# 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;
int StaticEntity22::y22;

int main()
{

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

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

    // StaticEntity22 se22_2 = {5, 8}; // This would fail for stati
    StaticEntity22 se22_2; // This would fail for static classes
    // se22_2.x22 = 5;
    // se22_2.y22 = 8;
    se22_2.Print(); // result should be the same as the other insta

    // we can access them by the class and not by the instance
    // And it'll change its value
    StaticEntity22::x22 = 5;
    StaticEntity22::y22 = 8;
    StaticEntity22::Print(); // 5, 8

    // Not static struct parameters
    Entity22 e22;

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

    Entity22 e22_2 = { 5, 8 };

    e22_2.Print();

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Hey
Hey
Hey
```

```cpp
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();
```

```
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
172  void PrintFunction26()
173  {
174      // Object will be created and deleted within this function
175      // Deleted when leaving it
176      Entity24 e24_6(10, 11);
177      e24_6.Print();
178  }
179
180  int main()
181  {
182
183      PrintFunction26();
```

```
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!
Created 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

```cpp
219  class Entity27
220  {
221      public:
222          // generate a v table for this function so if it is overriden,
223          // we can call the proper function
224          virtual std::string GetName() {return "Entity27"; }
225  };
226
227  class EntitySub27 : public Entity27
228  {
229      private:
230          std::string m_Name;
231      public:
232          EntitySub27(const std::string& name)
233          : m_Name(name) {}
234
235          std::string GetName() {return m_Name; }
236  };
237
238
239
240
241  int main()
242  {
243      Entity27* e27 = new Entity27();
244      std::cout << e27->GetName() << std::endl;
245
246      EntitySub27* esub27 = new EntitySub27("Hugo");
247      std::cout << esub27->GetName() << std::endl;
248
249      // things starts to crac, this is actually a player but references the base cl
250      // So we get the base class method output
251      Entity27* e27_2 =esub27;
252      // if not virtual function output is "entity27"
253      // if virtual function output is "hugo"
254      std::cout << e27_2->GetName() << std::endl;
255
256
257      std::cout<< "" << std::endl;
```

## Virtual Functions in C++

- Allow us to override methods in our derived method
- If created a virtual function in class A, we have the option to override them in the base class B
  - To do something else
- Vitual functions introduce something called dynamic dispatch
  - Based a V table for all the virtual functions, so we can match to the correct function we desired
  - If you want to oberride a function, ou need to mock the base function
- If not declared as virtual, the function s associated with the class itself, if we call a method from a base class, the base class behvior will prevail. If virtual is defined, the vtable will determine the correct functon to use based on the object calling and not just the class
- Maybe costly but the impact is minimal, don't worrie

## Interfaces in C++ (Pure Virtual Functions)

- Pure virtual functions

- Define a funtion in the base class that doesn't have na implementation
- force subclasses to actually implement that function
    - No base method definitions, implement in the inhered class is not optional
- Need to define the method as virtual and add a =0 to the end of the declaration, instead of the function body
    - Also, we can't instantiate that class