Constructor delegation

```
/*
      . Topics:
          . Reducing repetitions
          . Having one constructor that does the heavy lifting
          . Making others rely on it for constructing objects


*/
```

Constructor delegation

```cpp
export class Pixel {
 public:
    // Default constructor
    Pixel() = default;

    // Constructor with color only, delegates to the three-argument constructor
    Pixel(uint32_t color) : Pixel(color, 0, 0) {
        fmt::print("Body of the one param constructor\n");
    }

    // Constructor with all arguments
    Pixel(uint32_t color, unsigned int x, unsigned int y)
        : m_color{color}, m_pos_x{x}, m_pos_y{y} {
        fmt::print("Three-argument constructor\n");
    }

    // Copy constructor
    Pixel(const Pixel& other)
        : Pixel(other.m_color, other.m_pos_x, other.m_pos_y) {
        fmt::print("Body of the copy constructor\n");
    }

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

Constructor delegation

```
/*
    . Exploring constructor delegation
        . Delegated constructors are put in the constructor initializer list.
        . They should be the only member initializer in the list.
            . a member can't be both initialized by a delegated constructor
                and have an initializer in the member initializer list.
            . Variables not initialized by a delegated constructor, can't be added to the initializer list too.
        . The one parameter constructor delegates to the three parameter constructor.
        . The copy constructor delegates to the three parameter constructor.
        . This approach keeps your code DRY (Don't Repeat Yourself) and makes it easier to maintain.
*/
```