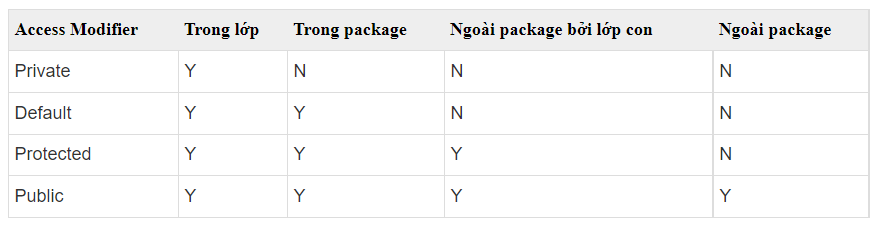
* + - * Trong ví dụ trên, chúng ta đã tạo một biến đối tượng p1 của lớp Language. Ở đây, pl là một biến đa hình. Điều này là do:
        + Trong câu lệnh p1 = new ProgrammingLanguage(), p1 tham chiếu đến đối tượng của lớp Language
        + Trong câu lệnh p1 = new Java(), p1 tham chiếu đến đối tượng của lớp Java.

⇒ Đây là một ví dụ về upcasting (khi biến đối tượng của lớp cha tham chiếu tới đối tượng của lớp con) trong Java.

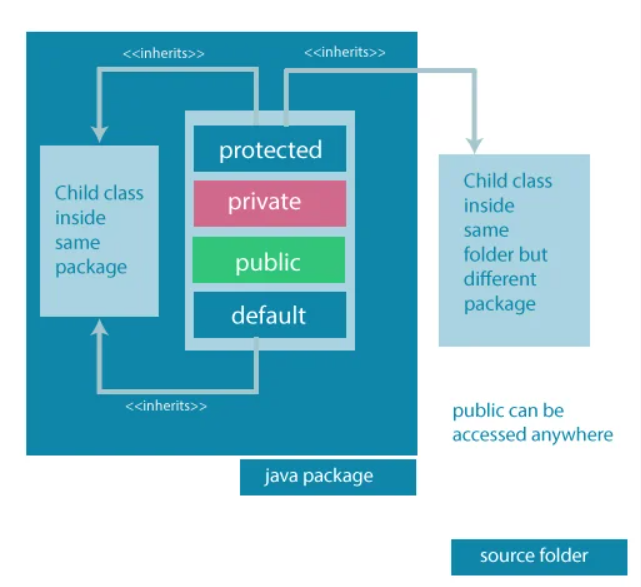
1. **Common Modifiers**

Chúng ta chia modifiers thành hai nhóm:

* Access Modifiers - kiểm soát cấp độ truy cập
* Non-Access Modifiers - không kiểm soát cấp độ truy cập, nhưng cung cấp chức năng khác
* Access Modifier
  + Access Modifier trong Java xác định phạm vi truy cập của lớp, thuộc tính, phương thức và constructor.
  + Bảng dưới đây mô tả khả năng truy cập của các Access Modifier trong java:



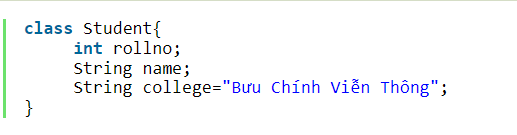
* + Hình ảnh minh hoạ

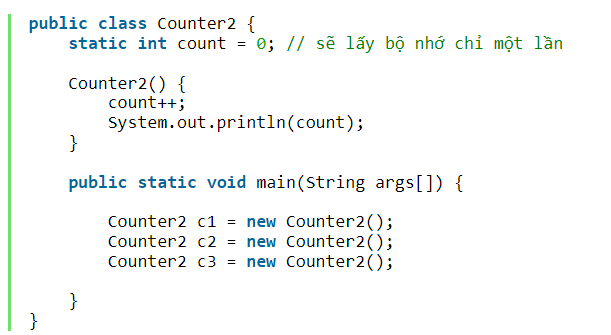
****

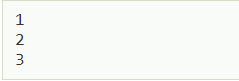
1. **Modifier static**

Từ khóa static trong Java được sử dụng chính để quản lý bộ nhớ. Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khối, các lớp lồng nhau(nested class). Từ khóa static thuộc về lớp chứ không thuộc về instance(thể hiện) của lớp.

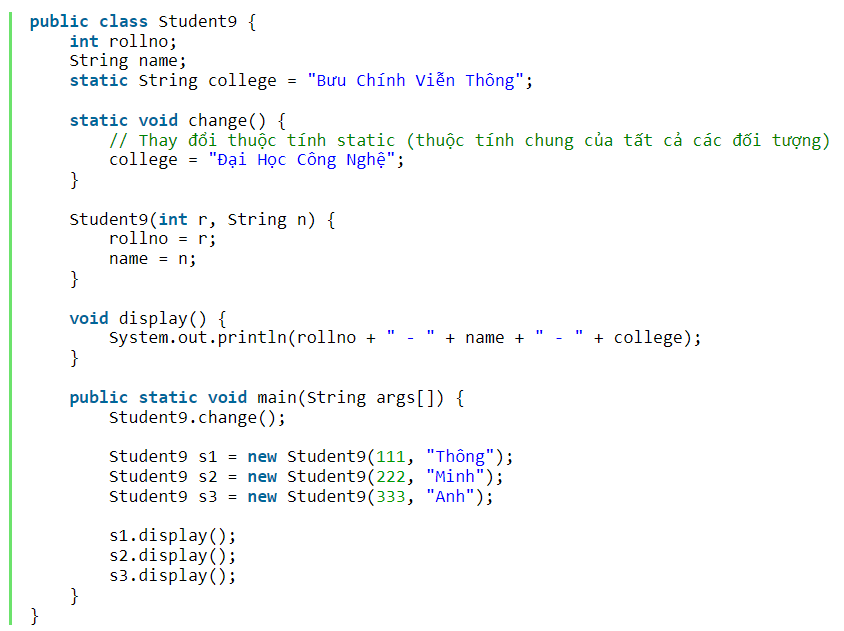
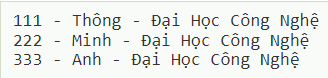
1. Biến static: Khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static

* Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, ...
* Biến static lấy bộ nhớ chỉ một lần trong Class Area tại thời gian tải lớp đó.
* Ví dụ:
* 
  + Giả sử có 1000 sinh viên trong trường đại học, bây giờ instance của các dữ liệu thành viên sẽ sử dụng bộ nhớ mỗi khi đối tượng được tạo. Tất cả sinh viên có rollno và name là thuộc tính riêng. Tuy nhiên, college là thuộc tính chung của tất cả đối tượng. Nếu chúng ta tạo nó là static, thì trường này sẽ chỉ sử dụng bộ nhớ một lần để lưu biến này.
* Thuộc tính static trong Java được chia sẻ tới tất cả đối tượng. Biến static sẽ lấy bộ nhớ chỉ một lần, nếu bất cứ đối tượng nào thay đổi giá trị của biến static, nó sẽ vẫn ghi nhớ giá trị của nó.



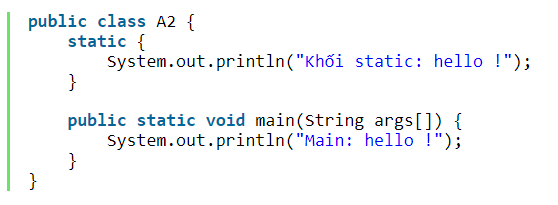
* + Output: 

1. Phương thức static: Khi bạn khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.

* Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
* Một phương thức static gọi mà không cần tạo một instance của một lớp.
* Phương thức static có thể truy cập biến static và có thể thay đổi giá trị của nó.
* 
  + Output:
* 
* Phương thức static không thể sử dụng biến non-static hoặc gọi trực tiếp phương thức non-static.

1. Khối static:

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



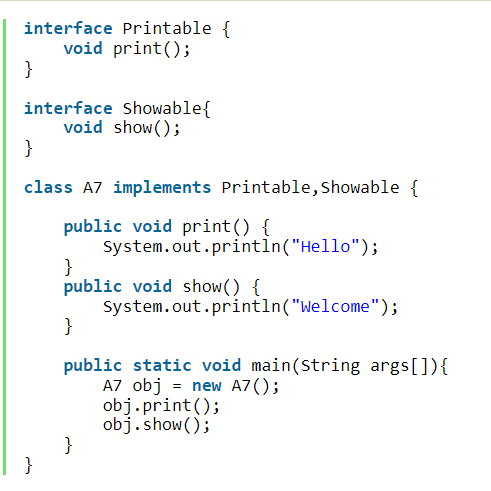
* + Output



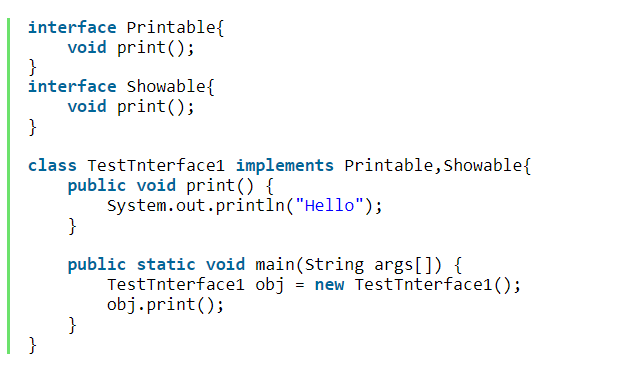
1. **Interfaces**

* Một Interface trong Java là một bản thiết kế của một lớp. Nó chỉ có các phương thức trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java. Interface trong Java cũng biểu diễn mối quan hệ IS-A. Nó không thể được khởi tạo giống như lớp trừu tượng.
  + Lưu ý: Java Compiler thêm từ khóa public và abstract trước phương thức của interface và các từ khóa public, static và final trước các thành viên dữ liệu.
* Nói cách khác, các trường của Interface là public, static và final theo mặc định và các phương thức là public và abstract.
* Một Interface trong Java là một tập hợp các phương thức trừu tượng (abstract). Một class triển khai một interface, do đó kế thừa các phương thức abstract của interface.
* Một interface không phải là một lớp. Viết một interface giống như viết một lớp, nhưng chúng có 2 định nghĩa khác nhau. Một lớp mô tả các thuộc tính và hành vi của một đối tượng. Một interface chứa các hành vi mà một class triển khai.
* Trừ khi một lớp triển khai interface là lớp trừu tượng abstract, còn lại tất cả các phương thức của interface cần được định nghĩa trong class.
* Một interface tương tự với một class bởi những điểm sau đây:
* Một interface được viết trong một file với định dạng .java, với tên của interface giống tên của file.
* Bytecode của interface được lưu trong file có định dạng .class.
* Khai báo interface trong một package, những file bytecode tương ứng cũng có cấu trúc thư mục có cùng tên package.
* Một interface khác với một class ở một số điểm sau đây:
* Bạn không thể khởi tạo một interface.
* Một interface không chứa bất cứ hàm Constructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Một interface không thể chứa một trường nào trừ các trường vừa static và final.
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.
* Đa kế thừa trong Java bởi Interface

Nếu một lớp triển khai đa kế thừa, hoặc một Interface kế thừa từ nhiều Interface thì đó là đa kế thừa.



* Đa kế thừa không được hỗ trợ thông qua lớp trong Java nhưng là có thể bởi Interface vì không có tính lưỡng nghĩa khi trình triển khai được cung cấp bởi lớp Implementation



* + Trong ví dụ trên, Printable và Showable interface có cùng các phương thức nhưng trình triển khai của nó được cung cấp bởi lớp TestInterface1, vì thế không có tính lưỡng nghĩa ở đây.

1. **Abstract Class**

Một lớp được khai báo với từ khóa abstract là lớp abstract trong Java. Lớp abstract có nghĩa là lớp trừu tượng, nó có thể có các phương thức abstract hoặc non-abstract.

1. Tính trừu tượng trong java

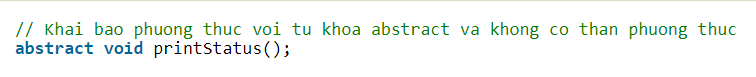
* Tính trừu tượng là một tiến trình ẩn các cài đặt chi tiết và chỉ hiển thị tính năng tới người dùng.
* Nói cách khác, nó chỉ hiển thị các thứ quan trọng tới người dùng và ẩn các chi tiết nội tại, ví dụ: để gửi tin nhắn, người dùng chỉ cần soạn text và gửi tin. Bạn không biết tiến trình xử lý nội tại về phân phối tin nhắn.
* Tính trừu tượng giúp bạn trọng tâm hơn vào đối tượng thay vì quan tâm đến cách nó thực hiện.

1. Các cách để đạt được sự trừu tượng hóa: Có 2 cách để đạt được sự trừu tượng hóa trong java

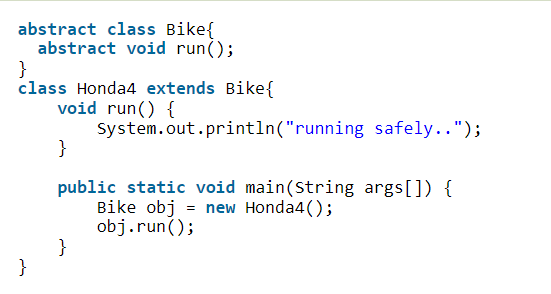
* Sử dụng lớp abstract
* Sử dụng interface

1. Phương thức trừu tượng trong Java

* Một phương thức được khai báo là abstract và không có trình triển khai thì đó là phương thức trừu tượng.
* Nếu bạn muốn một lớp chứa một phương thức cụ thể nhưng bạn muốn triển khai thực sự phương thức đó để được quyết định bởi các lớp con, thì bạn có thể khai báo phương thức đó trong lớp cha ở dạng abstract.
* Từ khóa abstract được sử dụng để khai báo một phương thức dạng abstract. Một phương thức abstract không có thân phương thức.
* Phương thức abstract sẽ không có định nghĩa, được theo sau bởi dấu chấm phẩy, không có dấu ngoặc nhọn theo sau:



* Ví dụ về lớp trừu tượng và phương thức trừu tượng trong Java

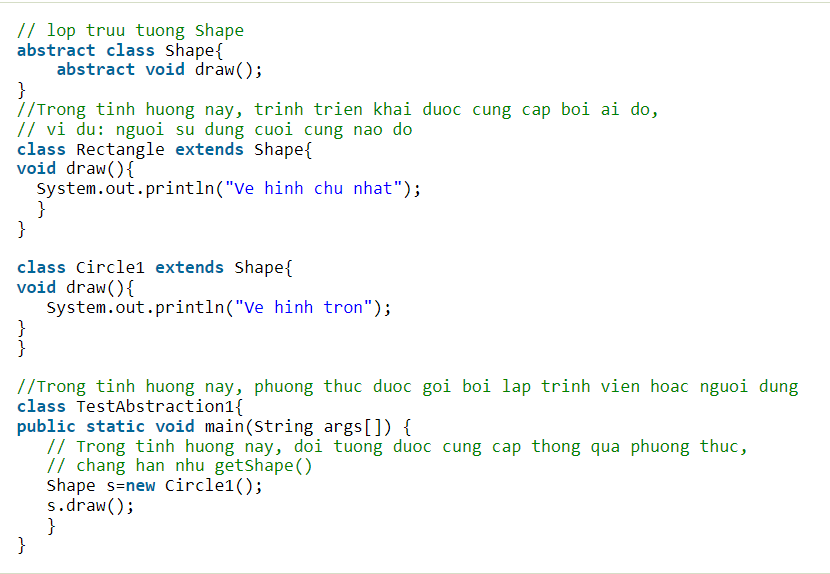


* Output



1. Kế thừa lớp Abstract trong Java

Trong ví dụ này, Shape là lớp trừu tượng, trình triển khai của nó được cung cấp bởi lớp Rectangle và lớp Circle. Hai lớp này kế thừa lớp trừu tượng Shape.



* Output

