

C++ OOP Kapsamlı Proje: Geometrik Hesaplama Sistemi

Proje Özeti

Geometrik şekilleri yöneten, matematiksel hesaplamalar yapan bir sistem geliştireceksiniz. Bu proje, öğrendiğiniz tüm C++ OOP kavramlarını kullanmanızı gerektirecek.

Kullanmanız Gereken Konular (Kodlarınızdan Çıkarılan)

1. Sınıf Tasarımı ve Constructor'lar

- Constructor overloading
- Member initializer list
- Const member variables
- Private/public members
- Copy constructor
- Destructor

2. Memory Management

- Dynamic memory allocation (new/delete)
- Deep copy
- Arrays of objects

3. Static Members

- Static inline members
- Static member methods

4. Pointer ve Reference Kullanımı

- this pointer
- Reference parameters
- Pointer return values
- Method chaining
- Reference return values
- Pointer member access (->)

5. Function Özellikleri

- Inline functions
- Function overloading

- Default parameters
- Call by value vs reference
- Function signatures

6. Struct ve Global Kavramlar

- Struct kullanımı
- Global constants
- Global variables
- Global scope operator (::)
- Using declarations

7. C++20 Modules

- Module declaration (export module)
- Module import
- Export keyword
- Module interface (.ixx)
- Module implementation (.cpp)

8. Advanced Kavramlar

- Auto keyword
- Const pointers (3 çeşit)
- Friend functions
- Mutable members
- Const objects
- Const methods
- Operator overloading

Gerekli Sınıflar ve Yapılar

1. Point Sınıfı

```
cpp
```

```

class Point {
public:
    Point();                // Default constructor
    Point(int x, int y);    // Parametreli constructor
    Point(int minX, int minY, int x, int y); // 4 parametreli constructor
    Point(const Point& other); // Copy constructor
    ~Point();               // Destructor

    void move(int x, int y);
    void print() const;     // Const method
    bool isAtOrigin() const; // Const method
    double distanceFromOrigin() const;
    const Point* maxDistanceFromOrigin(const Point& other) const; // this pointer kullanımı

    static unsigned int getPointCount(); // Static method

    friend void displayPoint(const Point& p); // Friend function

private:
    int m_x{}, m_y{};
    const int MIN_X{}; // Const member (member initializer list ile)
    const int MIN_Y{}; // Const member
    static inline unsigned int s_pointCount{0}; // Static inline member
    mutable unsigned int m_accessCount{0}; // Mutable member
};

```

2. CustomString Sınıfı

```

cpp

class CustomString {
public:
    CustomString(const char* str); // Constructor
    CustomString(const CustomString& other); // Copy constructor
    ~CustomString();               // Destructor

    void print() const;
    size_t length() const;

private:
    char* m_data; // Dynamic memory
    size_t m_size;
};

```

3. Shape Sınıfı (Base class)

cpp

```
class Shape {
public:
    Shape(const char* name);
    virtual ~Shape();           // Virtual destructor

    virtual double area() const = 0;    // Pure virtual
    virtual double perimeter() const = 0; // Pure virtual
    virtual void print() const;

    static unsigned int getTotalShapes(); // Static method

protected:
    CustomString m_name;           // Composition
    static inline unsigned int s_totalShapes{0}; // Static member
};
```

4. Circle Sınıfı

cpp

```
class Circle : public Shape {
public:
    Circle();           // Default constructor
    Circle(double radius); // Parametrelili constructor
    Circle(const Point& center, double radius); // Point ile constructor
    Circle(const Circle& other); // Copy constructor

    // Operator overloading
    bool operator==(const Circle& other) const;
    bool operator>(const Circle& other) const;
    Circle operator+(const Circle& other) const;

    double area() const override;
    double perimeter() const override;
    void print() const override;

    friend void displayCircle(const Circle& c); // Friend function

private:
    Point m_center;           // Composition
    double m_radius{1.0};
    mutable unsigned int m_calculateCount{0}; // Mutable member
};
```

5. Rectangle Sınıfı

cpp

```
class Rectangle : public Shape {
public:
    Rectangle(double width = 1.0, double height = 1.0); // Default parameters
    Rectangle(const Point& topLeft, double width, double height);
    Rectangle(const Rectangle& other);    // Copy constructor
    ~Rectangle();                        // Destructor

    double area() const override;
    double perimeter() const override;

    Rectangle& setDimensions(double w, double h); // Method chaining için reference return
    Rectangle& move(const Point& newTopLeft);    // Method chaining

private:
    Point m_topLeft;
    double m_width, m_height;
};
```

6. MathUtils Struct

cpp

```
struct MathUtils {
    static inline double PI{3.14159};    // Static inline constant

    // Function overloading
    static inline double max(double a, double b) { return (a > b) ? a : b; }
    static inline double max(double a, double b, double c) { return max(max(a, b), c); }

    static double distance(const Point& p1, const Point& p2); // Reference parameters
};
```

Module Yapısı

geometry.ixx (Interface)

cpp

```
export module geometry;

export class Point;
export class Shape;
export class Circle;
export class Rectangle;
export class CustomString;

export const double PI;
export double calculateCircleArea(double radius);
export double calculateCirclePerimeter(double radius);
```

geometry.cpp (Implementation)

```
cpp

module geometry;

double calculateCircleArea(double radius) {
    return PI * radius * radius;
}

double calculateCirclePerimeter(double radius) {
    return 2 * PI * radius;
}
```

utils.ixx (Utility Module)

```
cpp

export module utils;

export struct MathUtils;
export void printSeparator();
export template<typename T> inline T maximum(T a, T b); // Inline function template
```

main.cpp Gereksinimleri

Global Değişkenler ve Using

```
cpp
```

```
#include <iostream>

import geometry;
import utils;

using std::cout;
using std::endl;

const int MAX_SHAPES(100);           // Global constant

unsigned int g_operationCount(0);     // Global variable
```

Ana Program Özellikleri

1. Constructor Testing

- Farklı constructor'ları test et
- Copy constructor davranışını göster
- Member initializer list kullanımını göster

2. Memory Management

- Dynamic object creation (3 farklı yöntem)
- Object arrays
- Proper cleanup (delete)

3. Static Member Testing

- Static member count tracking
- Static method calls

4. Operator Overloading

- Comparison operators test
- Arithmetic operators test
- Method chaining examples

5. Pointer/Reference Usage

- this pointer usage examples
- Reference parameter passing
- Pointer return value usage

6. Const Correctness

- Const objects
- Const methods
- Mutable member behavior

7. Friend Function Usage

- Friend function examples
- Private member access

Örnek Program Akışı

cpp


```
int main() {  
    // Global scope operator usage  
    cout << "Max shapes allowed: " << ::MAX_SHAPES << endl;  
  
    // Auto keyword usage  
    auto point1 = Point(10, 20);  
    auto point2 = Point(5, 5, 15, 25); // member initializer list constructor  
  
    // Copy constructor  
    Point point3{point1};  
  
    // Dynamic allocation (3 ways)  
    Point* ptr1 = new Point;  
    Point* ptr2 = new Point(30, 40);  
    Point* ptr3{new Point(point2)};  
  
    // Object arrays  
    Point pointArray[3] = {Point(), Point{1, 2}}; // default constructor for third  
  
    // Static member usage  
    cout << "Total points: " << Point::getPointCount() << endl;  
  
    // Method chaining  
    Rectangle rect(10, 20);  
    rect.setDimensions(15, 25).move(Point(5, 5));  
  
    // Operator overloading  
    Circle c1(5.0);  
    Circle c2(3.0);  
    if (c1 > c2) {  
        cout << "c1 is bigger" << endl;  
    }  
  
    Circle c3 = c1 + c2; // operator+ usage  
  
    // Friend function usage  
    displayCircle(c1);  
  
    // Const object and mutable  
    const Circle constCircle(7.0);  
    constCircle.area(); // This should increment mutable counter  
  
    // Function overloading  
    cout << MathUtils::max(10.5, 20.3) << endl;  
    cout << MathUtils::max(10.5, 20.3, 15.7) << endl;
```

```
// Reference vs value
testValueVsReference(point1);

// Cleanup
delete ptr1;
delete ptr2;
delete ptr3;

return 0;
}
```

CMakeLists.txt

```
cmake

cmake_minimum_required(VERSION 3.20)
project(GeometrySystem CXX)

set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Module support for different compilers
if(CMAKE_CXX_COMPILER_ID STREQUAL "MSVC")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /experimental:module")
elseif(CMAKE_CXX_COMPILER_ID STREQUAL "GNU")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fmodules-ts")
elseif(CMAKE_CXX_COMPILER_ID STREQUAL "Clang")
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fmodules")
endif()

add_executable(GeometrySystem
    src/main.cpp
    src/geometry.cpp
    src/utils.cpp
)

target_include_directories(GeometrySystem PRIVATE include)
```

Proje Yapısı

```
GeometrySystem/  
├── CMakeLists.txt  
├── src/  
│   ├── main.cpp  
│   ├── geometry.ixx  
│   ├── geometry.cpp  
│   ├── utils.ixx  
│   └── utils.cpp  
└── include/  
    └── common.h
```

Test Edilmesi Gereken Durumlar

1. **Memory Management:** Valgrind ile memory leak kontrolü
2. **Copy Semantics:** Deep vs shallow copy behavior
3. **Static Members:** Multiple object creation/destruction
4. **Const Correctness:** Const objects, mutable members
5. **Operator Overloading:** All overloaded operators
6. **Method Chaining:** Reference return values
7. **Module System:** Import/export functionality
8. **Friend Functions:** Private member access
9. **Constructor Overloading:** Different initialization paths
10. **Global Scope:** :: operator usage

Değerlendirme Kriterleri

- Tüm 36 konunun doğru implementasyonu
- Code organization ve clean code principles
- Memory management (no leaks)
- Const correctness
- Proper module usage
- CMake build system
- Comprehensive testing

Bu proje, kodlarınızda bulunan **her bir konuyu** kapsamlı şekilde kullanmanızı sağlayacak ve C++ OOP bilginizi derinlemesine test edecektir.