**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

Факультет информационных технологий и программирования

Лабораторная работа №4

Вариант 21.

*Виртуальные функции.*

**Выполнил студент группы № M3111**
Соловьев Михаил Александрович.

Санкт-Петербург
2023

В данном задании необходимо согласно варианту реализовать все указанные интерфейсы (абстрактные базовые классы) для классов: Отрезок и Шестиугольник
Функционал системы:
- Хранение множества фигур
- Динамическое добавление фигур пользователем.
- Отобразить все фигуры.
- Суммарная площадь всех фигур.
- Суммарный периметр всех фигур.
- Центр масс всей системы.
- Память, занимаемая всеми экземплярами классов.
- Сортировка фигур между собой по массе.

Код:

hexagon.cpp

```
#include "hexagon.h"
#include "interfaces.h"
#include <cmath>

Hexagon::Hexagon() {
    this->Hexagon::initFromDialog();
}
Hexagon::~Hexagon() = default;

const char* Hexagon::className() {
    return name;
}

unsigned int Hexagon::size() {
    return sizeof(*this);
}

double Hexagon::square() {
    return (3 * sqrt(3) * pow(sqrt(pow(A.x, 2) + pow(A.y, 2)), 2)) / 2;
}

double Hexagon::perimeter() {
    return 6 * sqrt(pow(A.x, 2) + pow(A.y, 2));
}

double Hexagon::mass() const {
```

```
    return weight;
}

CVector2D Hexagon::position() {
    center.x = (A.x + C.x) / 2;
    center.y = (A.y + C.y) / 2;
    return center;
}

void Hexagon::initFromDialog() {
    std::cout << "Add first point: \n";
    std::cin >> A.x >> A.y;
    std::cout << "Add second point: \n";
    std::cin >> B.x >> B.y;
    std::cout << "Add third point: \n";
    std::cin >> C.x >> C.y;
    std::cout << "Add fourth point: \n";
    std::cin >> D.x >> D.y;
    std::cout << "Add fifth point: \n";
    std::cin >> E.x >> E.y;
    std::cout << "Add sixth point: \n";
    std::cin >> F.x >> F.y;
    std::cout << "Add weight: " << std::endl;
    std::cin >> weight;
}

void Hexagon::draw() {
    std::cout << "Name: " << this->className() << "\n";
    std::cout << "Position: " << "(" << A.x << ", " << A.y << "); "
            << "(" << B.x << ", " << B.y << "); "
            << "(" << C.x << ", " << C.y << "); "
            << "(" << D.x << ", " << D.y << "); "
            << "(" << E.x << ", " << E.y << "); "
            << "(" << F.x << ", " << F.y << "); " << "\n";
    std::cout << "Weight: " << weight << "\n";
    std::cout << "Size: " << size() << "\n";
    std::cout << "Perimeter: " << perimeter() << "\n";
    std::cout << "Square: " << square() << "\n";
}

bool Hexagon::operator<(IPhysObject &obj) const {
    return mass() < obj.mass();
```

```
}

bool Hexagon::operator==(IPhysObject &obj) const {
    return mass() == obj.mass();
}
```

hexagon.h

```
#ifndef COURSE_C___HEXAGON_H
#define COURSE_C___HEXAGON_H

#include "interfaces.h"

class Hexagon: public IFigure {
private:
    static inline const char* name = "Hexagon";
    CVector2D A;
    CVector2D B;
    CVector2D C;
    CVector2D D;
    CVector2D E;
    CVector2D F;
    CVector2D center;
    double weight = 0;

public:
    Hexagon();
    ~Hexagon();
    const char* className() override;
    unsigned int size() override;
    double square() override;
    double perimeter() override;
    double mass() const override;
    CVector2D position() override;
    void initFromDialog() override;
    void draw() override;
    bool operator==(IPhysObject &obj) const override;
    bool operator<(IPhysObject &obj) const override;
};

#endif //COURSE_C___HEXAGON_H
```

interfaces.h

```
#ifndef COURSE_C___INTERFACES_H
#define COURSE_C___INTERFACES_H

#include <iostream>

class IGeoFig {
public:
   virtual double square() = 0;
   virtual double perimeter() = 0;
};

class CVector2D {
public:
   double x, y;

   CVector2D() = default;
};

class IPhysObject {
public:
   virtual double mass() const = 0;
   virtual CVector2D position() = 0;
   virtual bool operator== (IPhysObject &obj) const = 0;
   virtual bool operator< (IPhysObject &obj) const = 0;
};

class IPrintable {
public:
   virtual void draw() = 0;
};

class IDialogInitiable {
   virtual void initFromDialog() = 0;
};

class BaseCObject {
public:
   virtual const char* className() = 0;
   virtual unsigned int size() = 0;
};
```

```
class IFigure: public IGeoFig, public IPhysObject, public IPrintable, public
IDialogInitiable,
        public BaseCObject {};

#endif //COURSE_C___INTERFACES_H
```

main.cpp

```
#include <iostream>
#include <vector>
#include "interfaces.h"
#include "segment.h"
#include "hexagon.h"

unsigned int getTotalMemoryUsage(const std::vector<IFigure*> &shapes) {
    unsigned int total = 0;
    for (auto shape : shapes) {
        total += shape->size();
    }
    return total;
}

void sortShapesByMass(std::vector<IFigure*>& shapes) {
    std::sort(shapes.begin(), shapes.end(), [](IFigure* a, IFigure* b) {
        return *a < *b;
    });
}

int main() {
    std::vector<IFigure*> shapes;
    int choice = 0;

    while (choice != 9) {
        std::cout << "\nSelect an option:\n"
                  << "1. Add a segment\n"
                  << "2. Add a hexagon\n"
                  << "3. List all shapes\n"
                  << "4. Calculate total area of all shapes\n"
                  << "5. Calculate total perimeter of all shapes\n"
                  << "6. Calculate center of mass of the whole system\n"
                  << "7. Total memory usage:\n"
                  << "8. Sorting shapes with each other by mass:\n"
                  << "9. Quit" << std::endl;
```

```
        std::cin >> choice;

        switch (choice) {
            case 1: {
                auto *segment = new Segment();
                shapes.push_back(segment);
                break;
            }

            case 2: {
                auto *hexagon = new Hexagon();
                shapes.push_back(hexagon);
                break;
            }

            case 3: {
                std::cout << "List of Shapes:\n";
                for (auto shape: shapes) {
                    std::cout << " - " << shape->className() << "\n";
                }
                break;
            }

            case 4: {
                double totalArea = 0.0;
                for (auto shape: shapes) {
                    totalArea += shape->square();
                }
                std::cout << "\nThe total area of all shapes is " << totalArea << ".\n";
                break;
            }

            case 5: {
                double totalPerimeter = 0.0;
                for (auto shape: shapes) {
                    totalPerimeter += shape->perimeter();
                }
                std::cout << "\nThe total perimeter of all shapes is " << totalPerimeter
<< ".\n";
                break;
            }
```

```
        case 6: {
            double totalMass = 0.0;
            double xCenter = 0.0;
            double yCenter = 0.0;

            // Calculate total mass and center of mass of the whole system
            for (auto shape: shapes) {
                double mass = shape->mass();
                totalMass += mass;
                xCenter += mass * shape->position().x;
                yCenter += mass * shape->position().y;
            }

            xCenter /= totalMass;
            yCenter /= totalMass;

            std::cout << "\nThe center of mass of the whole system is (" << xCenter
<< ", " << yCenter << ").\n";
            break;
        }

        case 7: {
            std::cout << "Total memory usage: " << getTotalMemoryUsage(shapes)
<< " bytes\n";
            break;
        }

        case 8: {
            sortShapesByMass(shapes);
            std::cout << "Shapes sorted by mass:\n";
            for (auto shape: shapes) {
                std::cout << " - " << shape->className() << " (mass = " << shape-
>mass() << ")\n";
            }
            break;
        }

        case 9: {
            std::cout << "Exit\n";
            break;
        }
```

```
            default: {
                std::cout << "Invalid choice\n";
                break;
            }
        }
    }

    for (auto shape : shapes) {
        delete shape;
    }

    return 0;
}
```

segment.cpp

```
#include "interfaces.h"
#include "segment.h"
#include <cmath>

Segment::Segment() {
    this->Segment::initFromDialog();
}

Segment::~Segment() = default;

const char* Segment::className() {
    return name;
}

unsigned int Segment::size() {
    return sizeof(*this);
}

double Segment::square() {
    return 0;
}

double Segment::perimeter() {
    return sqrt(pow(len.x, 2) + pow(len.y, 2));
}

double Segment::mass() const {
```

```
    return weight;
}

CVector2D Segment::position() {
    return len;
}

void Segment::initFromDialog() {
    std::cout << "Enter the vector, that describe your segment: " << std::endl;
    std::cin >> len.x >> len.y;
    std::cout << "Add weight: " << std::endl;
    std::cin >> weight;
}

void Segment::draw() {
    std::cout << "Name: " << this->className() << "\n";
    std::cout << "Position: " << this->len.x << ' ' << this->len.y << "\n";
    std::cout << "Length: " << perimeter() << "\n";
    std::cout << "Weight: " << weight << "\n";
    std::cout << "Size: " << size() << "\n";
}

bool Segment::operator==(IPhysObject &obj) const {
    return mass() == obj.mass();
}

bool Segment::operator<(IPhysObject &obj) const {
    return mass() < obj.mass();
}
```

segment.h

```
#ifndef COURSE_C___SEGMENT_H
#define COURSE_C___SEGMENT_H

#include "interfaces.h"

class Segment: public IFigure {
    static inline const char* name="Segment";
    double weight = 0;
    CVector2D len;

public:
```

```
    Segment();
    ~Segment();

    const char *className() override;
    unsigned int size() override;
    double square() override;
    double perimeter() override;
    double mass() const override;
    CVector2D position() override;
    void initFromDialog() override;
    void draw() override;

    bool operator==(IPhysObject &obj) const override;
    bool operator<(IPhysObject &obj) const override;
};

#endif //COURSE_C___SEGMENT_H
```

Вывод:
Я реализовал указанные классы и научился работать с интерфейсами