More on C++20 Modules

```
/*

    . More on modules:
        .#1: Block export

        .#2: Export import

        .#3: Submodules

        .#4 Module interface partitions

        .#5: Visibility and reachability

        .#6: Private module fragment
*/
```

Block export

```
export{

    double add(double a, double b) {
        return a + b;
    }

    void greet(const std::string& name) {
        std::string dest;
        dest = "Hello ";
        dest.append(name);
        fmt::println("{}",dest);
    }
}

export void print_name_length(const char* c_str_name) {
    fmt::println("Length: {}",std::strlen(c_str_name));
}
```

Export Import

point.ixx

```
export module point;

namespace more_on_modules_02{

    export class Point {
    public:
        Point(int x, int y) : m_x(x), m_y(y) {}
        int x() const { return m_x; }
        int y() const { return m_y; }
    private:
        int m_x;
        int m_y;
    };
}
```

line.ixx

```
module;

export module line;

export import point;

namespace more_on_modules_02{

    export class Line {
    public:
        Line(Point start, Point end) : m_start(start), m_end(end) {}
        Point start() const { return m_start; }
        Point end() const { return m_end; }
    private:
        Point m_start;
        Point m_end;
    };
}
```

Export Import

```
more_on_modules_02::Point p1{ 1, 2 };    //We can instantiate Point even if the module is not directly imported.
more_on_modules_02::Point p2{ 3, 4 };
more_on_modules_02::Line line{ p1, p2 };

fmt::println("p1 [{}, {}]", p1.x(), p1.y());
fmt::println("p2 [{}, {}]", p2.x(), p2.y());
```

Sub-modules

add_sub.ixx

```
module; // This module can be imported independently

export module math.add_sub;

namespace more_on_modules_03{
  export{
    double add(double a, double b) {
      return a + b;
    }

    double sub(double a, double b) {
      return a - b;
    }
  }
}  //namespace more_on_modules_03
```

mult_div.ixx

```
module; // This module can be imported independently

export module math.mult_div;

namespace more_on_modules_03{
  export{
    double mult(double a, double b) {
      return a * b;
    }

    double div(double a, double b) {
      return a / b;
    }
  }
}  //namespace more_on_modules_03
```

math.ixx

```
module;// This module brings the other two modules together under the module name math

export module math;

export import math.add_sub; // We are export importing so importers of math have access to content from "sub-modules"
export import math.mult_div;
```

Sub-modules

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.28)
set(CMAKE_TOOLCHAIN_FILE "$ENV{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake" CACHE STRING "Vcpkg toolchain file")
project(Rocket  VERSION 1.0.0 LANGUAGES CXX)
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)

add_executable(${PROJECT_NAME} main.cpp)

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

Sub-modules

```
auto result = more_on_modules_03::add(10, 20);
fmt::println("result : {}", result);

auto result1 = more_on_modules_03::mult(10, 20);
fmt::println("result1 : {}", result1);
```

Module partitions

add_partition.ixx

```
module; // This partition can't be independently imported

export module math_v1:addition;

namespace more_on_modules_04{
  export{
    double add(double a, double b) {
      return a + b;
    }
  }
}  //namespace more_on_modules_04.
```

mult_partition.ixx

```
module; // This partition can't be independently imported

export module math_v1:multiplication;

namespace more_on_modules_04{
  export{
    double mult(double a, double b) {
      return a * b;
    }
  }
}  //namespace more_on_modules_04
```

math_v1.ixx

```
module; //This module brings together the partitions

export module math_v1;

export import :addition;
export import : multiplication;
```

Module partitions

```
/*
  . Module interface partitions:
    . This a technique used to split the module interface into many manageable sub-interfaces.
    . The sub-interfaces can't show up on the outside themselves; they have to be grouped,
      and exported together into the main interface file: math.ixx in our case.
    . partitions are internal to the module itself, they usually contain helper functions
      and entities that help with the implementation of the main interface.
    . Don't confuse partitions with submodules.
      . Submodules are legit full modules in their own right.We just conveniently name them
        with a dot in the name to give consumers of our module some granularity in having
        the choice to only use parts for our module when that helps.
*/
```

Module partitions

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.28)
set(CMAKE_TOOLCHAIN_FILE "$ENV{VCPKG_ROOT}/scripts/buildsystems/vcpkg.cmake" CACHE STRING "Vcpkg toolchain file")
project(Rocket  VERSION 1.0.0 LANGUAGES CXX)
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)

add_executable(${PROJECT_NAME} main.cpp)

target_sources(${PROJECT_NAME}
        PUBLIC
        FILE_SET CXX_MODULES FILES
        add_partition.ixx
        mult_partition.ixx
        math_v1.ixx
)
target_link_libraries(${PROJECT_NAME} PRIVATE fmt::fmt)
```

Module partitions

CMakeLists.txt

```
//#4: Module partitions: add_partition.ixx + mult_partition ==> math_v1.ixx
auto result1 = more_on_modules_04::add(10, 20);
fmt::println("result1: {}", result1);

auto result2 = more_on_modules_04::mult(10, 20);
fmt::println("result2: {}", result2);
```

Visibility and reachability

```
/*
    . Getting access to forbiden parts of our code
*/
```

Visibility and reachability

```cpp
//The point class is not exported.
class Point {
private:
  double x_;
  double y_;
public:
  Point(double x, double y) {
    this->x_ = x;
    this->y_ = y;
  }

  //An output stream operator to print the point
  friend std::ostream& operator<<(std::ostream& os, const Point& p) {
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
  }
  double get_x() { return x_; }
  double get_y() { return y_; }
  void set_x(double x) { this->x_ = x; }
  void set_y(double y) { this->y_ = y; }
};
```

Visibility and reachability

```
//The function returning Point is exported, even though Point itself is not exported.
export Point generate_random_point() {
    // Create random number generator engines for x and y
    std::random_device random_device;
    std::mt19937 generator_xy(random_device());

    // Define distributions for x and y coordinates
    std::uniform_real_distribution<double> distribution_xy(0, 99);

    return Point(distribution_xy(generator_xy), distribution_xy(generator_xy));     // When a Point object is returned,
                                                                                    // it will be reachable, but not visible.
}
```

Visibility and reachability

```cpp
//Point is not visible. We can't create its instance
//more_on_modules_05::math::Point p(1, 2);

//C++ is clever here by creating the p object without explicitly mentioning Point. We have indirect access.
auto p = more_on_modules_05::generate_random_point();
//print p
std::cout << p << std::endl;  //Indirect access to a reachable entity.
std::cout << "x: " << p.get_x() << std::endl;  //Indirect access to a reachable entity.
```

Private module fragment

```
/*
    . We keep the declaration separate from the implementation in a single file.
    . It SHOULD be in a single file.
    . Changes of code in the private fragement don't trigger recompilation of the translation units that
      import the module.
    . Good for optimized builds.
*/
```

Private module fragment

```
/*
    . Private module fragment
*/
module;

export module more_on_modules_06;

namespace more_on_modules_06{

    export double add( double a, double b);

}   //namespace more_on_modules_06

module: private;

namespace more_on_modules_06{

    double add( double a, double b){
        return a + b;
    }

}   //namespace more_on_modules_06
```