Separate the interface from the implementation

```
/*
      . Topics:
          . Reasons:
              . Encapsulation
              . Clarity
          . Keeping everything in the same file
          . Separating:
              . module interface file
              . module implementation file


*/
```

Separate the interface from the implementation

```
export class Pixel {

   public:
       Pixel() = default;  //Explicitly defaulted default constructor

       Pixel(uint32_t color, unsigned int x, unsigned int y){...}
       ~Pixel(){...}

       uint32_t get_color() const { return m_color; }
       void set_color(uint32_t color) { this→m_color = color;}

       unsigned int get_x() const {...}
       void set_x(unsigned int x) {...}

       unsigned int get_y() const {...}

       // Add a function to print all the mutable variables
       void print_access_count() const {
           fmt::print("Access count: {}\n", m_access_count);
       }

   private:
       uint32_t m_color{0xFF000000};
       unsigned int m_pos_x{0};
       unsigned int m_pos_y{0};

       mutable unsigned int m_access_count{0}; // Tracks how many times the pixel has been accessed
};
```

The interface

```
export class Pixel {
    public:
        Pixel(uint32_t color, unsigned int x, unsigned int y);

        uint32_t get_color() const;
        void set_color(uint32_t color);

        unsigned int get_x() const;
        void set_x(unsigned int x);

        unsigned int get_y() const;
        void set_y(unsigned int y);

    private:
        uint32_t m_color{0xFF000000};
        unsigned int m_pos_x{0};
        unsigned int m_pos_y{0};
};

//The definitions
Pixel::Pixel(uint32_t color, unsigned int x, unsigned int y){
    m_color = color;
    m_pos_x = x;
    m_pos_y = y;
}
//...
```

module interface file and implementation file

ct4.ixx

```
module;

export module ct4;

namespace ct4{

  export class Pixel{
    // ...
  }


};   //namespace ct4
```

ct4.cpp

```
module;

 module ct4;

namespace ct4{

    //The definitions
    Pixel::Pixel(){
        fmt::print("Pixel created\n");
    }

    Pixel::Pixel(uint32_t color, unsigned int x, unsigned int y){
        m_color = color;
        m_pos_x = x;
        m_pos_y = y;
    }
    // ...


};   //namespace ct4
```

The CMakelists.txt file

ct4.ixx

```
#The root CMakeLists.txt file.
cmake_minimum_required(VERSION 3.28)

set(CMAKE_TOOLCHAIN_FILE "$ENV{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake" CACHE STRING "Vcpkg toolchain file")

#The project name is set here. In this case it's Rocket, but you can change it to whatever you want.
project(Rocket  VERSION 1.0.0 LANGUAGES CXX)

#Require C++23
set(CXX_STANDARD_REQUIRED ON) #Make C++23 a hard requirement
set(CMAKE_CXX_STANDARD 23) # Default C++ standard for targets
set(CMAKE_CXX_SCAN_FOR_MODULES ON)

find_package(fmt CONFIG REQUIRED)

#We're using the project name as the target name, but you can change it to make them different.
#With this setup, the name of the executable will be the same as the project name.
add_executable(${PROJECT_NAME} main.cpp ct4.cpp)

target_sources(${PROJECT_NAME}
        PUBLIC
        FILE_SET CXX_MODULES FILES
        utilities.ixx
        ct4.ixx
)
target_link_libraries(${PROJECT_NAME} PRIVATE fmt::fmt)
```