Polymorphism: I want more!

```
/*

    . Topics:
        . Overriding, overloading and hiding
        . Polymorphism at different levels
        . Polymorphism with static members
*/
```

Polymorphism: Overloading, overriding and hiding

```
Shape
  |
  |---> Shape()                 // Default constructor
  |---> Shape(std::string_view) // Param constructor
  |---> virtual ~Shape()        // Virtual destructor
  |---> virtual draw() const    // Base draw()
  |---> virtual draw(int) const // Overloaded: draw with color depth
  v
Oval : public Shape
  |---> Oval()                    // Default constructor
  |---> Oval(double, double, std::string_view) // Param constructor
  |---> virtual draw() const override   // Overridden draw()
  |---> virtual draw(int, std::string_view) const // Overloaded: draw with color
  v
Circle : public Oval
  |---> Circle()                   // Default constructor
  |---> Circle(double, std::string_view)  // Param constructor
  |---> virtual draw() const override    // Overridden draw()
```

Polymorphism: Overloading, overriding and hiding

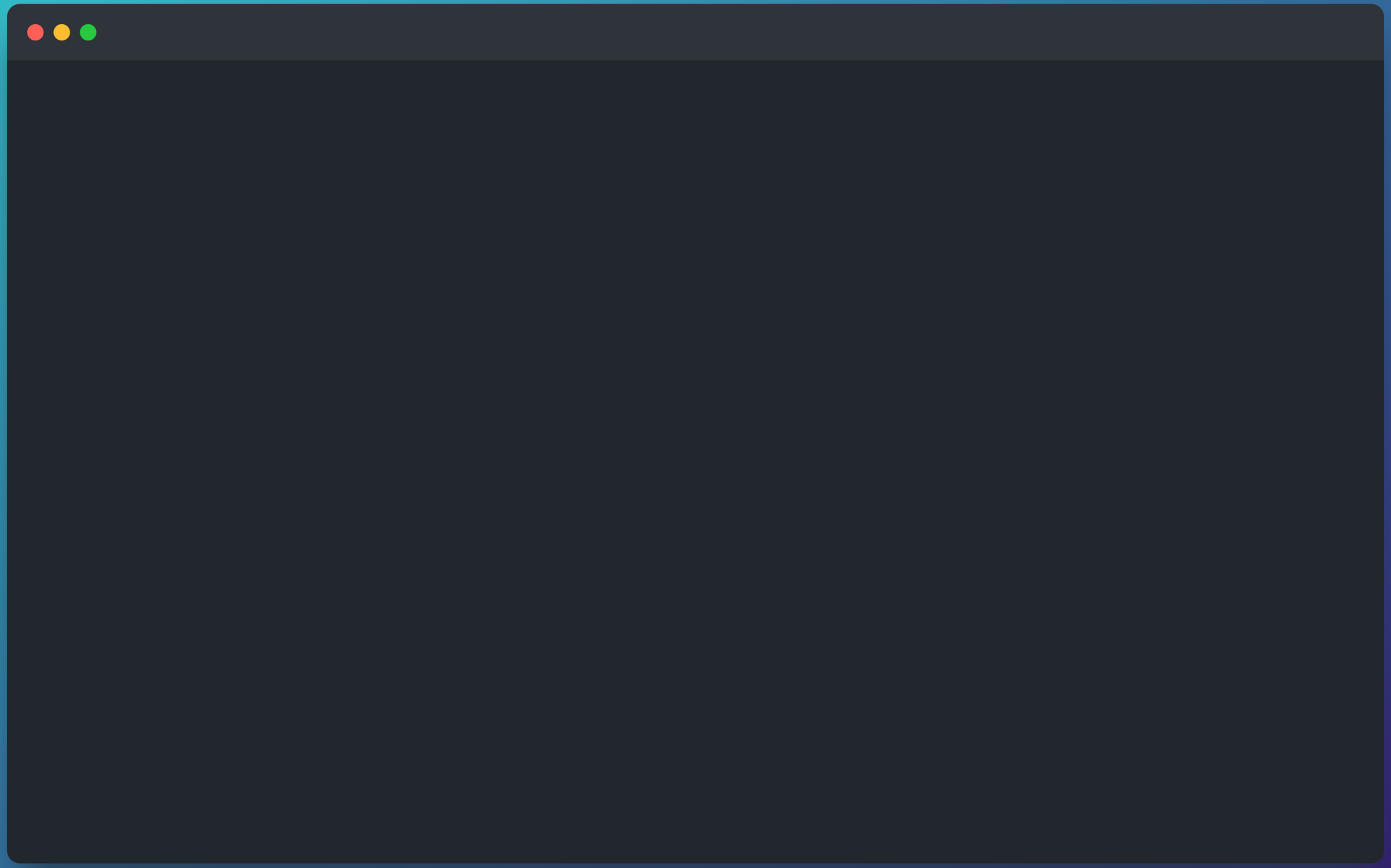
```
Shape
  |
  |---> Shape()                 // Default constructor
  |---> Shape(std::string_view) // Param constructor
  |---> virtual ~Shape()        // Virtual destructor
  |---> virtual draw() const    // Base draw()
  |---> virtual draw(int) const // Overloaded: draw with color depth
  v
Oval : public Shape
  |---> Oval()                    // Default constructor
  |---> Oval(double, double, std::string_view) // Param constructor
  |---> virtual draw() const override   // Overridden draw()
  |---> virtual draw(int, std::string_view) const // Overloaded: draw with color
  v
Circle : public Oval
  |---> Circle()                   // Default constructor
  |---> Circle(double, std::string_view)  // Param constructor
  |---> virtual draw() const override    // Overridden draw()
```

```
/*
        . Member Hiding: In the Oval class, the overloaded draw(int color_depth,
            std::string_view color) function hides the Shape::draw(int) function.

            . The moment you set up the two param draw function in the Oval class,
                the one param draw function in the Shape class is hidden.
                If you don't explicitly override the one param draw function in the
                Oval class, you lose the ability to call draw() on an Oval object.
*/
```

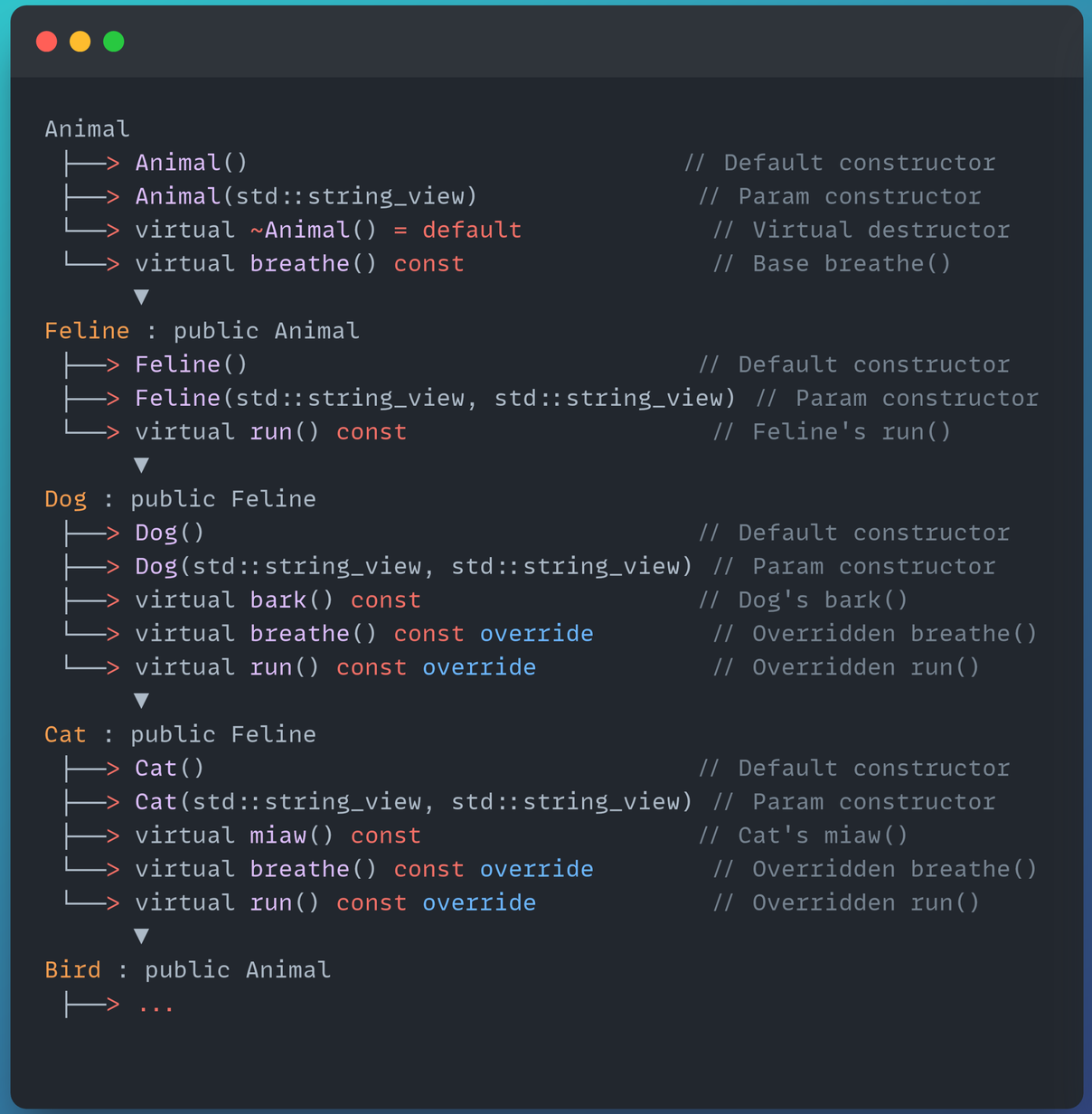
Polymorphism: Overloading, overriding and hiding

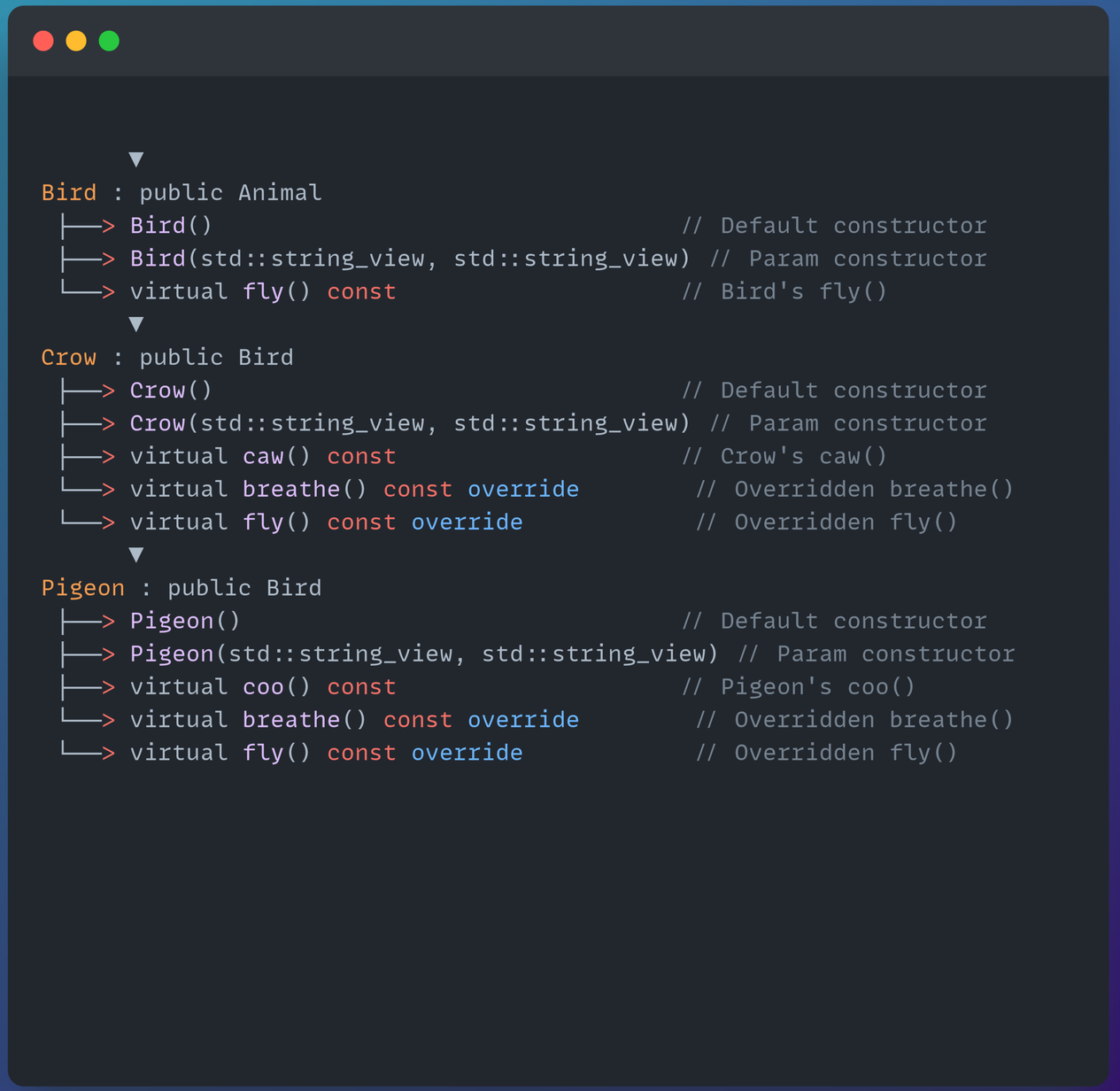
```
//Try it out
// Create a vector of unique_ptr to Shape
  std::vector<std::unique_ptr<Shape>> shapes;

  // Add shapes to the vector
  shapes.emplace_back(std::make_unique<Shape>("Generic Shape"));
  shapes.emplace_back(std::make_unique<Oval>(3.0, 5.0, "Oval Shape"));
  shapes.emplace_back(std::make_unique<Circle>(4.0, "Circle Shape"));

  // Drawing all shapes using polymorphism
  for (const auto& shape : shapes) {
      shape->draw();  // Calls the appropriate draw function (overridden)
  }

  // Demonstrating overloaded draw function
  shapes[1]->draw(24);  // Calls Oval's draw(int) (overloaded)

//The cast below is necessary because the draw member with two members
//is not part of the Shape's interface. We cast to Oval* because we know
//that the second element in the vector is an Oval.
  static_cast<Oval*>(shapes[1].get())->draw(24, "Blue");  // Calls Oval's overloaded draw(int, std::string_view)

//Set up a Oval object and call the draw() function
//Oval oval(3.0, 5.0, "Oval Shape");
//oval.draw();  // Calls Oval's draw() (overridden)
```

Demo time!



Polymorphism at different levels





Polymorphism at different levels

```
// Animal polymorphism
poly_2::Dog dog1("dark gray", "dog1");
poly_2::Cat cat1("black stripes", "cat1");
poly_2::Pigeon pigeon1("white", "pigeon1");
poly_2::Crow crow1("black", "crow1");

poly_2::Animal *animals[]{ &dog1, &cat1, &pigeon1, &crow1 };
fmt::println("\nAnimal polymorphism: ");
for (const auto &animal : animals) { animal→breathe(); }
```

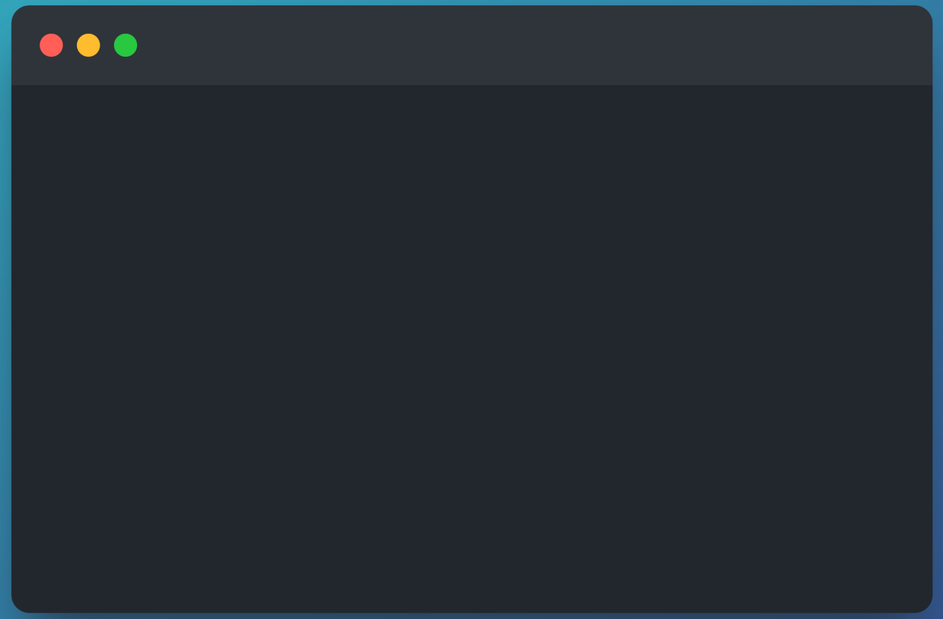
Polymorphism at different levels

```
// Feline polymorphism
poly_2::Dog dog2("dark gray", "dog2");
poly_2::Cat cat2("black stripes", "cat2");
poly_2::Pigeon pigeon2("white", "pigeon2");// Putting pigeon in felines will result in compiler error
                  //  pigeon is and Animal,a but is not a feline.
poly_2::Animal animal1("some animal");

poly_2::Feline *felines[]{ &dog2, &cat2 };
fmt::println("\nFeline polymorphism: ");
for (const auto &feline : felines) { feline->run(); }
```

Polymorphism at different levels

```cpp
// Bird polymorphism
poly_2::Pigeon pigeon3("white", "pigeon1");
poly_2::Crow crow3("black", "crow1");

poly_2::Bird *birds[]{ &pigeon3, &crow3 };
fmt::println("\nBird polymorphism: ");
for (const auto &bird : birds) { bird->fly(); }
```

Demo time!



Inheritance and polymorphism: static members

```
Shape
 ├──> Shape(std::string_view = "NoDescription") // Param constructor (default description)
 ├──> virtual ~Shape()                          // Virtual destructor
 ├──> virtual void draw() const                 // Base draw()
 ├──> virtual unsigned int get_count() const    // Base get_count()
 ├──> static unsigned int get_static_count()    // Static get_static_count()
 ├──> protected:                                 // Member variables
 │      std::string m_description;              // Shape description
 │      inline static unsigned int m_count{ 0 }; // Static instance count for Shape
 └──> private:                                  // No private members
      // No private members
      ▼
Ellipse : public Shape
 ├──> Ellipse(double x_radius = 0.0, double y_radius = 0.0,
               std::string_view = "NoDescription") // Param constructor
 ├──> ~Ellipse()                                // Destructor
 ├──> void draw() const override                // Overridden draw()
 ├──> unsigned int get_count() const override    // Overridden get_count()
 ├──> private:                                  // Member variables
 │      double m_x_radius;                      // X radius of the ellipse
 │      double m_y_radius;                      // Y radius of the ellipse
 │      inline static unsigned int m_count{ 0 }; // Static instance count for Ellipse
```

Inheritance and polymorphism: static members

```
Shape
 ├──> Shape(std::string_view = "NoDescription") // Param constructor (default description)
 ├──> virtual ~Shape()                          // Virtual destructor
 ├──> virtual void draw() const                 // Base draw()
 ├──> virtual unsigned int get_count() const    // Base get_count()
 ├──> static unsigned int get_static_count()    // Static get_static_count()
 ├──> protected:                                 // Member variables
 │      std::string m_description;              // Shape description
 │      inline static unsigned int m_count{ 0 }; // Static instance count for Shape
 └──> private:                                  // No private members
      // No private members
      ▼
Ellipse : public Shape
 ├──> Ellipse(double x_radius = 0.0, double y_radius = 0.0,
               std::string_view = "NoDescription") // Param constructor
 ├──> ~Ellipse()                                // Destructor
 ├──> void draw() const override                // Overridden draw()
 ├──> unsigned int get_count() const override    // Overridden get_count()
 ├──> private:                                  // Member variables
 │      double m_x_radius;                      // X radius of the ellipse
 │      double m_y_radius;                      // Y radius of the ellipse
 │      inline static unsigned int m_count{ 0 }; // Static instance count for Ellipse
```

Inheritance and polymorphism: static members

```
/*
    . Facts about static variables:
            . Single Instance: There is only one instance of the static variable
              shared across all instances of that class.
            . Independent in Each Class: If a derived class also declares a static
              variable with the same name, it creates a separate static variable
              independent of the base class's static variable.
*/
```

Inheritance and polymorphism: static members

```cpp
//The Shape class
export class Shape {
public:
    Shape(std::string_view description = "NoDescription")
        : m_description(description) { ++m_count; }

    virtual ~Shape() { --m_count; }

    virtual void draw() const {
        fmt::println("Shape::draw() called for: {}", m_description);
    }

    virtual unsigned int get_count() const { return m_count; }

    static unsigned int get_static_count() { return m_count; }

protected:
    std::string m_description;

private:
    inline static unsigned int m_count{ 0 };  // Static variable for Shape
};
```

Inheritance and polymorphism: static members

```
//The Ellipse class
export class Ellipse : public Shape {
public:
    Ellipse(double x_radius = 0.0, double y_radius = 0.0,
            std::string_view description = "NoDescription")
        : Shape(description), m_x_radius(x_radius), m_y_radius(y_radius) {
        ++m_count; // Increment Ellipse count
    }

    ~Ellipse() { --m_count; }

    void draw() const override {
        fmt::println("Ellipse::draw() called for: {} with radii ({}, {})",
                     m_description, m_x_radius, m_y_radius);
    }

    unsigned int get_count() const override { return m_count; }

    static unsigned int get_static_count() { return m_count; }

private:
    inline static unsigned int m_count{ 0 };  // Static variable for Ellipse
    double m_x_radius;
    double m_y_radius;
};
```

Inheritance and polymorphism: static members

```
//Trying it out
// Shape
poly_3::Shape shape1("shape1");
fmt::println("shape count : {}", poly_3::Shape::get_static_count()); // 1

poly_3::Shape shape2("shape2");
fmt::println("shape count : {}", poly_3::Shape::get_static_count());// 2

poly_3::Shape shape3;
fmt::println("shape count : {}", poly_3::Shape::get_static_count());// 3

fmt::println("*****************************************");

// Ellipse
poly_3::Ellipse ellipse1(10, 12, "ellipse1");
fmt::println("shape count : {}", poly_3::Shape::get_static_count());// 4
fmt::println("ellipse count : {}", poly_3::Ellipse::get_static_count());// 1
```