Canvas

```
/*
      . Topics:
          . Challenge with 2D array managed in memory
          . All the custom type facilities learn about so far
          . The modern way to do things


*/
```

Pixel

```cpp
export struct Position {
    unsigned int x{0};
    unsigned int y{0};
};

export class Pixel {
public:
    Pixel() = default;
    Pixel(uint32_t initial_color, Position initial_position);

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

    void set_position(Position position);
    Position get_position() const;

private:
    uint32_t m_color{0xFFFFFF}; // Default to white color in hex
    Position m_position{0, 0}; // Default initialized anyway.
};
```

Canvas

```cpp
export class Canvas {
public:
    Canvas(std::size_t width, std::size_t height);
    Canvas(const Canvas& src);
    ~Canvas();

    Canvas& operator=(const Canvas& rhs);

    void modify_pixel(std::size_t x, std::size_t y, const Pixel& pixel);
    Pixel& retrieve_pixel(std::size_t x, std::size_t y);

    void swap(Canvas& other) noexcept;

    void print() const;

private:
    void verify_coordinate(std::size_t x, std::size_t y) const;

    std::size_t m_width{ 0 };
    std::size_t m_height{ 0 };
    Pixel** m_pixels{ nullptr };
};
```

Canvas: Constructors

```cpp
Canvas::Canvas(std::size_t width, std::size_t height)
    : m_width{width}, m_height{height} {
    m_pixels = new Pixel*[m_width];
    for (std::size_t i = 0; i < m_width; ++i) {
        m_pixels[i] = new Pixel[m_height];
    }
}

Canvas::Canvas(const Canvas& src)
    : Canvas(src.m_width, src.m_height) {
    for (std::size_t i = 0; i < m_width; ++i) {
        for (std::size_t j = 0; j < m_height; ++j) {
            m_pixels[i][j] = src.m_pixels[i][j];
        }
    }
}

Canvas::~Canvas() {
    for (std::size_t i = 0; i < m_width; ++i) {
        delete[] m_pixels[i];
    }
    delete[] m_pixels;
    fmt::println("Canvas destroyed");
}
```

Canvas: Copy assignment operator

```
//Copy assignment operator; copy and swap idiom
Canvas& Canvas::operator=(const Canvas& rhs) {
    Canvas temp(rhs);
    swap(temp);
    return *this;
}
```

Canvas: Other methods

```cpp
void Canvas::modify_pixel(std::size_t x, std::size_t y, const Pixel& pixel) {
    verify_coordinate(x, y);
    m_pixels[x][y] = pixel;
}

Pixel& Canvas::retrieve_pixel(std::size_t x, std::size_t y) {
    verify_coordinate(x, y);
    return m_pixels[x][y];
}

void Canvas::swap(Canvas& other) noexcept {
    std::swap(m_width, other.m_width);
    std::swap(m_height, other.m_height);
    std::swap(m_pixels, other.m_pixels);
}

void Canvas::print() const {
    for (std::size_t y = 0; y < m_height; ++y) {
        for (std::size_t x = 0; x < m_width; ++x) {
            const Pixel& pixel = m_pixels[y][x];
            auto color = pixel.get_color();
            fmt::print("#{0:06X} ", color);
        }
        fmt::print("\n");
    }
}
```

Canvas: Other methods

```cpp
void Canvas::verify_coordinate(std::size_t x, std::size_t y) const {
    if (x ⩾ m_width || y ⩾ m_height) {
        throw std::out_of_range("Invalid coordinate");
    }
}

//global
void swap(Canvas& first, Canvas& second) noexcept {
    first.swap(second);
}
```

Canvas: Modern C++

```
    export class Canvas {
public:
    Canvas(std::size_t width, std::size_t height);
    Canvas(const Canvas& src);

    void modify_pixel(std::size_t x, std::size_t y, const Pixel& pixel);
    Pixel& retrieve_pixel(std::size_t x, std::size_t y);

    void print() const;

private:
    void verify_coordinate(std::size_t x, std::size_t y) const;

    std::size_t m_width{ 0 };
    std::size_t m_height{ 0 };
    std::vector<std::vector<Pixel>> m_pixels; // std::vector manages the memory
};
```

Canvas: Modern C++

```cpp
Canvas::Canvas(std::size_t width, std::size_t height)
    : m_width(width), m_height(height), m_pixels(width, std::vector<Pixel>(height)) {}

void Canvas::modify_pixel(std::size_t x, std::size_t y, const Pixel& pixel) {
    verify_coordinate(x, y);
    m_pixels[x][y] = pixel;
}

Pixel& Canvas::retrieve_pixel(std::size_t x, std::size_t y) {
    verify_coordinate(x, y);
    return m_pixels[x][y];
}

void Canvas::print() const {
    for (const auto& row : m_pixels) {
        for (const auto& pixel : row) {
            fmt::print("{:06X} ", pixel.get_color());
        }
        fmt::print("\n");
    }
}

void Canvas::verify_coordinate(std::size_t x, std::size_t y) const {
    if (x >= m_width || y >= m_height) {
        throw std::out_of_range("Coordinate out of range");
    }
}
```

Canvas: Usage

```cpp
auto canvas = std::make_unique<modern::Canvas>(10, 10);
auto pixel = std::make_unique<Pixel>(0xFF0000, Position{5, 5});
canvas->modify_pixel(5, 5, *pixel);
canvas->print();

fmt::println("Before copying");
auto canvas2 = std::make_unique<modern::Canvas>(10, 10);
canvas2->print();
fmt::println("After copying");
*canvas2 = *canvas;
canvas2->print();
```