C++ Training Course

Basic about OOP

Lesson Objectives

- Understand about characteristics of OOP
- Understand about Class
- Understand about Data encapsulation
- Understand about Access modifiers
- Understand about Constructors, Copy constructor, Destructors

phân biệt tính trừu tượng và tính đóng gói:
VD: + class cơ sở : động vật có vú có các đặc tính như: có tứ chi, máu nóng, tim 4 ngăn, đẻ con và nuôi con bằng sữa,...
+ các class con của class động vật có vú đều có đặc tính này và có 1 thêm 1 số đặc tính riêng của loài như màu lông, lượng lông, cân nặng trung bình... đó là tính kế thừa
+ trong class cơ sở có các đặc tính như tuổi, chiều cao, cân nặng, ...và các phương thức như đi, kêu, ... được bảo vệ đó là tính đóng gói
+ các class khác nhau có thể có kiểu kêu khác nhau (overloader hoặc virtual) đó là tính đa hình
+ khi nhắc đến các đặc tính tứ chi, máu nóng, tim 4 ngăn, đẻ con và nuôi con bằng sữa ta sẽ nghĩ đến động vật có vú đó như là 1 cách để trừu tượng hóa động vật có vú - tính trừu tượng

Section 1

Characteristics of OOP

tính đa hình:

Characteristics of OOP

- Encapsulation is capturing data and keeping it safely and securely from outside interfaces Đóng gói (Encapsulation): Đóng gói là việc bao gồm dữ liệu và giữ nó an toàn và bảo mật khỏi các giao diện bên ngoài.
- Inheritance: This is the process by which a class can be derived from a base class with all features of base class and some of its own. This increases code reusability Kế thừa (Inheritance): Kế thừa cho phép một lớp con có thể được dẫn xuất từ một lớp cơ sở với tất cả các tính năng của lớp cơ sở và một số tính năng riêng của nó. Điều này tăng tính tái sử dụng mã nguồn.
- Polymorphism: This is the ability to exist in various forms. For example an operator can be overloaded so as to add two integer numbers and two floats. Đa hình (Polymorphism): Đa hình là khả năng tồn tại dưới nhiều hình thức khác nhau. Ví dụ, một toán tử có thể được nạp chồng để thực hiện phép cộng hai số nguyên hoặc hai số thực.
- Abstraction- The ability to represent data at a very conceptual level without any details. Trừu tượng (Abstraction): Trừu tượng cho phép biểu diễn dữ liệu ở một mức khái niệm mà không cần chi tiết cụ thể.

Section 2

Class

Agenda

- What is Class
- What is object
- How to declare a class
- How to implement class

What is Class

mở rộng khái niệm

- Class is an expanded concept of data structures:

like data structures, they can contain data members (also called properties/attributes), but they can also contain functions as members (also called methods/activities

thuộc tính / đặc điểm

phương thức/ hành động

class = data + functions

Lớp là một khái niệm mở rộng của cấu trúc dữ liệu:
giống như cấu trúc dữ liệu, chúng có thể chứa các thành viên dữ liệu (còn được gọi là thuộc tính/đặc điểm),
nhưng chúng cũng có thể chứa các hàm như là thành viên (còn được gọi là phương thức/hoạt động)

What is Class

- Properties:
  - Name
  - Age
  - Color
  - Weight
- Activities:
  - Eat
  - Sleep
  - Run
  - Bark

```
class Dog {
public:
        void eat();
        void sleep();
        void run();
        void bark();
private:
        string mName;
        int mAge;
        string  mColor;
        int mWeight;
};
```

What is Object đối tượng

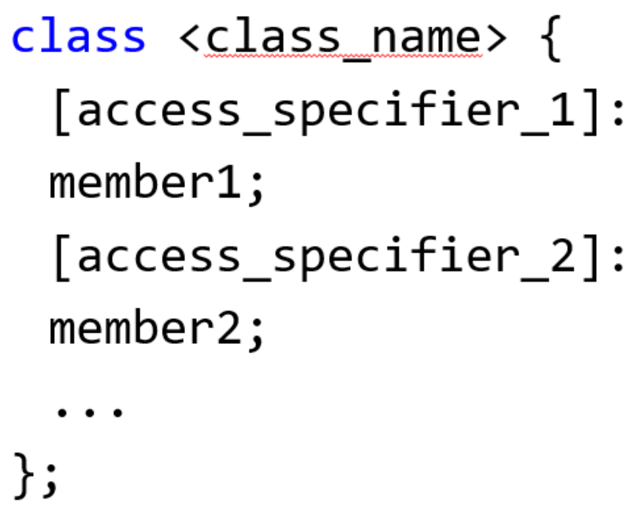
thể hiện

- An Object is an instantiation of a class ==> That means a class is the a data type, and an Object is a variable of this type

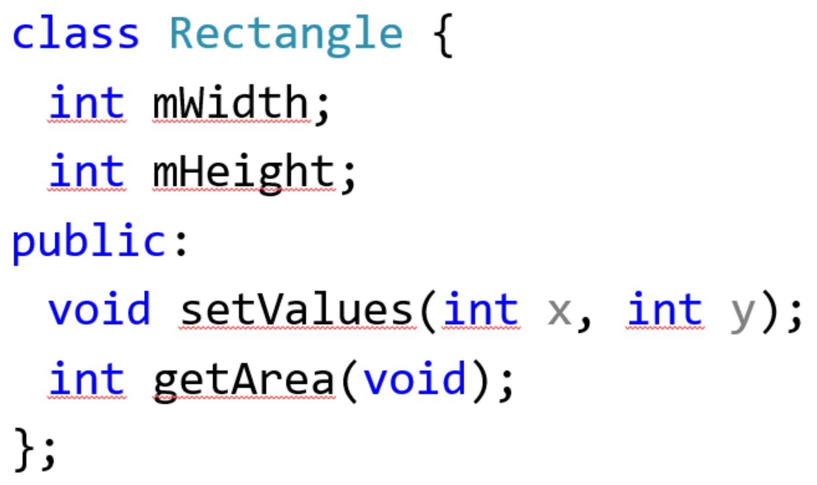
```
Dog dog;  // dog is an object
```

Đối tượng là một thể hiện của một lớp ==> Điều này có nghĩa là một lớp là một kiểu dữ liệu, và một đối tượng là một biến của kiểu dữ liệu này.

Declare class



```
class <class_name> {
  [access_specifier_1]:
  member1;
  [access_specifier_2]:
  member2;
  ...
};
```



```
class Rectangle {
  int mWidth;
  int mHeight;
public:
  void setValues(int x, int y);
  int getArea(void);
};
```

Implement class

```
void Rectangle::setValues(int x, int y) {
  mWidth = x;
  mHeight = y;
}

int Rectangle::getArea() {
  return mWidth*mHeight;
}
```

Implement class

Rectangle.h

```
class Rectangle {
   int mWidth;
   int mHeight;

public:
void setValues(int, int);
   int getArea(void);

};
```

Rectangle.cpp

```
#include "Rectangle.h"

void Rectangle::setValues(int x, int y)
{
        mWidth = x;
        mHeight = y;
}

int getArea() {
        return mWidth*mHeight;
}
```

main.cpp

```
#include "Rectangle.h"
#include <iostream>
using namespace std;

int main() {
  Rectangle rect;
  rect.setValue(3, 4);
  cout << "area = " << rect.getArea();
  return 0;
}
```

Section 3

Data encapsulation

Agenda

- What is Encapsulation?
- Benefit?

+ Đóng gói dữ liệu (cũng được gọi là ẩn thông tin) là quá trình giữ kín các chi tiết về cách một đối tượng được triển khai khỏi người dùng của đối tượng.
+ Người dùng của đối tượng truy cập đối tượng thông qua một giao diện công khai. Bằng cách này, người dùng có thể sử dụng đối tượng mà không cần phải hiểu cách nó được triển khai.
+ Tất cả các biến thành viên của lớp được đặt ở chế độ riêng tư (ẩn các chi tiết triển khai), và hầu hết các hàm thành viên được đặt ở chế độ công khai (cung cấp một giao diện cho người dùng).

What is Encapsulation đóng gói

ẩn thông tin

- **Encapsulation** (also called **information hiding**) is the process of keeping the details about how an object is implemented hidden away from users of the object.
- Users of the object access the object through a public interface. In this way, users are able to use the object without having to understand how it is implemented. triển khai
- All member variables of the class are made private (hiding the implementation details), and most member functions are made public (exposing an interface for the user)

What is Encapsulation

- All member variables of the class are made private (hiding the implementation details), and most member functions are made public (exposing an interface for the user)

```
class Rectangle {

private:
    int mWidth;
    int mHeight;

public:
    void setValues(int, int);
    int getArea(void);

};
```

Benefit

- Encapsulated classes are easier to use and reduce the complexity of programs

giảm

Các lớp được đóng gói dễ sử dụng hơn và giảm độ phức tạp của chương trình.

tính phức tạp

- Encapsulated classes help protect your data and prevent misuse

bảo vệ

sử dụng sai Các lớp được đóng gói giúp bảo vệ dữ liệu của bạn và ngăn chặn việc sử dụng sai.

- Encapsulated classes are easier to change Các lớp được đóng gói dễ thay đổi hơn.

- Encapsulated classes are easier to debug Các lớp được đóng gói dễ gỡ lỗi hơn.

Section 4

Access modifiers

Agenda

- Public Access Modifier
- Private Access Modifier
- Protected Access Modifier

Public Access Modifier

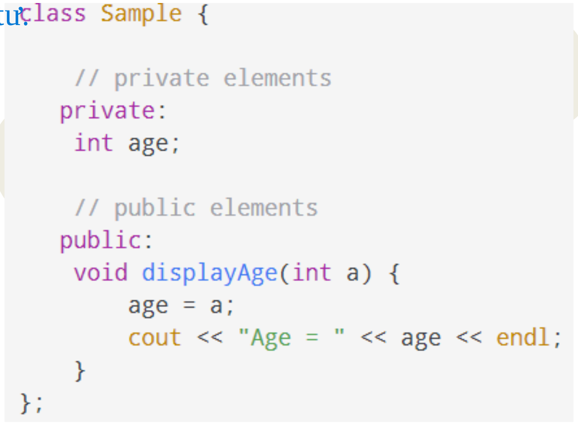
- The public keyword is used to create public members (data and functions). Từ khóa public được sử dụng để tạo các thành viên công khai (dữ liệu và hàm)
  Các thành viên công khai có thể được truy cập từ bất kỳ phần nào của chương trình.
- The public members are accessible from any part of the program.

```cpp
#include <iostream>
using namespace std;

// define a class
class Sample {

    // public elements
   public:
    int age;

    void displayAge() {
        cout << "Age = " << age << endl;
    }
};
```

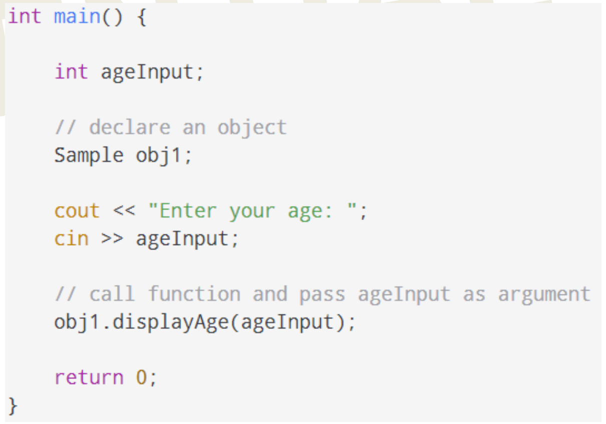
```cpp
int main() {

    // declare a class object
    Sample obj1;

    cout << "Enter your age: ";

    // store input in age of the obj1 object
    cin >> obj1.age;

    // call class function
    obj1.displayAge();

    return 0;
}
```

Private Access Modifier

- The private keyword is used to create private members (data and functions). Từ khóa private được sử dụng để tạo các thành viên riêng tư (dữ liệu và hàm).
- The private members can be accessed only from within the class. Các thành viên riêng tư chỉ có thể được truy cập từ bên trong lớp.
- However, friend classes and friend functions can access private members Tuy nhiên, các lớp bạn (friend classes) và các hàm bạn (friend functions) có thể truy cập các thành viên riêng tư.



```cpp
class Sample {

   // private elements
  private:
   int age;

   // public elements
  public:
   void displayAge(int a) {
       age = a;
       cout << "Age = " << age << endl;
   }
};
```



```cpp
int main() {

    int ageInput;

    // declare an object
    Sample obj1;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call function and pass ageInput as argument
    obj1.displayAge(ageInput);

    return 0;
}
```

Protected Access Modifier

- The protected keyword is used to create protected members (data and function). Từ khóa protected được sử dụng để tạo các thành viên được bảo vệ (dữ liệu và hàm).
- The protected members can be accessed within the class and from the derived class. Các thành viên được bảo vệ có thể được truy cập từ bên trong lớp và từ các lớp dẫn xuất.

```cpp
#include <iostream>
using namespace std;

// declare parent class
class Sample {
    // protected elements
   protected:
    int age;
};
```

```cpp
// declare child class
class SampleChild : public Sample {

   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }

};
```

```cpp
int main() {
    int ageInput;

    // declare object of child class
    SampleChild child;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call child class function
    // pass ageInput as argument
    child.displayAge(ageInput);

    return 0;
}
```

Các loại kế thừa :
Private : không cho con sử dụng, tiết lộ ra ngoài (trọng số 3)
Protected: Cho con sử dụng, không lộ ra ngoài (trọng số 2)
Public: cho con sử dụng, cho tiết lộ ra ngoài (trọng số 1)
Xét theo trọng số:

- 3+1, 3+2, 3+3: con không được thừa hưởng, sử dụng phần private của cha cả ở trong lẫn ngoài class.
- 2+1, 2+2, 2+3: được sử dụng ở trong, không sử dụng ở ngoài , có thể thay đổi, tuy cùng tên nhưng trở thành giao thức của con.
- 1+3, 1+2: được sử dụng ở trong, không sử dụng ở ngoài, có thể thay đổi, tuy cùng tên nhưng trở thành giao thức của con.
- 1+1, được sử dụng ở trong, được sử dụng ở ngoài, có thể thay đổi luôn, , tuy cùng tên nhưng trở thành giao thức của con.

Section 4

Constructors, Copy constructor, Destructors

Agenda

- Constructors
- Copy constructor
- Destructors

Destructor trong C++ được gọi mặc định trong các trường hợp sau:
1. Khi đối tượng ra khỏi phạm vi: biến local khi ra khỏi phạm vi
2. Khi hủy một đối tượng:
+ hủy một đối tượng con qua con trỏ lớp cha:
+ xóa một đối tượng được cấp phát động
+ hủy một đối tượng trong container (ví dụ: vector)
+ khi một đối tượng của lớp con được khởi tạo và sau đó bị ép kiểu dữ liệu thành lớp cha, phần của đối tượng lớp con mà lớp cha không có sẽ bị cắt đi (sliced), dẫn đến việc không thể truy cập được nữa.

```
 // Ep kieu du lieu tu Con thanh Cha
Cha cha = Con(5, 10); // Constructor cua Con duoc goi, sau do ep kieu thanh Cha => khi đó quá trình slicing xảy ra, chỉ lấy phần Cha của đối tượng Con
// khi đối tượng con được khởi tạo nó được cấp phát bộ nhớ tuy nhiên ngay sau đó sẽ bị clicing nên sẽ diễn ra Destructor ngay khi đó
 cha.hienThi(); // Goi phuong thuc hienThi() cua lop Cha
// ở cuối chương trình thêm 1 destructor nữa được gọi đó là của đối tượng cha
```

```
// Ép kiểu con trỏ từ Con* thành Cha*
 Con con(5, 10);
Cha* pCha = static_cast<Cha*>(&con); // Ép kiểu con trỏ an toàn từ Con* thành Cha*, khi này không xảy ra hiện tượng slicing

// Gọi phương thức hienThi() của lớp Con thông qua con trỏ lớp Cha nhờ tính đa hình
 pCha->hienThi();
```

Constructors

- A constructor is a special kind of class member function that is automatically called when an object of that class is instantiated. Constructors are typically used to initialize member variables of the class to appropriate default or user-provided values, or to do any setup steps necessary for the class to be used (e.g. open a file or database).

khởi tạo

biến thành viên

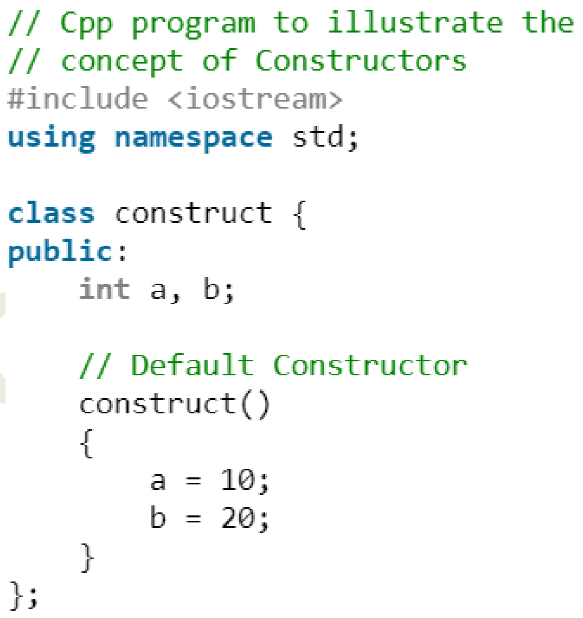
- Unlike normal member functions, constructors have specific rules for how they must be named
  - ✓ Constructors must have the same name as the class (with the same capitalization)
  - ✓ Constructors have no return type (not even void)

+ Một constructor là một loại hàm thành viên đặc biệt trong lớp được gọi tự động khi một đối tượng của lớp đó được khởi tạo.
+ Thường được sử dụng để khởi tạo các biến thành viên của lớp với các giá trị mặc định hoặc do người dùng cung cấp, hoặc để thực hiện bất kỳ bước thiết lập nào cần thiết để lớp có thể sử dụng (ví dụ: mở một tập tin hoặc cơ sở dữ liệu).
+ Khác với các hàm thành viên thông thường, các constructor có các quy tắc cụ thể về cách đặt tên:
 . Constructor phải có cùng tên với tên của lớp (với cùng cách viết hoa chữ cái).
 . Constructor không có kiểu trả về (thậm chí cả void cũng không có).

Default constructors

- A constructor that takes no parameters is called a default constructor

một constructor không có tham số được gọi là default constructor

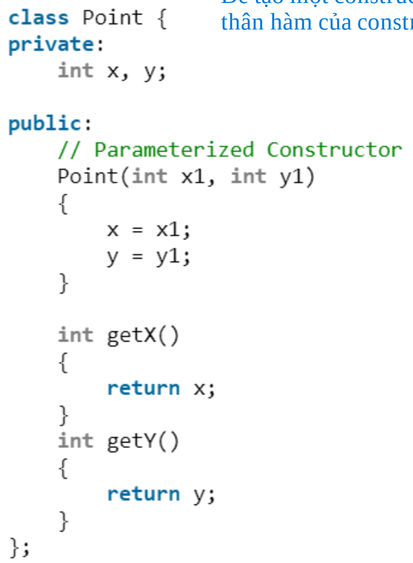


```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;

class construct {
public:
    int a, b;

    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};
```

Parameterized Constructors

- To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object

Để tạo một constructor có tham số, bạn chỉ cần thêm các tham số vào như bạn làm với bất kỳ hàm nào khác. Khi bạn định nghĩa thân hàm của constructor, sử dụng các tham số để khởi tạo đối tượng.



```
class Point {
private:
    int x, y;

public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }

    int getX()
    {
        return x;
    }
    int getY()
    {
        return y;
    }
};
```

Fpt Software | CAMPUS

Copy constructor

- A copy constructor is a special type of constructor used to create a new object as a copy of an existing object A copy constructor là một loại đặc biệt của constructor được sử dụng để tạo một đối tượng mới như một bản sao của một đối tượng hiện có.
- if we do not provide a copy constructor for your classes, C++ will create a public copy constructor.

```cpp
#include<iostream>
using namespace std;

class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p2) {x = p2.x; y = p2.y; }

    int getX()            {  return x; }
    int getY()            {  return y; }
};
```

```cpp
int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```

không viết thì C++ có thể tự sinh ra, tuy nhiên khi khởi tạo thì có thể custom lại theo yêu cầu

PHẢI VIẾT LẠI NẾU NHƯ CÓ ĐẶC TÍNH LÀ CON TRỎ HOẶC CẤP PHÁT ĐỘNG ĐỂ TRÁNH TỰ SINH RA SHALLOW COPY (DAY 8)

Destructors

không viết thì C++ có thể tự sinh ra, tuy nhiên khi khởi tạo thì có thể custom lại theo yêu cầu (để hủy cấp phát động, ...)

- A destructor is another special kind of class member function that is executed when an object of that class is destroyed

  A destructor là một loại đặc biệt khác của hàm thành viên lớp được thực thi khi một đối tượng của lớp đó bị hủy.

- Like constructors, destructors have specific naming rules:

  Giống như constructors, destructors có các quy tắc đặt tên cụ thể:

  ✓ The destructor must have the same name as the class, preceded by a tilde (~).

  Giống như constructors, destructors có các quy tắc đặt tên cụ thể:

  ✓ The destructor can not take arguments.

  Destructor không thể có tham số.

  ✓ The destructor has no return type.

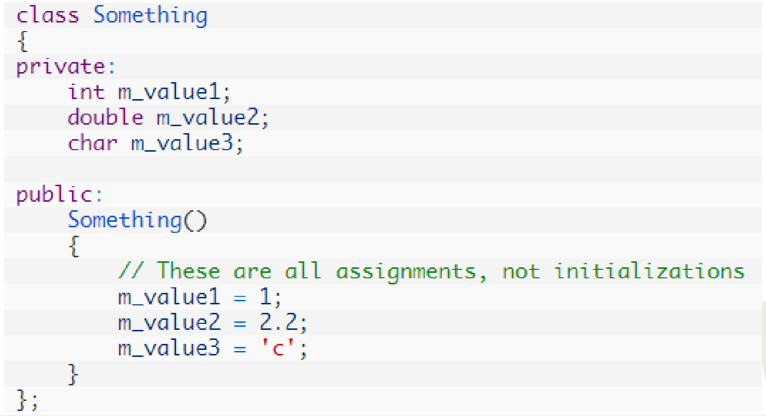
  Destructor không có kiểu trả về.

SOLID in OOP

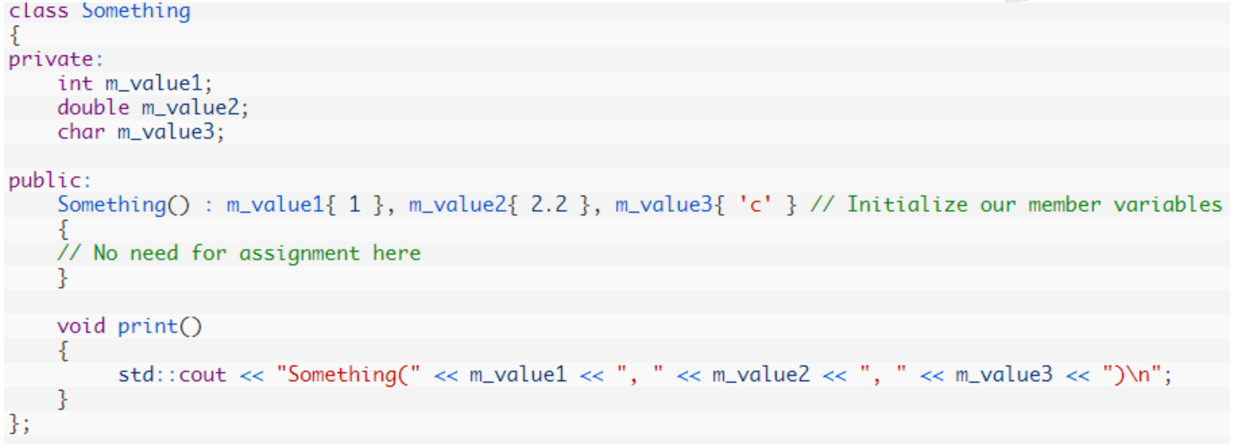
```
class Rectangle {
private:
      int mWidth;
      int mHeight;
public:
      Rectangle(); // This is the default constructor
      ~Rectangle(); // This is the destructor
      void setValues(int, int);
      int getArea(void);
};
Rectangle::~Rectangle() {
      cout << "Object is being deleted" << endl;
}
```

Member initializer

- What is different?



```cpp
class Something
{
private:
    int m_value1;
    double m_value2;
    char m_value3;

public:
    Something()
    {
        // These are all assignments, not initializations
        m_value1 = 1;
        m_value2 = 2.2;
        m_value3 = 'c';
    }
};
```



```cpp
class Something
{
private:
    int m_value1;
    double m_value2;
    char m_value3;

public:
    Something() : m_value1{ 1 }, m_value2{ 2.2 }, m_value3{ 'c' } // Initialize our member variables
    {
    // No need for assignment here
    }

    void print()
    {
        std::cout << "Something(" << m_value1 << ", " << m_value2 << ", " << m_value3 << ")\n";
    }
};
```

References

- https://www.tutorialspoint.com
- https://www.learncpp.com/

Lesson Summary

- 4 characteristics of OOP
- What is Class and Object
- Data encapsulation
- Public, Private, Protected
- Constructors, Copy constructor, Destructors

Thank you