18 - CLASSES in C++

sexta-feira, 7 de fevereiro de 2025 07:01

- OOP is a styla on how to write your code
- C++ doesn't mipl certain impies but support it
- way to grudata and functionalites together
- Variables made of class are called object variables
  - And a new object is na instance of that class
- Defining a class we define the visibility of the variables and functions
  - By default the visibility is private, need to specif as public to acces or protected
- Fuctions inside classes are called methods
- USEFULL TO GROUP THINGS TOGETHER AND ADD FUNCTIONALITIES TO THE OBJECT

CLASSES in C++

CLASSES vs STRUCTS in C++

- Kind a similar one
- there is no much difference
- the main diference is the visibility options in structures (private, public, protected
  - Class is private by default
  - struct the default is public
- But this is tecnicly, but the use in code may differ
- struct exists by bacward compatibility with previous versions
  - the ompiler wouldn't know wht it was in old codes
- The usage differs
  - That is no right or wrong answer, differ by opinion
- struct used just to represent variables
- Never use a structure with inherence, go to classes

How to Write a C++ Class

- Log class to manage the log messages, used for debug process
- console is like na information dump
- Defined simple functions, member variables (public and private)
- Instantiated in main and also used the public functions

Static in C++

- 2 meanins,
  - outside of a class
    - Linkage of that symbel will be internal, only visible to that transation unit that you are working with (translation unit = file)
  - Inside of a class
    - All instances of that class will share the same memory, will only be one instance of that static variable across all instances of the class
- Focus on static outside of a class

Static for Classes and Structs in C++

- If used with a variable
  - Only one instance of taht variable across al isntances of that class
  - If one of the entity changes taht variable, it'll affect all other instances
  - Better to update the value by it's class than instance
    - By isntance could cause confusion and bugs
- Static method
  - Don't have access to the class instance
  - call without a class instance
  - canno write code that refer to a class instance

```
struct StaticEntity22
{
    static int x22, y22;

    void Print(){
        std::cout << "Entity 22 x22 " << x22 << " Y22 " << y22 << std::endl;
    }
};

// When making this variables static, we need ot initialize
// them without any instantiation
int StaticEntity22::x22;
```

```
95    StaticEntity22 se22;
96    se22.x22 = 2;
97    se22.y22 = 3;
98    se22.Print();
99
100   // StaticEntity22 se22_2 = {5, 8}; // This would fail for stati
101   StaticEntity22 se22_2; // This would fail for static classes
102   // se22_2.x22 = 5;
103   // se22_2.y22 = 8;
104   se22_2.Print(); // result should be the same as the other insta
105
106   // we can access them by the class and not by the instance
107   // And it'll change its value
108   StaticEntity22::x22 = 5;
109   StaticEntity22::y22 = 8;
110   StaticEntity22::Print(); // 5, 8
111
112   // Not static struct parameters
113   Entity22 e22;
114
115   e22.x22 = 2;
116   e22.y22 = 3;
117   e22.Print();
118
119   Entity22 e22_2 = { 5, 8 };
```

```cpp
// When making this variables static, we need ot initialize
// them without any instantiation
int StaticEntity22::x22;
int StaticEntity22::y22;

int main()
{

    StaticEntity22 se22;
    e22.x22 = 2;
    e22.y22 = 3;
    e22.Print();

    // StaticEntity22 se22_2 = {5, 8}; // This would fail for static classes
    StaticEntity22 se22_2; // This would fail for static classes
    e22.x22 = 5;
    e22.y22 = 8;
    e22.Print();
```

```cpp
    e22.y22 = 3;
    e22.Print();

    Entity22 e22_2 = { 5, 8 };

    e22_2.Print();
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Hey
Hey
Hey

Hey
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Static Entity 22 x22 2 Y22 3
Static Entity 22 x22 2 Y22 3
Static Entity 22 x22 5 Y22 8
Entity 22 x22 2 Y22 3
Entity 22 x22 5 Y22 8
```

Can access a non-static variable within a class, t generates na error

```cpp
struct StaticEntity22
{
    // static int x22, y22;
    int x22, y22;

    static void Print(){
        std::cout << "Static Entity 22 x22 " << x22 << " Y22 " << y22 << std::endl;
    }
};
```

Constructors in C++

- Special type of method that runs each time we instantiate na object
- When we instantiate a class without initializing the parameters, there is no actual value and they would receive garbage
- To declare it, there is no return type and needs to match the name of the class
  - Can ptionally give parameters
- Has to manually initialize the primitive values, otherwise i'll get garbages in c++
  - Other languages may have different behaviours
- We can write as much constructors as we want, but with different parameters to have diferent signatures
- can defien class with static propertis and methods, and don't want to instantiate nothing (no constructores
  - <Class Name>() = delete;

Destructors in C++

- evel twin, the destructor kkk
- call every time when destroy na object
  - Usually free and uninitialize and clean memory that will not use anymore
  - If initialized objects with new, te destructor will delete them
- Destroyed in the end of the scope... if in a function, will be destroyed when leaving the function
- Used to delete memory alocation, in the heap for example.... or any other initialization
- But is not very commun

```cpp
void PrintFunction26()
{
    // Object will be created and deleted within this function
    // Deleted when leaving it
    Entity24 e24_6(10, 11);
    e24_6.Print();
}

int main()
{
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Destroyed Entity!
Destroyed Entity!
Destroyed Entity!
root@aee12d748e6b:/src/Dev/HelloWorld/out/build# ./HelloWorld
Created Entity!
Entity 24 x24 10 Y24 11
Destroyed Entity!
```

Inheritance in C++

- Allow us to have iherache of classes that relates with each other
- create subclasses from a parent class
- avoid code duplication
  - put duplicated code into a base class
  - So we don't need to keep implementing that
- polymorfism is the idea of having multiple types of a single type
  - We can use a sub class whenever we want to use the base class
- the subclass always have everything that the base class have
- Used all the time to extend na existing class
  - Separate responsibilities