**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

Bài 1: Đối tượng, lớp, và các thành phần của lớp trong java

1. Đối tượng

* Khái niệm: một thực thể dược gọi là đối tượng khi nó chứa 2 đặc điểm: trạng thái (state) và hành vi( behavior)

1. Lớp

* Là một tập hợp các đối tượng có cùng trạnh thái và hành vi
* Mỗi đối tượng là một thể hiện ( instance) của một lớp, thông thường các đối tượng ở cùng 1 lớp sẽ có cùng hành vi( behavior ) , nghĩa là cách thức hoạt động giống nhau, cách thức hoạt động của một đối tượng được thiết lập nhờ các phương thức ( method) của lớp đó.
* Cấu trúc của một lớp:

[access\_modifier] class NameClass{

[access\_modifier] [datatype] nameVariable ;

[access\_modifier] [result\_type] NameOfmethod(){}

}

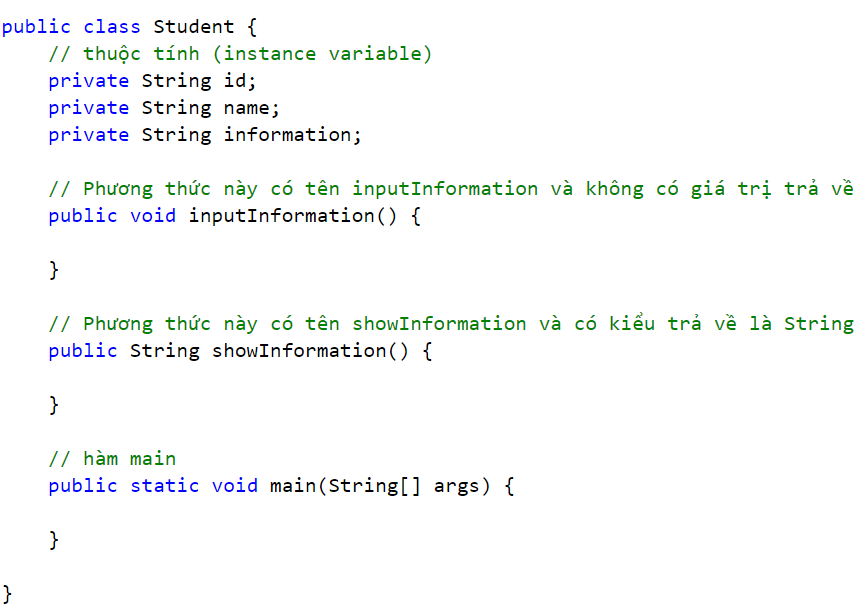
+ Trong đó: [access\_modifier]: phạm vi truy cập của lớp, thuộc tính, phương thức

+ NameClass: tên lớp, mỗi lớp có một tên duy nhất để phân biệt với lớp khác trong cùng phạm vi

+ nameVariable: tên thuộc tính, mô tả trường để lưu dữ liệu cho đối tượng của lớp đang mô tả, hay lưu tham chiếu đến các đối tượng của lớp khác.

+ NameOfMethod: tên phương thức, mỗi phương thức của 1 lớp thực chất là 1 hàm được viết riêng cho các đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng của lớp này.

Example:



* Access modifier: phạm vi truy cập của lớp, thuộc tính, phương thức

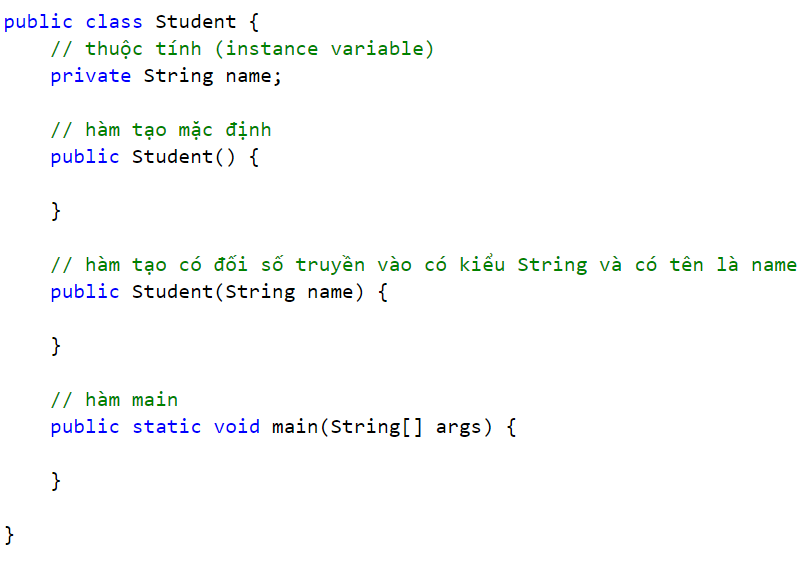
+ Có 4 phạm vi truy cập: public, private, protected, default( mặc định ).

Trong đó:

* Public: có thể truy cập mọi nơi trong project
* Protected: truy cập trong cùng 1 package, lớp con của lớp khai báo
* Default: truy cập trong cùng 1 package
* Private: chỉ có thể truy cập bên trong lớp
* Thuộc tính ( instance variable ): là 1 biến được khai báo bên trong lớp nhưng ngoài phương thức, hàm tạo hoặc khối lệnh.
* Phương thức: là một hàm được viết riêng cho đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng của lớp này. Bao gồm 3 thành phần ( phạm vi truy cập, kiểu trả về, tên )

1. Hàm tạo ( constructor)

* Đặc điểm: là 1 hàm đặc biệt trong hướng đối tượng, trong java hàm tạo có cùng tên với lớp và không có giá trị trả về

Example:

* Trong 1 lớp có 1 hoặc nhiều hàm tạo, có 1 loại thường gặp: hàm mặc định( không có đối số) và hàm tạo có đối số
* Hàm tạo mặc định: là hàm tạo không có tham số đầu vào, trong trường hợp người lập trình không định nghĩa hàm tạo thì trình biên dịch sẽ tự động thêm hàm tạo mặc định cho lớp.
* Hàm tạo có đối số: là hàm tạo được người lập trình định nghĩa với các tham số đầu vào khác nhau để khới tạo dữ liệu cho đối tượng.

Note: khi người lập trình định nghĩa hàm tạo thì trình biên dịch sẽ không tự động tạo ra hàm tạo mặc định cho lớp này nữa. trong trường hợp này nếu muốn dung hàm tạo mặc định, người dung phải tự định nghĩa nó ra.

Bài 2: Thao tác trên đối tượng và phạm vi truy cập trong java

Trong bài này tìm hiểu về: Thao tác trên đối tượng, thuộc tính và phương thức của lớp trong java.

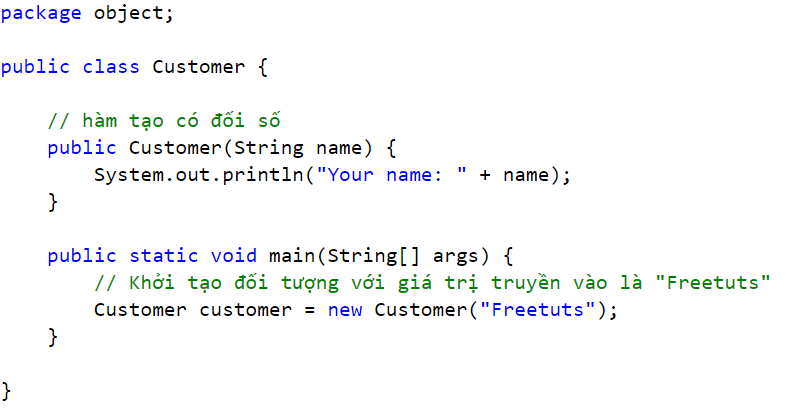
1. Các bước tạo mới 1 đối tượng:

Trong java để tạo mới 1 đối tượng chúng ta sử dụng từ khóa: new

Để khới tạo 1 đối tượng có 3 bước sau:

* Khai báo tên và kiểu dữ liệu cho đối tượng
* Sử dụng từ khóa “new” để tạo đối tượng
* Theo sau từ khóa “new” là một lời mời đến hàm tạo của lớp( hàm tạo này sẽ khởi tạo giá trị cho đối tượng được khai báo ).

Example:



1. Truy cập đến thuộc tính, phương thức của lớp

Chúng ta truy cập đến thuộc tính, phưng thức thông qua đối tượng được khới tạo:

* Đến thuộc tính: ten\_doi\_tuong.ten\_thuoc\_tinh
* Đến phương thức: tên\_doi\_tuong.ten\_phuong\_thuc

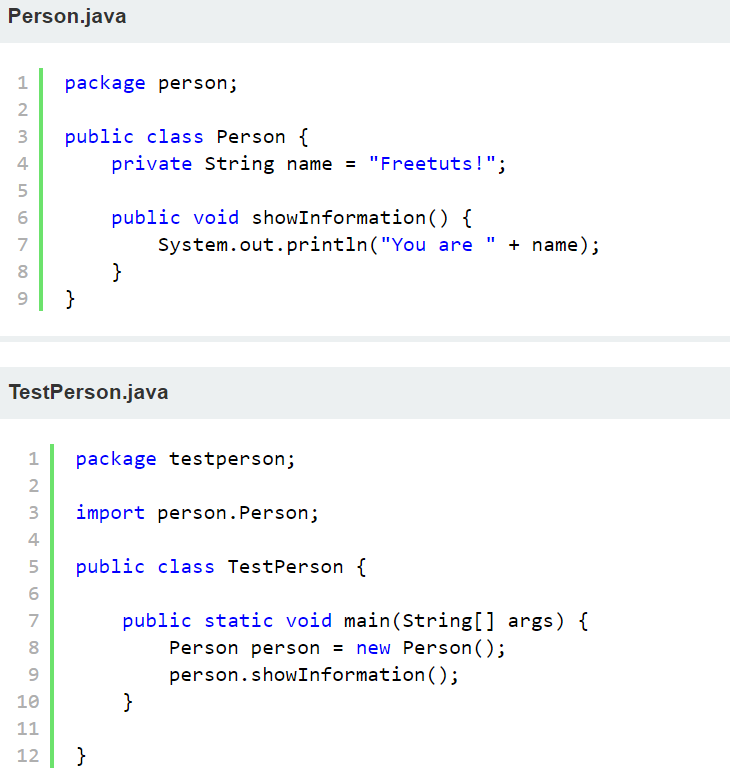
1. Phạm vi truy cập (access modifier ) của lớp, thuộc tính, phương thức

Như phần trước đã nói, chúng ta có 4 phạm vi:

* Public: truy cập mọi nơi trong project
* Protected: truy cập trong cùng 1 package, trong lớp con của lớp khai báo(với trường hợp khác package)
* Default: trong cùng 1 package
* Private: chỉ bên trong lớp khai báo

Note: nếu truy cập 2 package khác nhau thì phải import( tích hợp ) class cần truy cập vào trong class của chúng ta

Example:



Note: protected, private: không dung đề khai báo cho lớp và interface

Bài 3: Tính đóng gói trong java

Trong lập trình hướng đối tượng có 4 tính chất quan trọng:

* Tính đóng gói: **Encapsulation**
* Tính kế thừa: **Inheritance**
* Tính đa hình: **Polymorphism**
* Tính trừu tượng: **Abstraction**

Trong bài này chúng ta sẽ học về tính chất đầu tiên là đóng gói trong java và cách vận dụng nó trong lập trình

1. Khái niệm:

Trong bài trước chúng ta truy cập đến thuộc tính của đối tượng thông qua tên của thuộc tính đó, tuy nhiên chúng ta nên hạn chế dung cách này vì nó không đảm bảo một trong những nguyên tắc quan trọng của lập trình hướng đối tượng là bảo mật và che dấu thông tin. Vì vậy để khắc phục điều này trong lập trình hướng đối tượng có 1 tính chất có thể cho chúng ta truy cập, thay đổi giá trị thuộc tính thông qua phương thức của lớp, đó là tính đóng gói: **Encapsulation**

* Đóng gói: Nghĩa là thông tin của một đối tượng chỉ được truy xuất thông qua hệ thống các phương thức có sẵn của lớp( chỉ có gọi đến các phương thức này để hiển thị thông tin ). Vì vậy nó còn được gọi là **data hiding** ( che dấu dữ liệu )

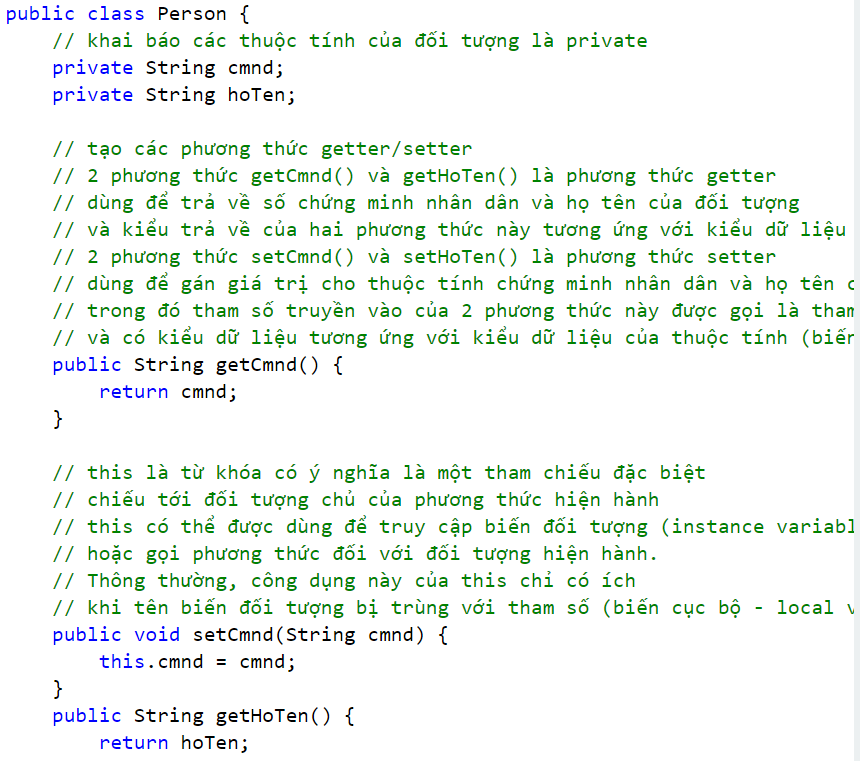
Những đặc điểm của tính đóng gói:

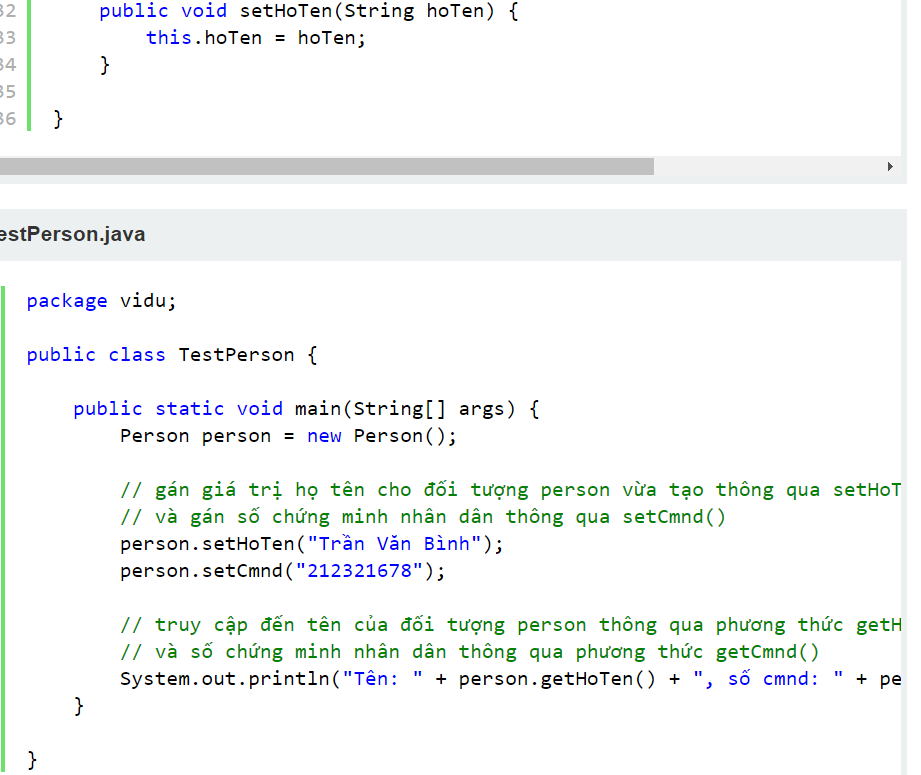
* Tạo cơ chế ngăn cách gọi phương thức của lớp này ảnh hưởng hay truy xuất thông tin về đối tượng của lớp khác
* Khi được khai báo private dữ liệu của đối tượng được bảo vệ khỏi sự truy xuất không hợp lệ từ bên ngoài
* Người lập trình dựa vào cơ chế này để ngăn ngừa việc gán dữ liệu không hợp lệ vào thành phần dữ liệu của đối tượng
* Cho phép thay đổi cấu trúc bên trong của một lớp mà không làm ảnh hưởng đến những lớp bên ngoài sử dụng lớp đó

Để cài đặt tính đóng gói, thực hiện 2 bước sau:

* Khai báo các thuộc tính của đối tượng là **private** ( để các lớp khác không truy cập trực tiếp hay sửa đổi được )
* Cung cấp các phương thức **getter**/**setter** có phạm vi **public,** kiểu trả về cùng với kiểu dữ liệu của thuộc tính để truy cập và sửa đổi các giá trị của thuộc tính trong lớp.
* Getter: truy cập vào thuộc tính của đối tượng, và trả về giá trị thuộc tính đó.
* Setter: truy cập vào thuộc tính của đối tượng, và gán giá trị cho thuộc tính đó.

Example:





Giải thích:

* Trong lớp person:
* This.hoten: gọi đến biến hoten (thuộc tính họ tên của đối tượng)
* Hoten: tham số của hàm tạo, để khởi tạo giá trị cho biến đối tượng
* Từ khóa this dùng để gọi thuộc tính ( biến đối tượng instance variable), nó dung để phần biệt với tham số của hàm tạo hay là biến cục bộ ( local variable)

Bài 4: Tính kế thừa trong java (**Inheritance**)

1. Khái niệm:

Được hiểu là chúng ta xây dựng một lớp mới( lớp con ) trên cơ sở một lớp sẵn có, lớp mới sẽ được thừa hưởng các thuộc tính, phương thức của lớp sẵn có ( lớp cha )

1. Tác dụng của việc kế thừa:

* Lớp con có thể tận dụng lại các thuộc tính và phương thức của lớp cha
* Lớp con có thể định nghĩa thêm thuộc tính, phương thức mới của riêng nó cho phù hợp với mục đích tạo ra

Có 3 các dạng kế thừa chính: kế thừa từ **class**, kế thừa từ lớp trừu tượng (**Abstract class**) và kế thừa từ **interface**.

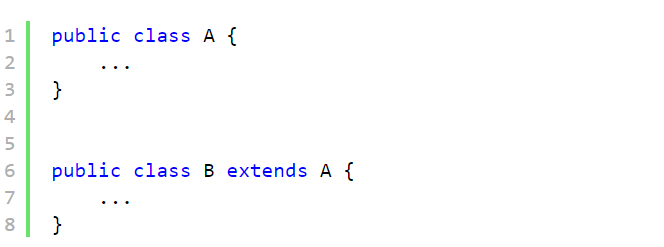
1. Xây dựng lớp con và lớp cha

* Lớp cha: chứa các thông tin chung giữa các đối tượng ( bao gồm cả phương thức , thuộc tính )
* Lớp con: chứa những thông tin riêng của từng đối tượng cụ thể

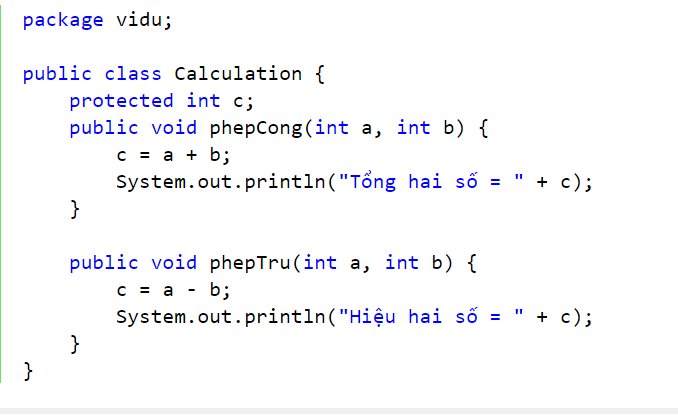
Example:

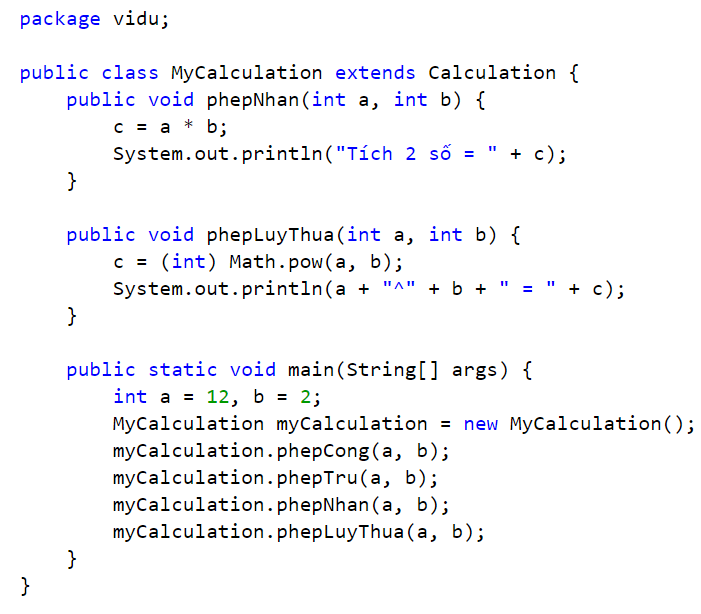
Để cài đặt tính kế thừa chúng ta sử dụng từ khóa **extends**

Ví du: lớp B kế thừa lớp A:



**1 ví dụ khác:**





**Note:** Nếu 1 lớp không kề thừa từ lớp nào thì mặc định nó sẽ kế thừa 1 lớp gọi là **Object**

1. Từ khóa **super**

Được sử dụng trong trường hợp sau:

* Để phân biệt có cùng tên giữa lớp cha và lớp con
* Gọi hàm tạo của lớp cha từ lớp con

+ Phân biệt thành phần cùng tên giữa 2 lớp cha, con:

Cú pháp: super.ten\_thuoc\_tinh, super.ten\_phuong\_thuc

+ Gọi hàm tạo của lớp cha từ lớp con:

Cú pháp: super(doi\_so);

**Note**: dung super để gọi hàm tạo từ lớp cha thì phải use trong hàm tạo của lớp con và phải đặt ở dòng đầu tiên.

# Bài 5: tính đa hình (Polymorphism) trong java

# Khái niệm:

# Kỹ thuật đa hình trong lập trình hướng đối tượng giúp lập trình viên tái sử dụng được đoạn code viết một cách tổng quát, thay đổi cách ứng xử linh hoạt tùy theo đối tượng.

# Tính đa hình trong java được hiểu là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau.

# Example: khi bạn đến trường là sinh viên thì nhiệm vụ là nghe giảng, học bài …., còn khi về nhà bạn đóng vai trò là thành viên trong gia đình, có nhiệm vụ giặt quần áo, rửa chén…, còn khi đến siêu thì bạn là khách hàng, nhiệm vụ mua hàng

# Có thể hiểu đại khái tính đa hình của đối tượng: là trong từng trường hợp, hoàn cảnh thì đối tượng thực hiện những công việc khác nhau.

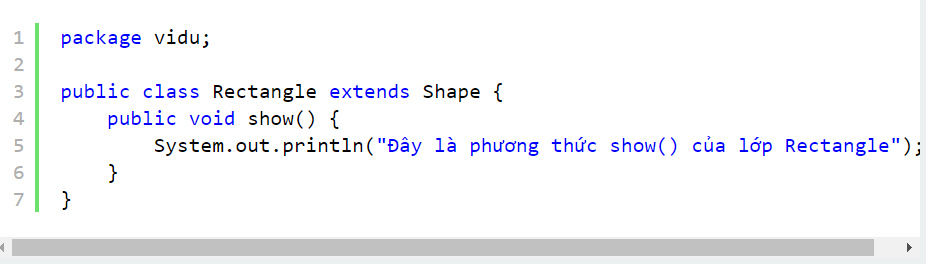
# Để thể hiện tính đa hình, cần đảm bảo 2 điều kiện sau:

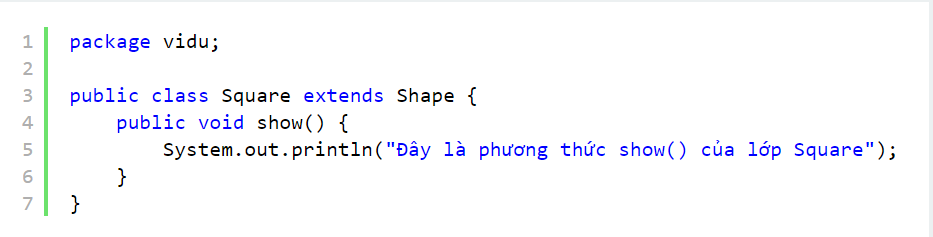
# Các lớp phải có quan hệ kế thừa với 1 lớp cha nào đó

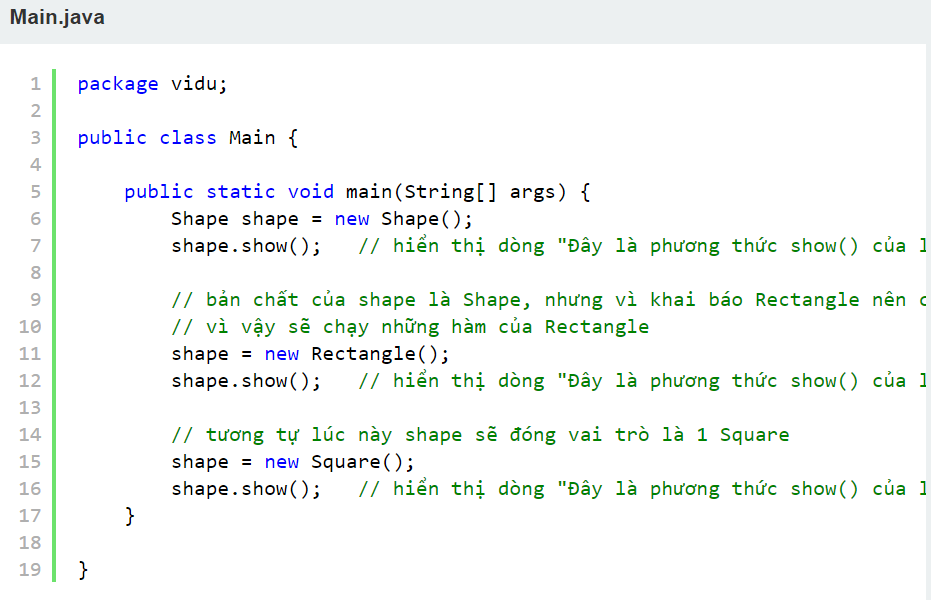
# Phương thức đa hình được ghi đè (Override) ở lớp con, tính đa hình chỉ được thể hiện khi đã ghi đè lên phương thức của lớp cha

# Ví dụ về tính đa hình và cách sử dụng

# 







**Giải thích**:

Trong ví dụ trên, có 3 class: **Shape**, **Rectangle**, **Square**, tương ứng có 3 phương thức **show()** tương ứng

Trong đó 2 lớp **Rectangle**, **Square** kế thừa lớp **Shape,** ở hàm main chúng ta gọi đến 3 phương thức **show(),** nếu là khi trước thì để gọi chúng ta cần phải tạo ra 3 đối tượng tương ứng cho từng lớp, còn ở đây thì chỉ tạo 1 đối tượng **shape** của lớp cha **Shape**, shape đóng vai trò là lớp con thông qua 2 dòng code:

**shape = new Rectangle();**

**shape = new Square();**

lúc này đối tượng **shape** đóng vai trò là đối tượng của lớp con tương ứng

đây là ý nghĩa của tính đa hình.

Bài 6: Tính trừu tượng

1. Khái niệm:

Tính trừu tượng trong java là tính chất không thể hiện cụ thể, đó là 1 quá trình che dấu các hoạt động bên trong và chỉ hiển thị những **tính năng thiết yếu của đối tượng tới người dùng.**

Ví dụ: việc gửi tin nhắn, người dùng chỉ quan tâm đến nội dung tin gửi đi, đối tượng gửi, kết quả gửi chứ không quan tâm đến quá trình gửi đi ntn.

Ưu điểm:

* Cho phép lập trình viên loại bỏ đi tính chất phức tạp của đối tượng bằng cách chỉ đưa ra các phương thức và thuộc tính cần thiết của đối tượng, cải thiện khả năng bảo trì của hệ thống
* Giúp ta tập trung vào những thứ cốt lõi, cần thiết thay vì quan tâm nó thực hiện ntn
* Cung cấp nhiều tính năng mở rộng khi sử dụng kết hợp với tính đá hình, kế thừa trong lập trình hướng đối tượng.

1. Phương thức trừu tượng ( Abstract method ) và lớp trừu tượng ( Abstract class)
   1. Phương thức trừu tượng:

Các phương thức mà chỉ có phần khai báo mà trong dấu {} không có gì, và được kết thức bằng dấu “ ; “ là phương thức trừu tượng

Cú pháp: để tạo phương thức trừu tượng dùng từ khóa **Abstract**

**Access\_modifier abstract data\_type NameOfMethod ();**

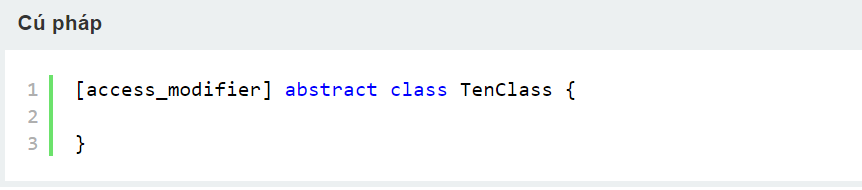
Trong đó:

* **Access\_modifier**: phạm vi truy cập ( không được để **private** )
* **Data\_type**: kiểu dữ liệu trả về
* **NameOfMethod**: tên phương thức
* **Parameter**: tham số truyền vào ( có thể có hoặc không )

**Note**: để sử dụng phương thức này cần ghi đè ( **Override** ) phương thức này ở lớp con của lớp khai báo nó.

* 1. Lớp trừu tượng

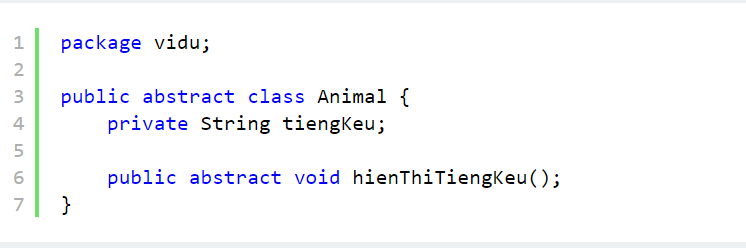
Là lớp có **abstract** ở trước tên của lớp, ví dụ:

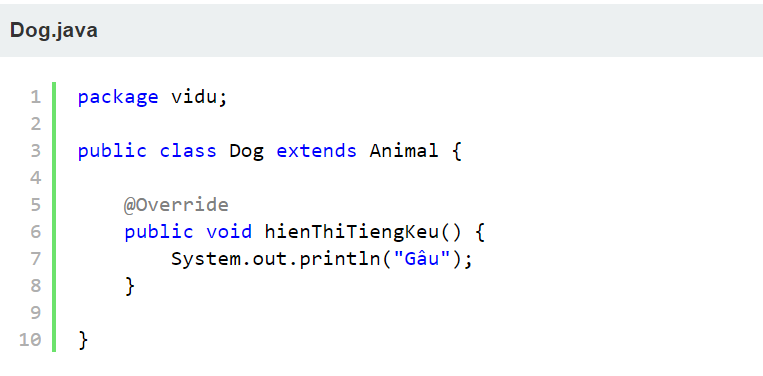


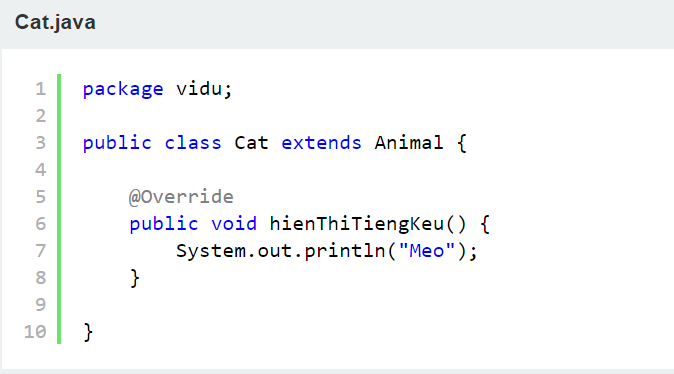
**Note**:

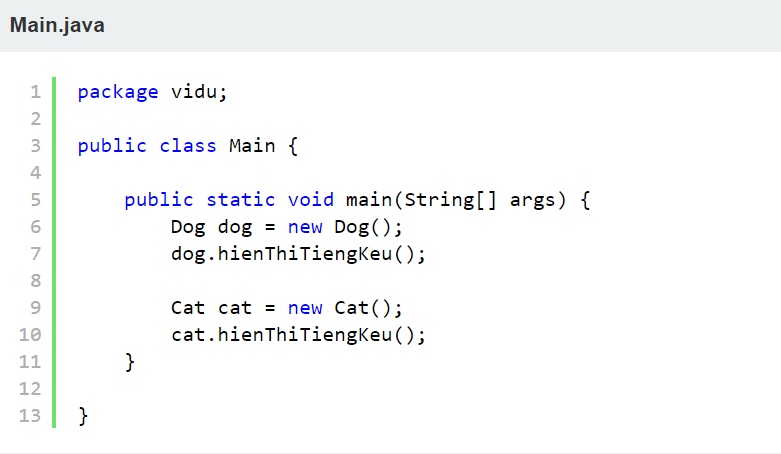
* Nếu một lớp là lớp trừu tượng thì chúng ta không thể dùng trực tiếp nó để tạo ra đối tượng và phải viết 1 lớp kế thừa lớp trừu tượng đó
* Lớp trừu tượng có thể có hoặc không phương thức trừu tượng. Nếu 1 lớp chứa phương thức trừu tượng thì nó phải được khai báo là lớp trừu tượng.
* Khi 1 lớp kế thừa 1 lớp trừu tượng thì lớp con đó phải Override lại tất cả các phương thức trừu tượng có trong lớp đó.
* Lớp trừu tượng là lớp không thể khởi tạo đối tượng từ nó, nhưng nó lại ràng buộc các lớp con kế thừa trực tiếp nó phải có các phương thức trừu tượng của nó thông qua sự ghi đè( override ).

**Example:**









**Kết luận**:

* Nếu lớp cha là trừu tượng thì lớp phải ghi đè tất cả các phương thức trừu tượng trong lớp cha
* Việc tạo ra lớp cha trừu tượng giúp giảm sự phức tạp đối với từng đối tượng cụ thể. Người ta nhìn váo đó có thể thấy được tổng quát các hành động, thuộc tính của lớp con.

Bài 7: **Alias** và từ khóa **static** trong java

1. **Alias** và cơ chế gom rác tự động

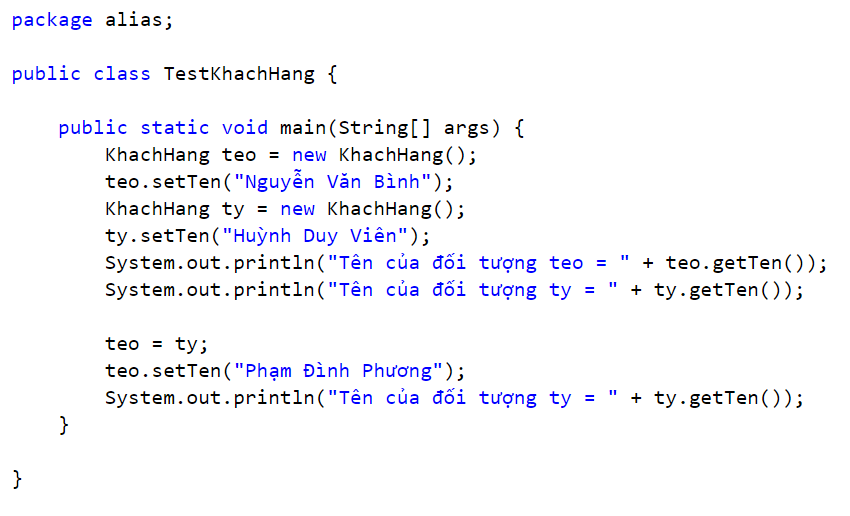
**Alias** là khả năng mà 1 ô nhớ có nhiều đối tượng cùng trỏ tới ( >= 2 đối tượng )

Cơ chế gom rác tự động còn gọi là **gargage collection**, cơ chế này tự động thu hồi bộ nhớ khi ô nhớ đó không còn đối tượng nào quản lý.

Ví dụ:

* Chúng ta tạo ra đối tượng teo của lớp KhachHang, và đối tượng teo sẽ trỏ tới ô nhớ X nào đó
* Tương tự tạo ra đối tượng ty của lớp KhachHang, và nó cũng trỏ đến ô nhớ Y nào đó.
* Hai đối tượng **teo**, **ty** trỏ tới ( hay là quản lý) 2 ô nhớ X, Y độc lập. tức là đối tượng teo thao tác tên ô nhớ X không làm ảnh hướng đến đối tượng **ty** và ngược lại. bây giờ ta gán **teo** = **ty** thì hệ thống máy tính sẽ làm việc theo cơ chế sau : **teo** trỏ đến ô nhớ mà **ty** đang quản lý(Y) => ô nhớ Y có 2 đối tượng **teo**, **ty** cùng quản lý => xuất hiện alias ở ô nhớ Y => xuất hiện hiện tượng như sau:
* Tại ô nhớ Y, nếu **teo** thay đổi thông tin thì sẽ làm cho **ty** thay đổi thông tin và ngược lại( vì cả hai cùng quản lý một ô nhớ)
* Ô nhớ X bây giờ không còn đối tượng nào quản lý dẫn đến trường hợp hệ thống sẽ tự thu hồi bộ nhớ đã cấp phát. Đó là cơ chế thu hồi rác tự động.

Example:



**Giải thích**: ở đây **teo** làm thay đổi giá trị của **ty**

1. Từ khóa static

Trong java từ khóa **static** chủ yếu được dùng để quản lý bộ nhớ, có thể áp dụng static với phương thức, thuộc tính, khối lệnh. Từ khóa **static** thuộc về lớp chứ không thuộc vế các đối tượng được tạo từ lớp. **Biến, phương thức được khai báo static sẽ gọi là biến tính, phương thức tĩnh**

Đặc điểm chung của các thành phần được khai báo static

* Được cấp phát bộ nhớ một lần duy nhất ngay khi biên dịch chương trình, và ra đời trước các đối tượng
* Có thể dùng chung cho mọi đối tượng
* Được truy xuất trực tiếp từ tên lớp mà không cần khởi tạo đổi tượng từ lớp đó
* Được hủy hết khi kết thúc chương trình
  1. Biến tĩnh/thuộc tính tĩnh ( static variable )

Biến tĩnh ( static variable) là biến dùng chung cho mọi đối tượng của lớp. nó cũng tương tự như hằng số nhưng linh hoạt hơn ở chỗ nó có thể thay đổi được giá trị.

Cú pháp:

Khởi tạo biến static:

**Access\_modifier static data\_type nameOfVariable = value;**

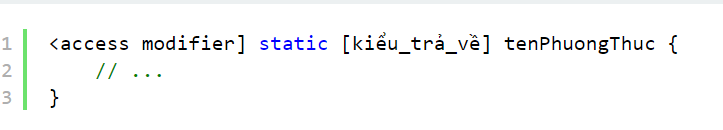
Biến tĩnh được truy xuất trực tiếp từ tên lớp:

NameOfClass . nameOfVariable;

* 1. Phương thức tĩnh

Cú pháp:

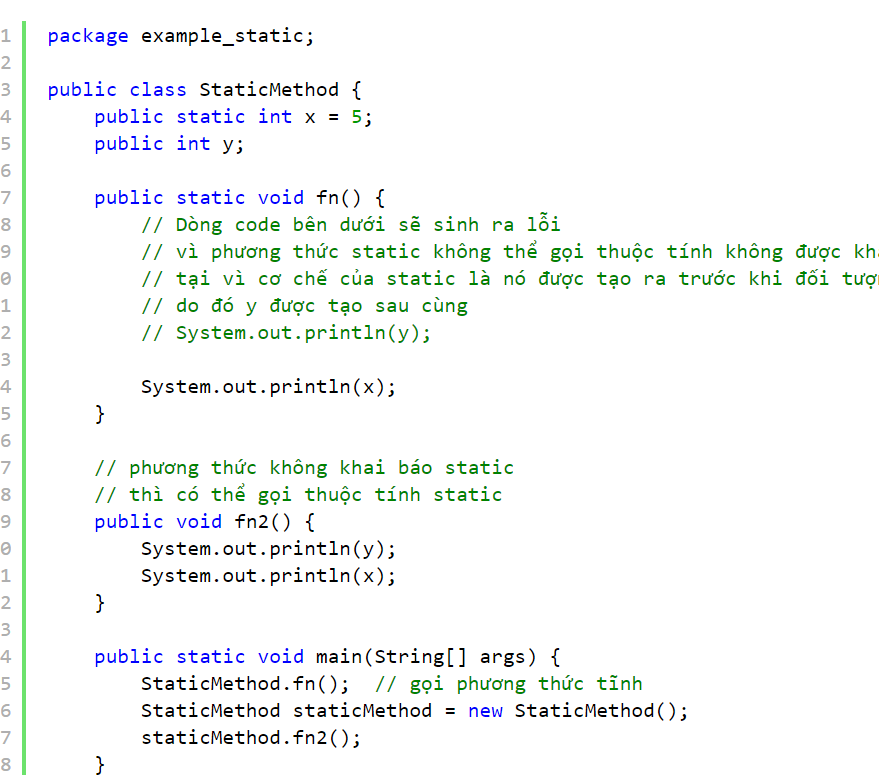
**Khởi tạo**:



**Truy xuất**: NameOfClass . NameOfMethod

**Note**: các phương thức được khởi tạo static không thể gọi phương thức hay thuộc tính không static, do phương thức này xuất hiện trước, không tìm thấy các phương thức thông thường khác.

Example:



Khi nào chúng ta sử dụng static: khi chúng tá cần chia sẻ các phương thức, thuộc tính cho toàn bộ hệ thống của mình, ví dụ: các hàm đọc file, ghi file, kết nối cơ sở dữ liệu.