**Encapsulation & Abstraction**

**(Đóng gói và trừu tượng)**

1. **Theory**
2. **Khái quát về OOP (Lập trình hướng đối tượng)**
3. **Khái niệm**

Lập trình hướng đối tượng (OOP) là một mô hình lập trình dựa trên khái niệm về lớp và đối tượng. Nó được sử dụng để cấu trúc một chương trình phần mềm thành các phần đơn giản, có thể tái sử dụng của các bản thiết kế mã (thường được gọi là các lớp), được sử dụng để tạo các thể hiện riêng lẻ của các đối tượng.

* Lập trình hướng đối tượng là một kỹ thuật lập trình cho phép lập trình viên tạo ra các *đối tượng* trong code trừu tượng hóa các *đối tượng thực tế trong cuộc sống*.

1. **Tại sao chúng ta cần Lập trình hướng đối tượng**

Lập trình hướng đối tượng được phát triển vì những hạn chế đã được phát hiện trong các cách tiếp cận lập trình trước đó.

* 1. ***Lập trình hướng cấu trúc (Lập trình thủ tục)***

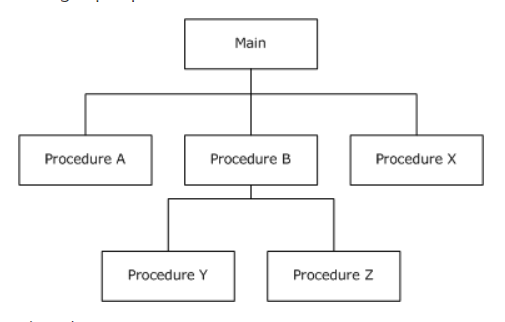
Lập trình thủ tục là mô hình lập trình mệnh lệnh. Ở mô hình này bạn sẽ sử dụng 3 cấu trúc chính để làm việc:

* **Tuần tự:** Các câu lệnh được thực hiện theo trình tự nhất định
* **Quyết định:** Chương trình chạy phụ thuộc vào điều kiện nhất định

nào đó. (If, Else là từ khoá đại diện)

* **Vòng lặp:** Các mệnh lệnh được lặp đi lặp lại khi điều kiện nào đó vẫn được thoả mãn (For, While)

🡺 Lập trình thủ tục là viết các thủ tục hoặc hàm thực hiện các thao tác trên dữ liệu, nghĩa là: Chia chương trình thành các hàm (function) chính, rồi từ hàm chính lại chia nhỏ ra các hàm khác sao cho chia nhỏ đến mức mà các hàm thực hiện các công việc cụ thể nhất.



* **Ưu điểm:**
  + Chương trình được module hoá, dễ hiểu, dễ theo dõi.
  + Tư duy giải thuật rõ ràng.
* **Nhược điểm:**
  + Dữ liệu và xử lý tách rời dẫn đến không quản lí được sự thay đổi dữ liệu khi có nhiều chương trình cùng thay đổi một biến chung.
  + Khi cấu trúc dữ liệu thay đổi sẽ dẫn đến thuật toán bị thay đổi, và do đó, phải viết lại mã chương trình từ đầu.
    - * Không tiết kiệm được tài nguyên
  1. ***Lập trình hướng đối tượng***
* Lập trình thủ tục là viết các thủ tục hoặc hàm thực hiện các thao tác trên dữ liệu, trong khi lập trình hướng đối tượng là tạo các đối tượng chứa cả dữ liệu và hàm.
* OOP có một số ưu điểm so với lập trình thủ tục:
* Đóng gói dữ liệu để hạn chế sự truy nhập tự do vào dữ liệu
  + - * dễ dàng quản lý code khi có sự thay đổi chương trình
* Cho phép sử dụng lại mã nguồn, hạn chế việc phải viết lại

mã từ đầu cho các chương trình.

1. **Class – Object – Con trỏ this**

Class và Object là 2 khía cạnh chính của OOP. Vì vậy, một lớp là một khuôn mẫu cho các đối tượng và một đối tượng là một thể hiện của một lớp. Khi các đối tượng riêng lẻ được tạo, chúng kế thừa tất cả các biến và hàm từ lớp.

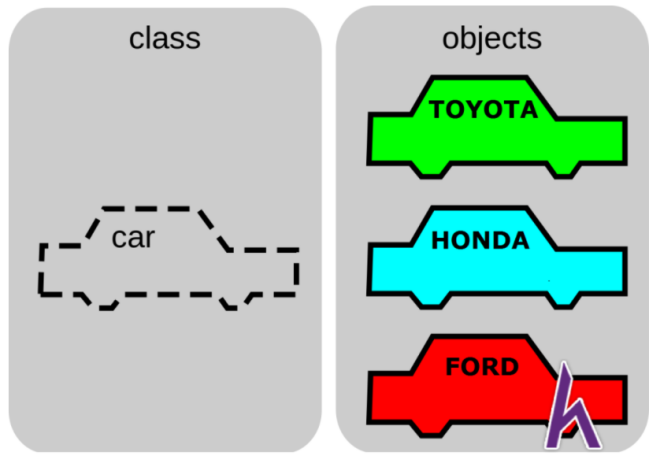
* 1. ***Object (Đối tượng)***
* Trong lập trình hướng đối tượng, đối tượng là một thực thể cụ thể, tồn tại trong hệ thống.

***Ví dụ:*** Chiếc xe Mercedes-Benz trong cửa hàng, màu đen, giá 200000$

- Mỗi đối tượng bao gồm 2 thông tin: ***thuộc tính*** & ***phương thức***

* Thuộc tính chính là những thông tin, đặc điểm của đối tượng. Thuộc tính của đối tượng xe hơi:
  + - Nhãn hiệu: Mercedes-Benz
    - Màu sắc: đen
    - Giá bán: 200000$
* Phương thức là những thao tác, hành động mà đối tượng đó có thể thực hiện. Đối tượng xe hơi trong cửa hàng có khả năng thực hiện những hành động sau:
  + - Khởi động.
    - Dừng lại.
    - Chạy.
  1. ***Class (Lớp đối tượng)***
* Trong lập trình hướng đối tượng, lớp là một khái niệm trừu tượng, dùng để chỉ một tập hợp các đối tượng có đặc tính tương tự nhau. **Lớp** là một khuôn mẫu còn **đối tượng** là một thể hiện cụ thể dựa trên khuôn mẫu đó.

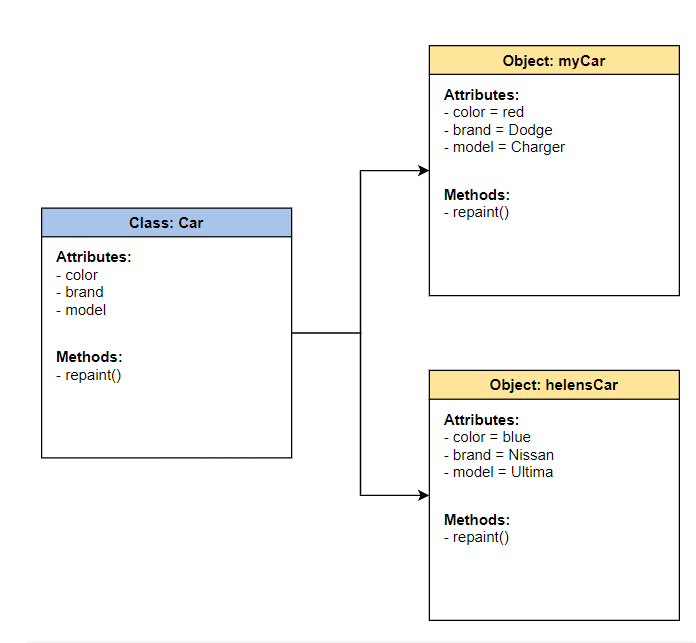
***Ví dụ:*** Xét **Lớp** xe hơi. Có thể có nhiều xe với các tên và thương hiệu khác nhau nhưng tất cả chúng đều có chung một số đặc tính như tất cả chúng đều có 4 bánh, có thiết kế tương tự như hình vẽ. Đối tượng chính là các chiếc xe ô tô cụ thể như Toyota camry, Honda city, Ford ranger.



* Class là một kiểu dữ liệu do người dùng tự định nghĩa, gồm các thành phần:
* Thành phần dữ liệu: là các biến dữ liệu
* Các hàm thành viên: là các hàm được sử dụng để thao tác

với các biến này.

* + - * Các thành phần dữ liệu và các hàm thành viên xác định các thuộc tính và phương thức của các đối tượng trong một lớp.

****

* Các **Lớp** xác định các thuộc tính mà một thể hiện của nó sẽ có, chẳng hạn như màu sắc, nhưng không xác định giá trị của các thuộc tính đó cho một đối tượng cụ thể. Các hàm được định nghĩa trong **Lớp** và thực hiện một số hành động hữu ích cho loại đối tượng cụ thể đó.

***Ví dụ:*** **Lớp Xe** **ô tô** có thể có phương thức sơn lại làm thay đổi thuộc tính màu của xe của chúng ta. Hàm này chỉ hữu ích cho các đối tượng kiểu Car.

* 1. ***Phân biệt Object và Class***

|  |  |
| --- | --- |
| **Class** | **Object** |
| Lớp được sử dụng như một khuôn mẫu để khai báo và tạo các đối tượng. | Một đối tượng là một thể hiện của một lớp |
| Khi một lớp được tạo, không có bộ nhớ nào được cấp phát. | Các đối tượng được cấp phát không gian bộ nhớ bất cứ khi nào chúng được tạo. |
| Lớp chỉ được khai báo một lần. | Một đối tượng được tạo nhiều lần theo yêu cầu. |
| Không thể thao tác một lớp vì chúng không có sẵn trong bộ nhớ. | Các đối tượng có thể được thao tác. |
| Lớp không xác định giá trị của các thuộc tính đó cho một đối tượng cụ thể | Mỗi đối tượng có các giá trị riêng,  gắn liền với nó. |
| Một lớp được sử dụng để liên kết dữ liệu cũng như các phương thức với nhau như một đơn vị duy nhất. | Các đối tượng giống như một biến của lớp. |

* 1. ***Con trỏ this***

Mọi đối tượng trong C ++ đều có quyền truy cập vào địa chỉ của chính nó thông qua một con trỏ quan trọng được gọi là con trỏ **this**. Con trỏ này là một tham số ngầm định cho tất cả các hàm thành viên. Do đó, bên trong một hàm thành viên, **this** có thể được sử dụng để tham chiếu đến đối tượng đang gọi.

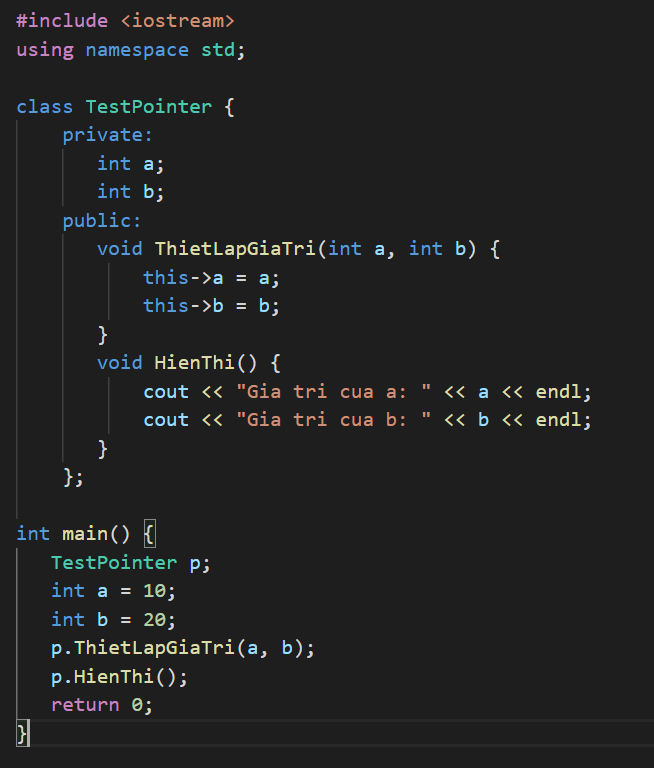
* + - * **This** được dùng để truy cập đến các thuộc tính, phương thức

của đối tượng hiện tại

***Ví dụ:***

Và kết quả sau khi thực thi chương trình bên như sau:





1. **Tính chất của OOP**
   1. ***Tính đóng gói*** ***(Encapsulation):***

Ý nghĩa của **Encapsulation** là đảm bảo rằng dữ liệu quan trọng được ẩn khỏi người dùng. Để đạt được điều này, bạn phải khai báo các biến hoặc thuộc tính của lớp là **private** (không thể truy cập từ bên ngoài lớp). Nếu bạn muốn người khác đọc hoặc sửa đổi giá trị của thành viên **private**, bạn có thể cung cấp các phương thức **get** và **set** công khai

***Ví dụ:*** Truy cập các thành viên riêng tư

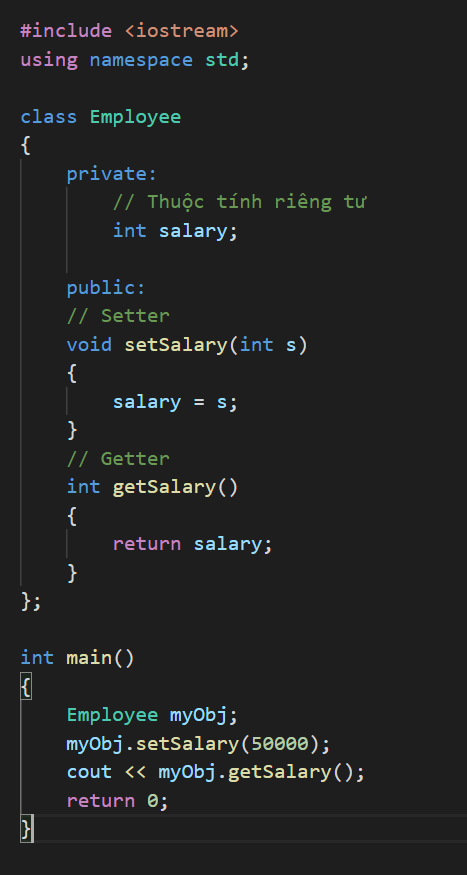
Để truy cập thuộc tính riêng tư, hãy sử dụng các phương thức công khai "get" và "set"

**salary** là thuộc tính riêng tư, có quyền truy cập hạn chế.

**setSalary()** là phương thức công khai nhận một tham số (s) và gán nó cho thuộc tính **salary** (salary = s).

**getSalary()** là phương thức công khai trả về giá trị của thuộc tính riêng tư **salary**

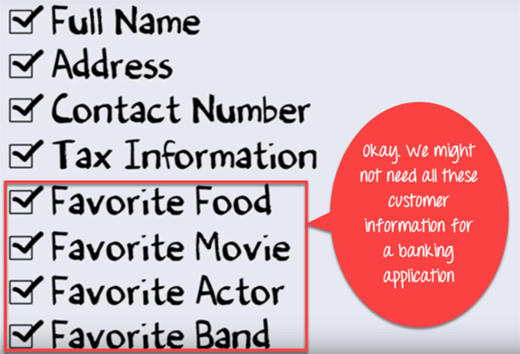
Bên trong **main(),** chúng ta tạo một đối tượng của lớp **Employee**. Bây giờ chúng ta có thể sử dụng phương thức **setSalary()** để đặt giá trị cho thuộc tính private thành 50000. Sau đó, chúng ta gọi phương thức **getSalary()** đối tượng để trả về giá trị.



* + - * ***Nhận xét:***
* Việc khai báo các thuộc tính lớp là private (thường xuyên nếu có thể) được coi là phương pháp hay. Tính năng đóng gói đảm bảo kiểm soát dữ liệu một cách tốt hơn, vì ta có thể thay đổi một phần của mã mà không ảnh hưởng đến các phần khác.
* Tăng tính bảo mật cho dữ liệu
  1. ***Tính trừu tượng (Abstraction):***

Tính trừu tượng có nghĩa là chỉ hiển thị các thuộc tính thiết yếu và ẩn thông tin không cần thiết của đối tượng. Tính trừu tượng là chọn dữ liệu từ một nhóm lớn hơn để chỉ hiển thị các chi tiết có liên quan của đối tượng cho người dùng.

***Ví dụ:*** Giả sử bạn muốn tạo một ứng dụng ngân hàng và bạn được yêu cầu thu thập tất cả thông tin về khách hàng của mình. Có khả năng bạn sẽ đưa ra thông tin sau về khách hàng



Tuy nhiên, bạn chỉ cần chọn thông tin hữu ích cho ứng dụng ngân hàng của mình từ nhóm đó. Dữ liệu như tên, địa chỉ, thông tin thuế, v.v. có ý nghĩa đối với ứng dụng ngân hàng. Vì ta đã tìm nạp / xóa / chọn thông tin khách hàng từ một nhóm lớn hơn, nên quá trình này được gọi là Abstraction trong OOP.



* 1. ***Tính kế thừa***

Kế thừa, tái sử dụng phương thức, thuộc tính của lớp cơ sở và lớp kế thừa được gọi là lớp con, nó sẽ thừa hưởng những gì lớp cha có và cho phép. Có nghĩa là nếu lớp A kế thừa lớp B thì lớp A sẽ có những thuộc tính và phương thức của lớp B.

***Ví dụ:*** Quản lí trường đại học, với hai lớp Nhân viên và lớp Sinh viên, đều kế thừa từ lớp Người

|  |  |
| --- | --- |
| **Lớp Người** ***Thuộc tính*** Tên Ngày sinh Giới tính ***Phương thức*** Nhập/xem tên Nhập/xem ngày sinh Nhập/xem giới tính | |
| **Lớp Nhân viên kế thừa**  **từ Lớp Người**   |  | | --- | | ***Thuộc tính*** | | Lương | | ***Phương thức*** | | Nhập/xem lương | | |  | | --- | | **Lớp Nhân viên kế thừa**  **từ Lớp Người** | | ***Thuộc tính*** | | Lớp | | ***Phương thức*** | | Nhập/xem lớp | |

* 1. ***Tính đa hình***

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

***Ví dụ:*** Quản lí trường đại học, với hai lớp Nhân viên và lớp Sinh viên, đều kế thừa từ lớp Người. Khi đó, ta thêm vào mỗi lớp một phương thức show():

• Phương thức show của lớp Người sẽ giới thiệu tên và tuổi của người đó.

• Phương thức show của lớp Nhân viên sẽ giới thiệu nhân viên đó có tiền lương là bao nhiêu

• Phương thức show của lớp Sinh viên sẽ giới thiệu là sinh viên đó đang học ở lớp nào.

🡺Việc chỉ cần gọi cùng một phương thức, nhưng từ các đối tượng khác nhau, sẽ cho kết quả khác nhau được gọi là tính đa hình trong lập trình hướng đối tượng.

1. **Access Modifiers (Nhãn phạm vi)**

Access modifier trong C++ được sử dụng để triển khai một tính năng quan trọng của Lập trình hướng đối tượng được gọi là Data Hiding. Access modifier hay Access Specifier trong một lớp được sử dụng để đặt khả năng truy cập của các thành viên lớp. Đó là, nó đặt ra một số hạn chế đối với các thành viên lớp không được truy cập trực tiếp bởi các hàm bên ngoài.

Trong C++ có hổ trợ 3 loại access modifier đó là:

* Private - các thành viên không thể được truy cập (hoặc xem) từ bên ngoài lớp
* Public - các thành viên có thể truy cập từ bên ngoài lớp
* Protected - các thành viên không thể được truy cập từ bên

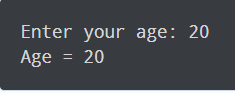
ngoài lớp, tuy nhiên, chúng có thể được truy cập trong các lớp kế thừa.

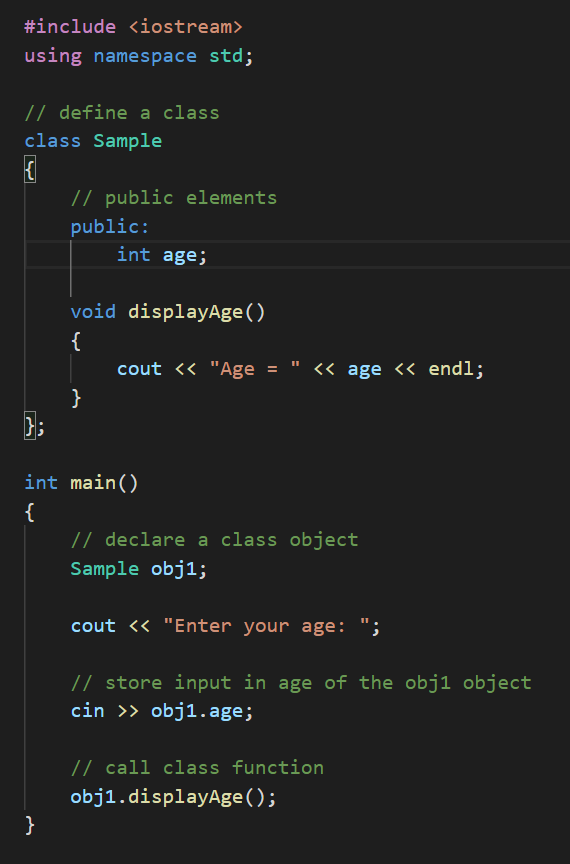
***Lưu ý:*** Nếu chúng ta không chỉ rõ bất kỳ access modifier cho thành viên của lớp, thì mặc định nó là Private.

* 1. **Public access modifier**

Trong chương trình này, chúng ta đã tạo một lớp có tên là **Sample**, chứa một biến **age** là thuộc tính **public** và một hàm **public** là **displayAge()**. Trong main(), chúng ta đã tạo một đối tượng của lớp **Sample** có tên là **obj1**. Sau đó, ta truy cập trực tiếp các phần tử **public** bằng cách sử dụng mã **obj1.age** và **obj1.displayAge ().**

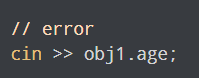
***Output:***





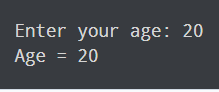
* 1. **Private access modifier**

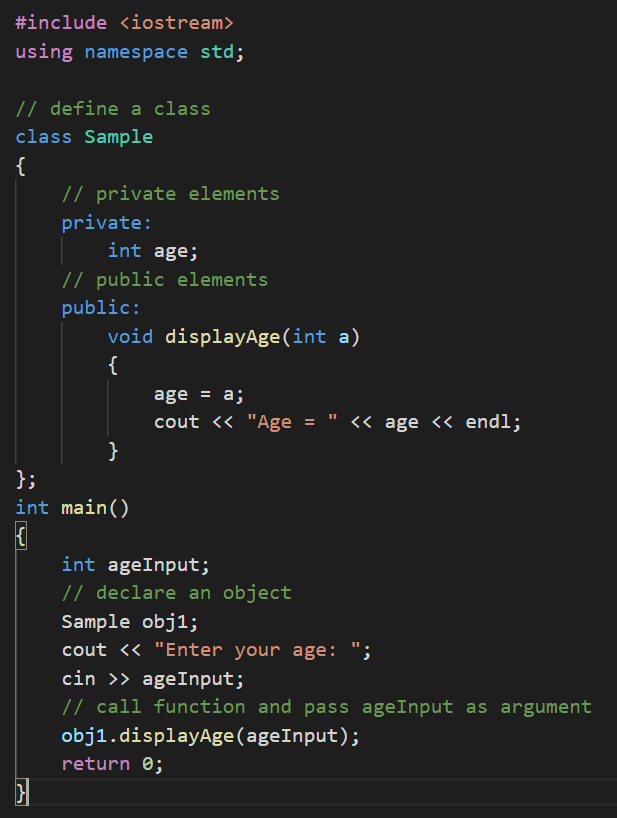
Trong main(), đối tượng **obj1** không thể truy cập trực tiếp vào biến lớp **age**.



Chúng ta chỉ có thể gián tiếp thao tác với **age** thông qua hàm công khai displayAge(), vì hàm này khởi tạo **age** với giá trị của đối số được truyền cho nó tức là tham số a.

***Output:***





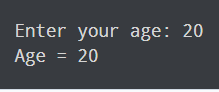
* 1. **Protected access modifier**

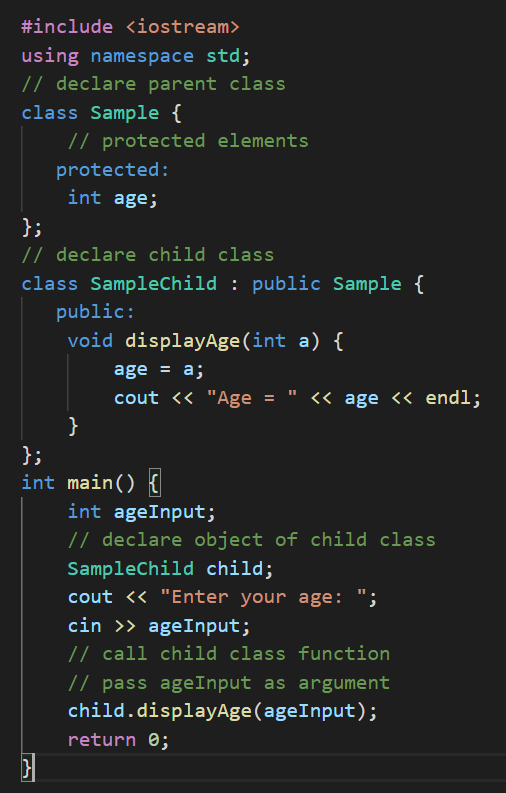
Ở đây, **SampleChild** là một lớp kế thừa có nguồn gốc từ **Sample**. Biến **age** được khai báo trong **Sample** bằng từ khoá **protected.**

Điều này có nghĩa là **SampleChild** có thể truy cập **age** bởi vì **Sample** là lớp cha của nó.

Ta thấy điều này vì chúng tôi đã gán giá trị **ageInput** (tham số a) cho thuộc tính **age** trong **SampleChild** mặc dù **age** được khai báo trong lớp **Sample**.

***Output:***





1. **Constructor – Destructor**

Phương thức khởi tạo (Constructor) hay phương thức hủy (Destructor) là 2 trong số các phương thức mang lại sự tiện lợi khi lập trình với class.

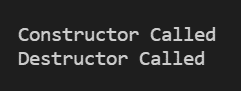
|  |  |
| --- | --- |
| **Constructor** | **Destructor** |
| * Đặt trùng tên với tên class * Không có kiểu trả về | |
| Phương thức tự động được gọi sau khi đối tượng được tạo xong. | Phương thức tự động được gọi trước khi đối tượng tiến hành hủy. |
| Hàm được định nghĩa bằng cách đặt tên trùng với tên class | Hàm được định nghĩa bằng cách đặt trùng với tên class và thêm ký tự ~ vào phía trước |
| Có thể có nhiều constructor bên trong 1 lớp. | Chỉ có 1 destructor duy nhất trong 1 lớp. |

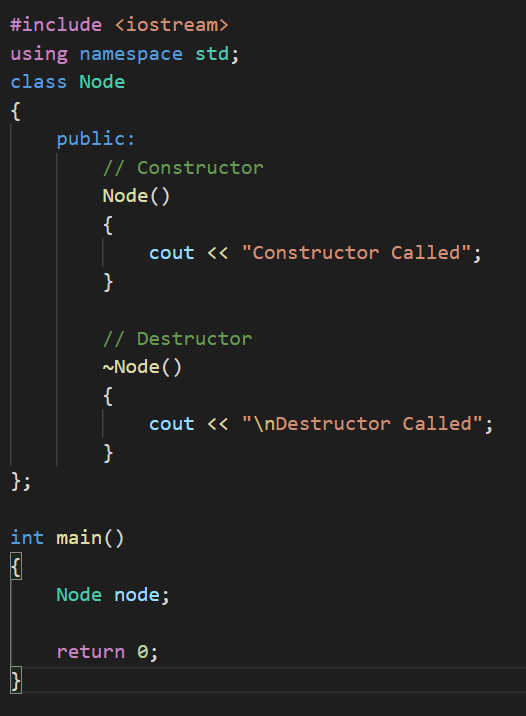
***Ví dụ:***

Sau dòng Node node;

* Đối tượng node được tạo ra.
* Sau đó Node() tự động được gọi.
* Đến giai đoạn kết thúc hàm main(), đối tượng node sẽ được thu hồi nhưng trước đó sẽ tự động gọi ~Node().

***Output:***

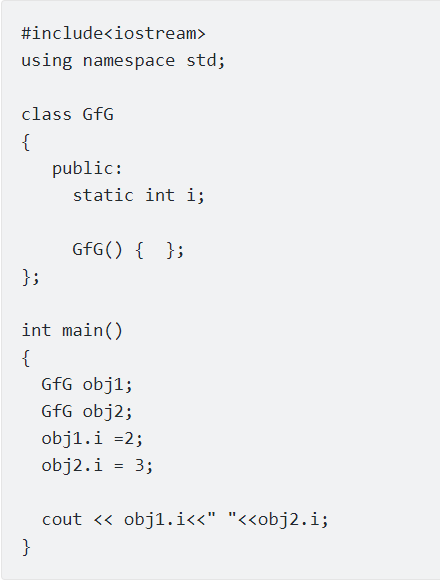




1. **Biến static trong class**

Vì các biến được khai báo là tĩnh chỉ được khởi tạo một lần khi chúng được cấp phát một địa chỉ trong bộ lưu trữ tĩnh riêng biệt, do đó, các biến tĩnh trong một lớp được chia sẻ bởi các đối tượng. Chúng ta không tạo ra các bản sao cho cùng một biến tĩnh của các đối tượng khác nhau. Cũng vì lý do này mà các biến tĩnh không thể được khởi tạo bằng cách sử dụng các hàm khởi tạo (hàm constructor()).

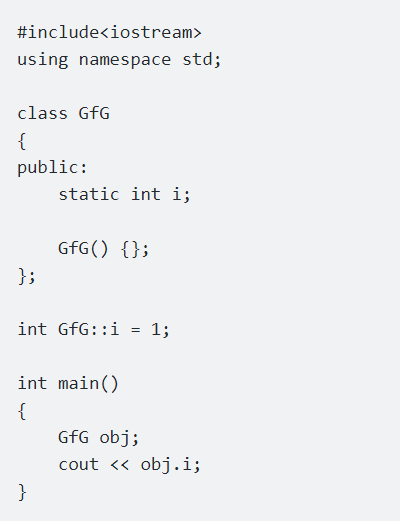
* Bạn có thể thấy trong chương trình bên, khi thử tạo ra nhiều bản sao của biến static i của các đối tượng thì chương trình lỗi, không chạy được.



* Vì vậy, một biến static bên trong một class nên được khởi tạo bằng cách sử dụng toán tử tên và toán tử phân giải phạm vi bên ngoài class như bên

***Output:***

****



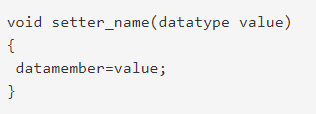
1. **Setter & Getter**

Setter và Getter là 2 phương thức sử dụng để cập nhật hoặc lấy ra giá trị thuộc tính, đặc biệt dành cho các thuộc tính ở phạm vi private.

***Syntax:***

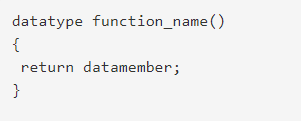
* 1. **Setter**

Setter thường được gọi là hàm đột biến. Đây là một hàm đặc biệt cho phép chúng ta thiết lập (gán) giá trị cho một thành viên dữ liệu của một đối tượng của một lớp. Nó cũng có thể thay đổi giá trị trước đó được gán cho một thành viên dữ liệu. Tuy nhiên, các thành viên dữ liệu của một đối tượng của lớp thường được gán giá trị bằng cách tạo một hàm đặc biệt gọi là hàm tạo. Cú pháp của nó là:



* 1. **Getter**

Getter thường được gọi là chức năng truy cập. Getter là một chức năng đặc biệt cho phép chúng ta truy cập vào các thành viên dữ liệu của lớp (kể cả các thành viên private và protected). Cú pháp của nó là:



1. **Kỹ thuật chia tách file (Cái này khó quá ạ)**
   1. **File.h**
   2. **File.cpp**
2. **Quy tắc đặt tên biến trong OOP C++**

* Tên biến, function viết hoa chữ cái đầu nhưng trừ chữ đầu tiên. Ví dụ: duongDinh24, hoVaTen, \_isPassWord . . .
* Tên lớp phải bắt đầu bằng chữ in hoa.
* Để chuyên nghiệp thì tên variable, function, class . . . nên đặt bằng tiếng anh (không nhất thiết nhưng thực ra chúng ta nên làm quen dần với điều này)
* Đặt tên biến dễ gợi nhớ, dễ cho cả người lập trình và người đọc code. Cẩn thận thì thêm chú thích vào cho chắc nhé!
* Tên hằng số thì nên viết hoa hết
* Các từ cách nhau bằng dấu gạch \_
* Tên đặt phải làm rõ nghĩa, tránh bị sai lạc, hiểu sai vấn đề.
* Dùng một cái tên dễ dàng đọc, có thể đọc được và tuyệt đối không được sai chính tả nhé!
* Tên biến, hàm, lớp nên đặt một cách dễ tìm kiếm, dễ nhớ. Điều này rất thuận lợi với chúng ta khi code