# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Объектно-ориентированное программирование» Тема: «Наследование»**

|  |  |
| --- | --- |
| Студент гр. 7303 | Мищенко М.А. |
| Преподаватель | Размочаева Н.В. |

Санкт-Петербург 2019

**Цель работы.**

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

**Задание.**

Необходимо спроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием). Задание предполагает использование виртуальных функций в иерархии наследования, проектирование и использование абстрактного базового класса. Разработанные классы должны быть наследниками абстрактного класса Shape, содержащего методы для перемещения в указанные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток.

Необходимо также обеспечить однозначную идентификацию каждого объекта.

Решение должно содержать:

1. Условие задания;
2. UML диаграмму разработанных классов;
3. Текстовое обоснование проектных решений;
4. Реализацию классов на языке С++.

**Индивидуализация.**

Вариант 10 – реализовать систему классов для фигур:

1. Прямоугольник;
2. Трапеция;
3. Равнобедренная трапеция;

**Обоснование проектных решений.**

Базовым классом для всех фигур является класс Shape, используемый для представления плоских геометрических фигур. Хранит в себе координаты центра фигуры, id фигуры, а также ее цвет.

Так же этот класс содержит в себе чисто виртуальные функции для перемещения в указанные координаты (move\_by\_coordinates), поворота на заданный угол (turn\_by\_corner), масштабирования на заданный коэффициент(inscrease\_by\_cf).

Были реализованы вспомогательные макросы:

#define Turn-положение точки при повороте на заданный угол относительно заданного центра.

#define Move- положение точки при перемещении относительно заданного центра.

#define Inscrease-увеличение сторон фигуры на заданный коэффициент

Класс Rectangle, используемый для представления прямоугольника, наследуется от Shape, содержит в себе поля для хранения координат вершин прямоугольника. Так же данный класс отличается от класса-родителя конструктором: данный класс

принимает всего 3 аргумента :

1) Высоту прямоугольника

2) Ширину прямоугольника

3) координаты центра прямоугольника

Класс Trapeze, используемый для представления трапеции, наследуется от Shape, содержит в себе поля для хранения координат вершин трапеции. Так же данный класс отличается от класса-родителя конструктором: данный класс

принимает всего 5 аргументов :

1) Высоту трапеции

2) Ширину 1-го основания

3) Ширину 2-го основания

4) Угол наклона оной из боковых сторон

5) координаты центра прямоугольника

Класс Lsosceles\_trapeze, используемый для представления равнобедренной трапеции, наследуется от Shape, содержит в себе поля для хранения координат вершин трапеции. Так же данный класс отличается от класса-родителя конструктором: данный класс

принимает всего 4 аргументов :

1) Высоту равнобедренной трапеции

2) Ширину 1-го основания

3) Ширину 2-го основания

4) координаты центра прямоугольника

**UML диаграмма разработанных классов.**

UML диаграмма разработанных классов представлена в приложении А и в соседнем документе (UML.png).

**Реализация классов на языке С++.**

Реализация классов представлена в приложении Б.

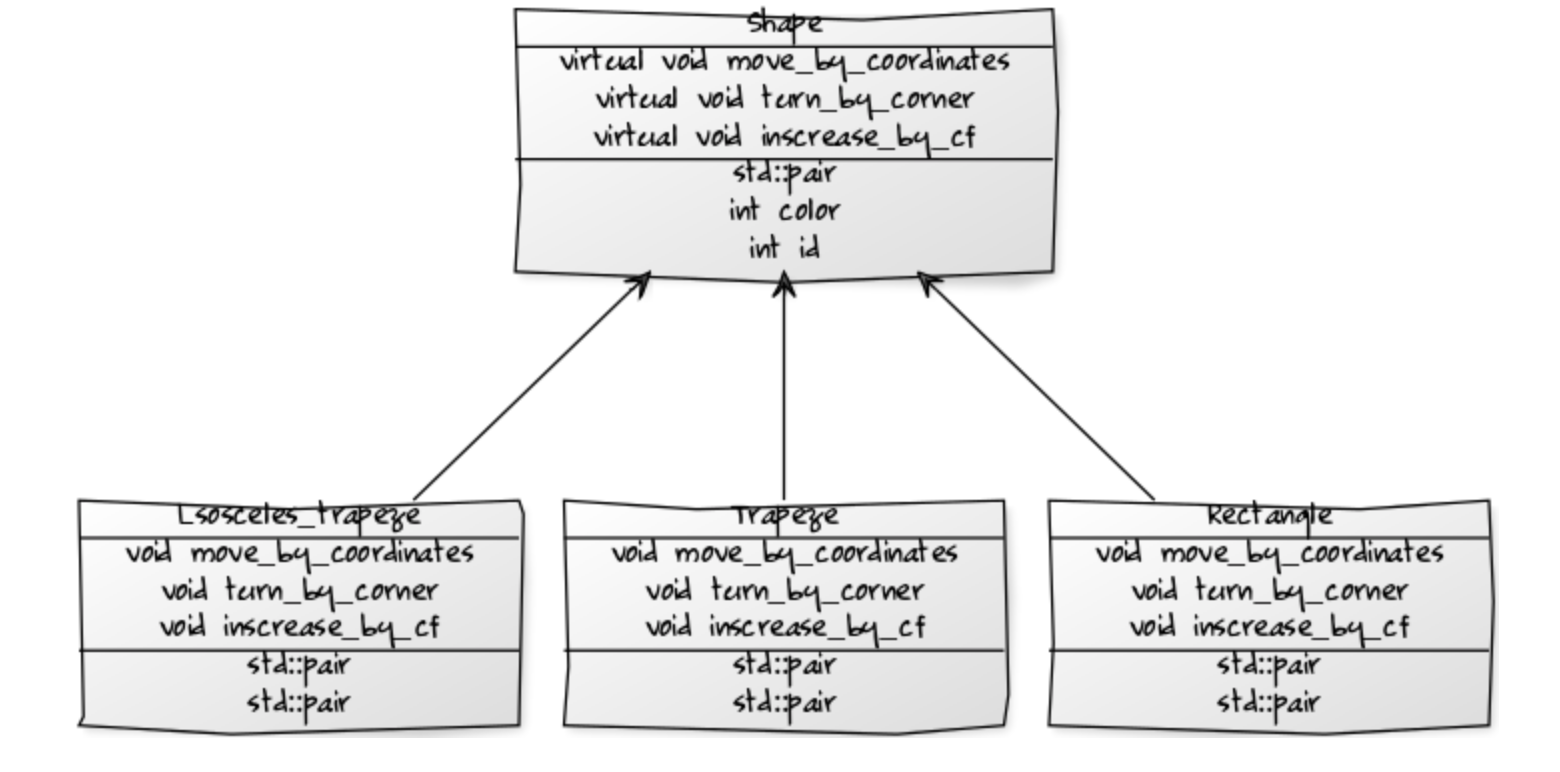
**Выводы.**

В ходе выполнения лабораторной работы была спроектирована система классов для работы с геометрическими фигурами в соответствии с индивидуальным заданием. В иерархии наследования были использованы виртуальные функции, базовый класс при этом является абстрактным(класс

называется абстрактны, если содержит хотя бы одну чисто виртуальную функцию). Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, была реализована однозначная идентификация объекта.

**ПРИЛОЖЕНИЕ А**

**UML ДИАГРАММА КЛАССОВ**

****

**ПРИЛОЖЕНИЕ Б**

**РЕАЛИЗАЦИЯ КЛАССОВ НА ЯЗЫКЕ C++**

**Shape.h**

**int** id1=0;

**class** Shape

{**public**:

Shape (){};

**virtual** ~Shape(){};

**virtual** **void** move\_by\_coordinates(**int** x,**int** y)=0;

**virtual** **void** turn\_by\_corner(**int** corner)=0;

**virtual** **void** inscrease\_by\_cf(**int** cf)=0;

std::pair<**int**,**int**>centr;

**int** color[3];

**int** **id**;

};

**class** Rectangle:**public** Shape

{ **public**:

Rectangle(**double** width,**double** height,std::pair<**int**,**int**>centr)

{ basis1=std::make\_pair(std::make\_pair(centr.first-(width/2),centr.second+(height/2)),

std::make\_pair(centr.first+(width/2),centr.second+(height/2)));

basis2=std::make\_pair(std::make\_pair(centr.first-(width/2),centr.second-(height/2)),

std::make\_pair(centr.first+(width/2),centr.second-(height/2)));

**this**->centr=centr;

**this**->**id**=id1;

id1++;

**this**->color[0]=rand()%255; **this**->color[1]=rand()%255; **this**->color[2]=rand()%255;

}

**void** move\_by\_coordinates(**int** x,**int** y);

**void** turn\_by\_corner(**int** corner);

**void** inscrease\_by\_cf(**int** cf);

**friend** std::ostream& **operator** <<(std::ostream &**out**,**const** Rectangle &rec);

~Rectangle(){};

**private**:

std::pair<std::pair<**double** ,**double** >, std::pair<**double**,**double** >> basis1;

std::pair<std::pair<**double** ,**double** > , std::pair<**double** ,**double** >> basis2;

};

**class** Trapeze:**public** Shape

{**public**:

Trapeze(**double** height,**double** size\_basis1,**double** size\_basis2,**double** corner1,std::pair<**int**,**int**>centr)

{

**if** (corner1==90){

basis1=std::make\_pair(std::make\_pair(centr.first-(size\_basis2/2),centr.second+(height/2)),

std::make\_pair(centr.first-(size\_basis2/2)+size\_basis1,centr.second+(height/2)));

basis2=std::make\_pair(std::make\_pair(centr.first-(size\_basis2/2),centr.second-(height/2)),

std::make\_pair(centr.first+(size\_basis2/2),centr.second-(height/2)));

}

**else**{

basis1=std::make\_pair(std::make\_pair(centr.first-(size\_basis2/2)+(height/tan((corner1\*Pi)/180)),

centr.second+height/2),

std::make\_pair(centr.first-(size\_basis2/2)+(height/tan((corner1\*Pi)/180))+size\_basis1,centr.second+(height/2)));

basis2=std::make\_pair(std::make\_pair(centr.first-(size\_basis2/2),centr.second-(height/2)),

std::make\_pair(centr.first+(size\_basis2/2),centr.second-(height/2)));}

**this**->**id**=id1;

id1++;

**this**->centr=centr;

**this**->color[0]=rand()%255; **this**->color[1]=rand()%255; **this**->color[2]=rand()%255;

}

**void** move\_by\_coordinates(**int** x,**int** y);

**void** turn\_by\_corner(**int** corner);

**void** inscrease\_by\_cf(**int** cf);

**friend** std::ostream& **operator** <<(std::ostream &**out**,**const** Trapeze &tr);

~Trapeze(){};

**private**:

std::pair<std::pair<**double** ,**double** >, std::pair<**double**,**double** >> basis1;

std::pair<std::pair<**double** ,**double** > , std::pair<**double** ,**double** >> basis2;

};

**class** Lsosceles\_trapeze:**public** Shape

{ **public**:

Lsosceles\_trapeze(**double** height,**double** size\_basis1,**double** size\_basis2,std::pair<**int**,**int**>centr)

{

basis1=std::make\_pair(std::make\_pair(centr.first-(size\_basis1/2),centr.second+(height/2)),

std::make\_pair(centr.first+(size\_basis1/2),centr.second+(height/2)));

basis2=std::make\_pair(std::make\_pair(centr.first-(size\_basis2/2),centr.second-(height/2)),

std::make\_pair(centr.first+(size\_basis2/2),centr.second-(height/2)));

**this**->**id**=id1;

id1++;

**this**->centr=centr;

**this**->color[0]=rand()%255; **this**->color[1]=rand()%255; **this**->color[2]=rand()%255;

}

**void** move\_by\_coordinates(**int** x,**int** y);

**void** turn\_by\_corner(**int** corner);

**void** inscrease\_by\_cf(**int** cf);

**friend** std::ostream& **operator** <<(std::ostream &**out**,**const** Lsosceles\_trapeze &Lr\_tr);

~ Lsosceles\_trapeze(){};

**private**:

std::pair<std::pair<**double** ,**double** >, std::pair<**double** ,**double**>> basis1;

std::pair<std::pair<**double** ,**double** > , std::pair<**double** ,**double** >> basis2;

};

**Main.cpp**

#define Pi 3.14

#include<iostream>

#include<cmath>

#include"Shape.h"

#include<iomanip>

#define Turn(corn,x,y,cen\_x,cen\_y) std::make\_pair(((x-cen\_x)\*cos((corn\*Pi)/180)-(y-cen\_y)\*sin((corn\*Pi)/180))+cen\_x,((x-cen\_x)\*sin((corn\*Pi)/180)+(y-cen\_y)\*cos((corn\*Pi)/180))+cen\_y)

#define Move(x,y,cen\_x,cen\_y,new\_pos\_x,new\_pos\_y) std::make\_pair(x-cen\_x+new\_pos\_x,y-cen\_y+new\_pos\_y)

#define Inscrease(x,y,cen\_x,cen\_y,cf) std::make\_pair((x-cen\_x)\*cf+cen\_x,(y-cen\_y)\*cf+cen\_y)

**void** Rectangle::move\_by\_coordinates(**int** x,**int** y)

{ basis1=std::make\_pair(Move(basis1.first.first,basis1.first.second,centr.first,centr.second,x,y), Move(basis1.second.first,basis1.second.second,centr.first,centr.second,x,y));

basis2=std::make\_pair(Move(basis2.first.first,basis2.first.second,centr.first,centr.second,x,y), Move(basis2.second.first,basis2.second.second,centr.first,centr.second,x,y));

centr=std::make\_pair(x,y);

}

**void** Trapeze::move\_by\_coordinates(**int** x,**int** y)

{ basis1=std::make\_pair(Move(basis1.first.first,basis1.first.second,centr.first,centr.second,x,y), Move(basis1.second.first,basis1.second.second,centr.first,centr.second,x,y));

basis2=std::make\_pair(Move(basis2.first.first,basis2.first.second,centr.first,centr.second,x,y), Move(basis2.second.first,basis2.second.second,centr.first,centr.second,x,y));

centr=std::make\_pair(x,y);

}

**void** Lsosceles\_trapeze::move\_by\_coordinates(**int** x,**int** y)

{ basis1=std::make\_pair(Move(basis1.first.first,basis1.first.second,centr.first,centr.second,x,y), Move(basis1.second.first,basis1.second.second,centr.first,centr.second,x,y));

basis2=std::make\_pair(Move(basis2.first.first,basis2.first.second,centr.first,centr.second,x,y), Move(basis2.second.first,basis2.second.second,centr.first,centr.second,x,y));

centr=std::make\_pair(x,y);

}

**void** Rectangle::turn\_by\_corner(**int** corner)

{

basis1=std::make\_pair(Turn(corner,basis1.first.first,basis1.first.second,centr.first,centr.second), Turn(corner,basis1.second.first,basis1.second.second,centr.first,centr.second));

basis2=std::make\_pair(Turn(corner,basis2.first.first,basis2.first.second,centr.first,centr.second), Turn(corner,basis2.second.first,basis2.second.second,centr.first,centr.second));

}

**void** Trapeze::turn\_by\_corner(**int** corner)

{

basis1=std::make\_pair(Turn(corner,basis1.first.first,basis1.first.second,centr.first,centr.second), Turn(corner,basis1.second.first,basis1.second.second,centr.first,centr.second));

basis2=std::make\_pair(Turn(corner,basis2.first.first,basis2.first.second,centr.first,centr.second), Turn(corner,basis2.second.first,basis2.second.second,centr.first,centr.second));

}

**void** Lsosceles\_trapeze::turn\_by\_corner(**int** corner)

{

basis1=std::make\_pair(Turn(corner,basis1.first.first,basis1.first.second,centr.first,centr.second), Turn(corner,basis1.second.first,basis1.second.second,centr.first,centr.second));

basis2=std::make\_pair(Turn(corner,basis2.first.first,basis2.first.second,centr.first,centr.second), Turn(corner,basis2.second.first,basis2.second.second,centr.first,centr.second));

}

**void** Rectangle::inscrease\_by\_cf(**int** cf)

{ basis1=std::make\_pair(Inscrease(basis1.first.first,basis1.first.second,centr.first,centr.second,cf), Inscrease(basis1.second.first,basis1.second.second,centr.first,centr.second,cf));

basis2=std::make\_pair(Inscrease(basis2.first.first,basis2.first.second,centr.first,centr.second,cf), Inscrease(basis2.second.first,basis2.second.second,centr.first,centr.second,cf));

}

**void** Trapeze::inscrease\_by\_cf(**int** cf)

{ basis1=std::make\_pair(Inscrease(basis1.first.first,basis1.first.second,centr.first,centr.second,cf), Inscrease(basis1.second.first,basis1.second.second,centr.first,centr.second,cf));

basis2=std::make\_pair(Inscrease(basis2.first.first,basis2.first.second,centr.first,centr.second,cf), Inscrease(basis2.second.first,basis2.second.second,centr.first,centr.second,cf));

}

**void** Lsosceles\_trapeze::inscrease\_by\_cf(**int** cf)

{ basis1=std::make\_pair(Inscrease(basis1.first.first,basis1.first.second,centr.first,centr.second,cf), Inscrease(basis1.second.first,basis1.second.second,centr.first,centr.second,cf));

basis2=std::make\_pair(Inscrease(basis2.first.first,basis2.first.second,centr.first,centr.second,cf), Inscrease(basis2.second.first,basis2.second.second,centr.first,centr.second,cf));

}

std::ostream& **operator** <<(std::ostream &out,**const** Rectangle &rec)

{

out<<"Centr("<<std::fixed<<std::setprecision(1)<<rec.centr.first<<","<<std::fixed<<std::setprecision(1)<<rec.centr.second<<")\n"<<"Id="

<<rec.id<<"\n"<<"Color="<<rec.color[0]<<"."<<rec.color[1]<<"."<<rec.color[2]<<"\n"

<<"Coordinat(("<<std::fixed<<std::setprecision(1)<<rec.basis1.first.first<<","

<<std::fixed<<std::setprecision(1)<<rec.basis1.first.second<<")("<<std::fixed<<std::setprecision(1)

<<rec.basis1.second.first<<","<<std::fixed<<std::setprecision(1)<<rec.basis1.second.second<<"("

<<std::fixed<<std::setprecision(1)<<rec.basis2.first.first<<","<<std::fixed<<std::setprecision(1)<<rec.basis2.first.second<<")("<<std::fixed<<std::setprecision(1)

<<rec.basis2.second.first<<","<<std::fixed<<std::setprecision(1)<<rec.basis2.second.second<<"))\n";

**return** out;

}

std::ostream& **operator** <<(std::ostream &out,**const** Trapeze &tr)

{

out<<"Centr("<<std::fixed<<std::setprecision(1)<<tr.centr.first<<","<<std::fixed<<std::setprecision(1)<<tr.centr.second<<")\n"<<"Id="

<<tr.id<<"\n"<<"Color="<<tr.color[0]<<"."<<tr.color[1]<<"."<<tr.color[2]<<"\n"

<<"Coordinat(("<<std::fixed<<std::setprecision(1)<<tr.basis1.first.first<<","<<std::fixed<<std::setprecision(1)<<tr.basis1.first.second<<")("<<std::fixed<<std::setprecision(1)

<<tr.basis1.second.first<<","<<std::fixed<<std::setprecision(1)<<tr.basis1.second.second<<")("<<std::fixed<<std::setprecision(1)<<tr.basis2.first.first<<","<<std::fixed<<std::setprecision(1)<<tr.basis2.first.second<<")("<<std::fixed<<std::setprecision(1)

<<tr.basis2.second.first<<","<<std::fixed<<std::setprecision(1)<<tr.basis2.second.second<<"))\n";

**return** out;

}

std::ostream& **operator** <<(std::ostream &out,**const** Lsosceles\_trapeze &Lr\_tr)

{

out<<"Centr("<<std::fixed<<std::setprecision(1)<<Lr\_tr.centr.first<<","<<std::fixed<<std::setprecision(1)<<Lr\_tr.centr.second<<")\n"<<"Id="

<<Lr\_tr.id<<"\n"<<"Color="<<Lr\_tr.color[0]<<"."<<Lr\_tr.color[1]<<"."<<Lr\_tr.color[2]<<"\n"

<<"Coordinat(("<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis1.first.first<<","<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis1.first.second<<")("<<std::fixed<<std::setprecision(1)

<<Lr\_tr.basis1.second.first<<","<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis1.second.second<<")("<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis2.first.first<<","<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis2.first.second<<")("<<std::fixed<<std::setprecision(1)

<<Lr\_tr.basis2.second.first<<","<<std::fixed<<std::setprecision(1)<<Lr\_tr.basis2.second.second<<"))\n";

**return** out;

}

**int** main()

{

Rectangle rec(6,2,std::make\_pair(0,0));

Lsosceles\_trapeze l\_tr(5.0,8,4,std::make\_pair(0,0));

Trapeze tr(6,4,8,60,std::make\_pair(0,0));

rec.move\_by\_coordinates(2, 2);

l\_tr.move\_by\_coordinates(2, 2);

tr.move\_by\_coordinates(2, 2);

rec.turn\_by\_corner(90);

l\_tr.turn\_by\_corner(90);

tr.turn\_by\_corner(90);

rec.inscrease\_by\_cf(2);

l\_tr.inscrease\_by\_cf(2);

tr.inscrease\_by\_cf(2);

std::cout<<rec;

std::cout<<l\_tr;

std::cout<<tr;

**return** 0;

}