Parallelogram.h

```cpp
#pragma once
#include <string>
#include "interface.h"
class Parallelogram : public IAllTheInterfaces
{
private:
    double side_a_;
    double side_b_;
    double ox_;
    double oy_;
    int angle_;
    double private_mass_;
    std::string name_ = "Parallelogram";
    CVector2D private_position_{};

public:
    Parallelogram();
    Parallelogram(double, double, double, double, int, double, CVector2D&);
    ~Parallelogram();
    double square() override;
    double perimeter() override;
    double mass() const override;
    CVector2D position() override;
    bool operator== (const IPhysObject& ob) const override;
    bool operator< (const IPhysObject& ob) const override;
    void draw() override;
    void initFromDialog() override;
    std::string classname() const override;
    unsigned int size() const override;
};
```

Circle.h

```cpp
#pragma once
#include <string>
#include "interface.h"
class Circle : public IAllTheInterfaces{
private:
    CVector2D r_{};
    double ox_;
    double oy_;
    double private_mass_;
    std::string name_ = "Circle";
    CVector2D private_position_{};
public:
    Circle();
    Circle(CVector2D&, double, double, double, CVector2D&);
    ~Circle();
    double square() override;
    double perimeter() override;
    double mass() const override;
    CVector2D position() override;
    bool operator== (const IPhysObject& ob) const override;
    bool operator< (const IPhysObject& ob) const override;
    void draw() override;
    void initFromDialog() override;
    std::string classname() const override;
    unsigned int size() const override;

    double get_r() const;
};
```

Parallelogram.cpp

```
#include <iostream>
#include <cmath>
#include "Parallelogram.h"

Parallelogram::Parallelogram() :
        side_a_(0), side_b_(0), ox_(0), oy_(0), angle_(0), private_mass_(0),
        private_position_(0, 0) {};
Parallelogram::Parallelogram(const double side_a, const double side_b, const double
ox, const double oy, const int angle, const double private_mass, CVector2D&
private_position) :
        side_a_(side_a), side_b_(side_b), ox_(ox), oy_(oy), angle_(angle),
        private_mass_(private_mass), private_position_(private_position) {};

double Parallelogram::square(){
    return(side_a_ * side_b_ * sin(angle_));
}

double Parallelogram::perimeter(){
    return(2 * (side_a_ + side_b_));
}

double Parallelogram::mass() const{
    return private_mass_;
}

CVector2D Parallelogram::position(){
    return private_position_;
}

bool Parallelogram::operator== (const IPhysObject& ob) const{
    return(mass() == ob.mass());
}

bool Parallelogram::operator<(const IPhysObject& ob) const{
    return(mass() < ob.mass());
}

void Parallelogram::draw(){
    std::cout << "Center coordinates: {" << ox_ << ", " << oy_ << "}" << std::endl
              << "Perimeter: " << perimeter() << std::endl
              << "Area: " << square() << std::endl
              << "Mass: " << private_mass_ << std::endl
              << "Position coordinates: {" << private_position_.x << ", " <<
private_position_.y << "}" << std::endl;
}

void Parallelogram::initFromDialog(){
    std::cout << "Please enter center coordinates (x y), coordinates of current
position (x y)," << std::endl
              << " lengths of both sides (a b), angle (number), and a mass (number):"
<< std::endl;
    std::cin >> ox_ >> oy_ >> private_position_.x >> private_position_.x >> side_a_
>> side_b_ >> angle_ >> private_mass_;
}

std::string Parallelogram::classname() const{
    return name_;
}

unsigned Parallelogram::size() const{
    return sizeof(*this);
}

Parallelogram::~Parallelogram() = default;
```

Circle.cpp

```
#include <iostream>
#include <cmath>
#include "Circle.h"
Circle::Circle() : r_(0, 0), ox_(0), oy_(0),
        private_mass_(0), private_position_(0, 0) {};
Circle::Circle(CVector2D& r, const double ox, const double oy, const double
private_mass, CVector2D& private_position) :
        r_(r), ox_(ox), oy_(oy),
        private_mass_(private_mass),
        private_position_(private_position) {};

double Circle::get_r() const{
    return(sqrt((r_.x - ox_) * (r_.x - ox_) + (r_.y - oy_) * (r_.y - oy_)));
}
double Circle::square(){
    return(3.1415 * pow(get_r(), 2));
}
double Circle::perimeter(){
    return(2 * 3.1415 * get_r());
}
double Circle::mass() const{
    return private_mass_;
}
CVector2D Circle::position(){
    return private_position_;
}
bool Circle::operator== (const IPhysObject& ob) const{
    return(mass() == ob.mass());
}
bool Circle::operator< (const IPhysObject& ob) const{
    return(mass() < ob.mass());
}
void Circle::draw(){
    std::cout << "Center coordinates: {" << ox_ << ", " << oy_ << "}" << std::endl
              << "Radius: " << get_r() << std::endl
              << "Perimeter: " << perimeter() << std::endl
              << "Area: " << square() << std::endl
              << "Mass: " << private_mass_ << std::endl
              << "Position coordinates: {" << private_position_.x << ", " <<
private_position_.y << "}" << std::endl;
}
void Circle::initFromDialog(){
    std::cout << "Please enter center coordinates (x y), coordinates where radius
ends (x y)," << std::endl
              << " coordinates of current position (x y), and a mass (number):" <<
std::endl;
    std::cin >> ox_ >> oy_ >> r_.x >> r_.y >> private_position_.x >>
private_position_.y >> private_mass_;
}
std::string Circle::classname() const{
    return name_;
}
unsigned Circle::size() const{
    return sizeof(*this);
}
Circle::~Circle() = default;
```

Interface.h

```
#pragma once

class IGeoFig{
public:
    //Area
    virtual double square() = 0;
    //Perimeter
    virtual double perimeter() = 0;
};//Interface "Geometrical figure"


class CVector2D{
public:
    CVector2D() : x(0), y(0) {}
    CVector2D(const double X, const double Y) : x(X), y(Y) {}
    ~CVector2D() = default;
    double x, y;
};//Vector

class IPhysObject{
public:
    //Mass, kg
    virtual double mass() const = 0;
    //Center  of mass coordinates
    virtual CVector2D position() = 0;
    //Mass comparison
    virtual bool operator== (const IPhysObject& ob) const = 0;
    virtual bool operator< (const IPhysObject& ob) const = 0;
};//Interface "Physical object"

class IPrintable{
public:
    //Show on the screen
    //(output object's parameters in text form
    virtual void draw() = 0;
};//Interface "Printable"

class IDialogInitiable{
public:
    virtual void initFromDialog() = 0;
};//Interface for objects tha could be defined by user

class BaseCObject{
public:
    //Class' name (data type)
    virtual std::string classname() const = 0;
    //Amount of used MEMORY
    virtual unsigned int size() const = 0;
};//Interface "Class"

class IAllTheInterfaces : public IGeoFig, public CVector2D, public IPhysObject,
public IPrintable, public IDialogInitiable, public BaseCObject {};
```

Main.cpp

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include "interface.h"
#include "Circle.h"
#include "Parallelogram.h"

int Commands()
{
    int number = 0;
    std::cout << "Command list: " << std::endl;
    std::cout << "press 1 to get the AREA." << std::endl;
    std::cout << "press 2 to get the PERIMETER." << std::endl;
    std::cout << "press 3 to get MASS." << std::endl;
    std::cout << "press 4 to get POSITION." << std::endl;
    std::cout << "press 5// Is this object EQUAL to other by MASS?" << std::endl;
    std::cout << "press 6// Is this object's MASS SMALLER than the other one's?." <<
std::endl;
    std::cout << "press 7 to DRAW the object." << std::endl;
    std::cout << "press 8 to get CLASSNAME of the object." << std::endl;
    std::cout << "press 9 to get SIZE of the object." << std::endl;
    std::cout << "press 10 for show all figures." << std::endl;
    std::cout << "press 11 to get SUM of SQUARES." << std::endl;
    std::cout << "press 12 to get SUM of PERIMETERS." << std::endl;
    std::cout << "press 13 to get the mass CENTER of the system." << std::endl;
    std::cout << "press 14 to get the SUM of MEMORY." << std::endl;
    std::cout << "press 15 for SORT objects by MASS and show their MASS." <<
std::endl;
    std::cout << "press 16 for Exit" << std::endl;
    std::cout << "press 0 to Continue" << std::endl;
    std::cin >> number;
    std::cout << std::endl;
    return number;
```

```
}

int main(){
    std::vector<IAllTheInterfaces*> figures;
    std::string figure_type;
    int i = 0;
    int key = 0;
    while (true){
        std::cout << "Please enter the type of figure you want to add
(Circle/Parallelogram); Enter 0 to stop:" << std::endl;
        std::cin >> figure_type;
        std::for_each(figure_type.begin(), figure_type.end(), [](char& c){
            c = tolower(c);
        });
        if (figure_type == "circle"){
            figures.push_back(new Circle);
        }
        else if (figure_type == "parallelogram"){
            figures.push_back(new Parallelogram);
        }
        else if (figure_type == "0"){
            break;
        }
        else{
            std::cout << "Wrong name/command, try again!" << std::endl;
            std::cin >> figure_type;
        }
        figures[i]->initFromDialog();
        i++;
    }
    key = Commands();
    while (true) {
        if (key == 1) {
            std::cout << "Square of this " << figures[i - 1]->classname() << " = " <<
figures[i - 1]->square() << std::endl;
            key = Commands();
        } else if (key == 2) {
            std::cout << "Perimeter of this " << figures[i - 1]->classname() << " = "
<< figures[i - 1]->perimeter() << std::endl;
            key = Commands();
        } else if (key == 3) {
            std::cout << "Mass of this " << figures[i - 1]->classname() << " = " <<
figures[i - 1]->mass() << std::endl;
            key = Commands();
        } else if (key == 4) {
            std::cout << "Position of this " << figures[i - 1]->classname() << " = {"
<< figures[i - 1]->position().x << ", " << figures[i - 1]->position().y << "}" <<
std::endl;
            key = Commands();
        } else if (key == 5) {
            std::cout << "Enter the index of another object: " << std::endl;
            int j = 0;
            std::cin >> j;
            if (figures[i - 1] == figures[j]) {
                std::cout << "These objects are equal by mass." << std::endl;
            } else {
                std::cout << "These objects are not equal by mass." << std::endl;
            }
            key = Commands();
        } else if (key == 6) {
            std::cout << "Enter the index of another object: " << std::endl;
            int j = 0;
            std::cin >> j;
            if (figures[i - 1] < figures[j]) {
                std::cout << "This object is smaller than another by mass." <<
std::endl;
            } else {
                std::cout << "This object is bigger than another by mass." <<
std::endl;
            }
            key = Commands();
        } else if (key == 7) {
            figures[i - 1]->draw();
            key = Commands();
        } else if (key == 8) {
            std::cout << figures[i - 1]->classname() << std::endl;
            key = Commands();
        } else if (key == 9) {
            std::cout << figures[i - 1]->size() << std::endl;
            key = Commands();
        } else if (key == 10) {
            for (auto& f : figures) {
                std::cout << f->classname() << std::endl;
                f->draw();
                std::cout << std::endl;
            }
            key = Commands();
        } else if (key == 11) {
            double sumSquares = 0;
            for (auto& f : figures) {
                sumSquares += f->square();
            }
            std::cout << "Sum of squares: " << sumSquares << std::endl;
            std::cout << std::endl;
            key = Commands();
        } else if (key == 12) {
            double sumPerimeters = 0;
            for (auto& f : figures) {
                sumPerimeters += f->perimeter();
            }
            std::cout << "Sum of squares: " << sumPerimeters << std::endl;
            std::cout << std::endl;
            key = Commands();
        } else if (key == 13) {
            CVector2D massCenter(0, 0);
            double massSum = 0;
            for (auto& f : figures) {
                massCenter.x += f->mass() * f->position().x;
                massCenter.y += f->mass() * f->position().y;
                massSum += f->mass();
            }
            massCenter.x = massCenter.x / massSum;
            massCenter.y = massCenter.y / massSum;
            std::cout << "Mass center: {" << massCenter.x << ", " << massCenter.y <<
"}" << std::endl;
            std::cout << std::endl;
            key = Commands();
        } else if (key == 14) {
            unsigned size = 0;
            for (auto& f : figures) {
                size += f->size();
            }
            std::cout << "Sum of memory: " << size << std::endl;
            std::cout << std::endl;
            key = Commands();
        } else if (key == 15) {
            std::sort(figures.begin(), figures.end(), [](IAllTheInterfaces* a,
IAllTheInterfaces* b) {
                return a->mass() < b->mass();
            });
            for (auto& f : figures) {
                std::cout << f->mass() << " ";
            }
            std::cout << std::endl;
            std::cout << std::endl;
            key = Commands();
        } else if (key == 16) {
            std::cout << "Exiting program.." << std::endl;
            exit(0);
        } else {
            break;
        }
    }
    return 0;
}
```

Ввод	Вывод
Круг	
Параллелограмм	

ЛР№4 Микаилова Микаила Аскеровича

Проверил преподаватель: Повышев Владислав Вячеславович