МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №4

По дисциплине «Программирование»

Выполнил студент группы №M3106

Казаков Никита Андреевич

Проверил

Повышев Владислав Вячеславович

Лабораторная работа №4. *“Виртуальные функции”.*
Реализовать все указанные интерфейсы (абстрактные базовые классы) для классов (согласно варианту):
A. Круг
B. Отрезок
C. Равносторонний треугольник
D. Прямоугольник
E. Шестиугольник
F. Параллелограмм
G. Равнобедренная трапеция
H. Эллипс (периметр можно считать по любой приближенной формуле: см. интернет, справочники и т.п.).
Функционал системы:
- Хранение множества фигур
- Динамическое добавление фигур пользователем. (через консоль)
- 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.
Вопросы для обдумывания:
- Есть ли необходимость делать методы сравнения по массе виртуальными?
- Получится ли также перегрузить операторы сравнения для интерфейса BaseCObject чтобы сравнивать объекты по объему занимаемой памяти?
- Предположите, что в дальнейшем придется изменить код таким образом, чтобы фигуры (оставаясь сами по себе плоскими) задавались уже не в двумерном, а в трехмерном пространстве. Укажите как бы вы действовали? Что пришлось бы изменить?

```
// Интерфейс "Геометрическая фигура".
class IGeoFig {
public:
// Площадь.
virtual double square() = 0;
// Периметр.
virtual double perimeter() = 0;
};
// Вектор
class CVector2D {
public:
double x, y;

};
// Интерфейс "Физический объект".
class IPhysObject {
public:
// Масса, кг.
virtual double mass() = 0;
// Координаты центра масс, м.
virtual Vector2D position() = 0;
```

// Сравнение по массе.
virtual bool operator== (const PhysObject& ob) const = 0;
// Сравнение по массе.
virtual bool operator< (const PhysObject& ob) const = 0;
};
// Интерфейс "Отображаемый"
class IPrintable {
public:
// Отобразить на экране
// (выводить в текстовом виде параметры фигуры).
virtual void draw() = 0;
};
// Интерфейс для классов, которые можно задать через диалог с пользователем.
class IDialogInitiable {
// Задать параметры объекта с помощью диалога с пользователем.
virtual void initFromDialog() = 0;
};
// Интерфейс "Класс"
class BaseCObject {
public:
// Имя класса (типа данных).
virtual const char classname() = 0;*
// Размер занимаемой памяти.
virtual unsigned int size() = 0;
};

main.cpp

```
#include <iostream>
#include "side_lib.hpp"
int main() {
	bool p_end = false;
	comm_list();
	while (!p_end) {
		p_end = command_input();
		if (!p_end) {
			std::cout << "All commands list - 8\n";
		}
	}
	std::cout << "==========Shutting down==========\n";
	return 0;
}
```

side_lib.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "Interfaces.hpp"
#include "side_lib.hpp"
#include "Circle.hpp"
#include "Equil_triangle.hpp"
```

```
std::vector <Figure *> shapes;
int order = 0;
void comm_list () {
	std::cout << "==========COMMANDS LIST==========\n";
	std::cout << "Add new shape - 1\n";
	std::cout << "Output all info about shapes - 2\n";
	std::cout << "Output overall square of all shapes - 3\n";
	std::cout << "Output overall perimeter of all shapes - 4\n";
	std::cout << "Output mass center - 5\n";
	std::cout << "Output memory usage - 6\n";
	std::cout << "Sort shapes - 7\n";
	std::cout << "================================\n";
	std::cout << "Shut down - 0\n\n";
}
bool command_input () {
	int temp_comm;
	std::cout << "Enter command\n~> ";
	std::cin >> temp_comm;
	switch (temp_comm) {
		case p_exit: {
			return true;
		} case new_shape: {
			int shape;
			std::cout << "=======Choose shape=======\n";
			std::cout << "Circle - 0\n";
			std::cout << "Equilateral triangle - 1\n";
			std::cout << "Cancel - other values\n";
			std::cout << "==========================\n";
			std::cout << "~> ";
			std::cin >> shape;
			if (shape == circle) {
				shapes.push_back(new Circle);
			} else if (shape == equil_triangle) {
				shapes.push_back(new Equil_triangle);
			} else {
				break;
			}
			shapes[order]->initFromDialog();
			order++;
			std::cout << '\n';
			break;
		} case all_shapes_info: {
			for (Figure *figure : shapes) {
				std::cout << figure << ": " << figure->classname() << '\n';
				figure->draw();
				std::cout << '\n';
			}
			std::cout << '\n';
			break;
		} case all_shapes_square: {
			double total = 0;
			for (Figure *figure : shapes) {
				total += figure->square();
			}
			std::cout << "Overall square: " << total << '\n';
			std::cout << '\n';
			break;
		} case all_shapes_perimeter: {
			double total = 0;
```

```
                for (Figure *figure : shapes) {
                        total += figure->perimeter();
                }
                std::cout << "Overall perimeter: " << total << '\n';
                std::cout << '\n';
                break;
        } case mass_center: {
                CVector2D mass_center = CVector2D();
                double mass_sum = 0;
                for (Figure *figure : shapes) {
                        mass_sum += figure->mass();
                        mass_center.x += figure->mass() * figure->position().x;
                        mass_center.y += figure->mass() * figure->position().y;
                }
                mass_center.x /= mass_sum;
                mass_center.y /= mass_sum;
                std::cout << "Mass center: {" << mass_center.x << ", " << mass_center.y <<
"}\n";
                std::cout << '\n';
                break;
        } case memory_size: {
                unsigned int mem_size = 0;
                for (Figure *figure : shapes) {
                        mem_size += figure->size();
                }
                std::cout << "Memory usage: " << mem_size << '\n';
                std::cout << '\n';
                break;
        } case mass_sort: {
                std::sort(shapes.begin(), shapes.end(), [](Figure *a, Figure *b) { return a->mass()
< b->mass(); });
                std::cout << "========Shapes sorted========\n";
                std::cout << '\n';
                break;
        } case command_list: {
                comm_list();
                break;
        } default: {
                std::cout << "==========Undefined command==========\n";
                std::cout << '\n';
        }
    }
    return false;
}
```

side_lib.hpp

```
#pragma once
void comm_list ();
bool command_input ();
enum command {
    p_exit,
    new_shape,
    all_shapes_info,
    all_shapes_square,
    all_shapes_perimeter,
    mass_center,
    memory_size,
    mass_sort,
```

```
	command_list,
};
enum Shape {
	circle,
	equil_triangle,
};
```

Circle.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>
#include "Circle.hpp"
Circle::Circle() : radius(0), weight(0) {}
void Circle::initFromDialog(){
	std::cout << "Enter coordinates" << '\n';
	std::cout << "~>x: ";
	std::cin >> place.x;
	std::cout << "~>y: ";
	std::cin >> place.y;
	std::cout << "Enter radius: \n~> ";
	std::cin >> radius;
	std::cout << "Enter mass: \n~> ";
	std::cin >> weight;
}
double Circle::square() {
	return M_PI * pow(radius, 2);
}
double Circle::perimeter() {
	return 2 * M_PI * radius;
}
double Circle::mass() const {
	return weight;
}
CVector2D Circle::position() {
	return place;
}
bool Circle::operator== (const IPhysObject & circle) const {
	return mass() == circle.mass();
}
bool Circle::operator< (const IPhysObject & circle) const {
	return mass() < circle.mass();
}
void Circle::draw() {
	std::cout << "Square: " << square() << '\n';
	std::cout << "Perimeter: " << perimeter() << '\n';
	std::cout << "Circle mass: " << mass() << '\n';
	std::cout << "Mass center: ";
	std::cout << "{" << position().x  << ", " << position().y << "}\n";
}
std::string Circle::classname() {
	return name;
}
unsigned int Circle::size() {
	return sizeof(*this);
}
```

Circle.hpp

```
#pragma once
```

```cpp
#include "Interfaces.hpp"
class Circle : public Figure {
	CVector2D place;
	double radius;
	double weight;
	std::string name = "Circle";
public:
	Circle();
	~Circle() = default;
	void initFromDialog() override;
	double square() override;
	double perimeter() override;
	double mass() const override;
	CVector2D position() override;
	bool operator< (const IPhysObject &) const override;
	bool operator== (const IPhysObject &) const override;
	unsigned int size() override;
	void draw() override;
	std::string classname() override;
};
```

Equil_triangle.cpp

```cpp
#include <iostream>
#include <cmath>
#include "Equil_triangle.hpp"
Equil_triangle::Equil_triangle() : side_size(0), weight(0) {}
void Equil_triangle::initFromDialog(){
	std::cout << "Enter coordinates" << '\n';
	std::cout << "~>x: ";
	std::cin >> place.x;
	std::cout << "~>y: ";
	std::cin >> place.y;
	std::cout << "Enter side length: \n~> ";
	std::cin >> side_size;
	std::cout << "Enter mass: \n~> ";
	std::cin >> weight;
}
double Equil_triangle::square() {
	return sqrt(3) * pow(side_size, 2) / 4;
}
double Equil_triangle::perimeter() {
	return 3 * side_size;
}
double Equil_triangle::mass() const {
	return weight;
}
CVector2D Equil_triangle::position() {
	return place;
}
bool Equil_triangle::operator== (const IPhysObject & circle) const {
	return mass() == circle.mass();
}
bool Equil_triangle::operator< (const IPhysObject & circle) const {
	return mass() < circle.mass();
}
void Equil_triangle::draw() {
	std::cout << "Square: " << square() << '\n';
	std::cout << "Perimeter: " << perimeter() << '\n';
	std::cout << "Equilateral triangle mass: " << mass() << '\n';
```

```
	std::cout << "Mass center: ";
	std::cout << "{" << position().x  << ", " << position().y << "}\n";
}
std::string Equil_triangle::classname() {
	return name;
}
unsigned int Equil_triangle::size() {
	return sizeof(*this);
}
```

Equil_triangle.hpp

```
#pragma once
#include "Interfaces.hpp"
class Equil_triangle : public Figure {
	CVector2D place;
	double side_size;
	double weight;
	std::string name = "Equil_triangle";
public:
	Equil_triangle();
	~Equil_triangle() = default;
	void initFromDialog() override;
	double square() override;
	double perimeter() override;
	double mass() const override;
	CVector2D position() override;
	bool operator< (const IPhysObject &) const override;
	bool operator== (const IPhysObject &) const override;
	unsigned int size() override;
	void draw() override;
	std::string classname() override;
};
```