I. Базовый класс для всех вариантов:

class Figure

{

int c; // цвет

bool visible;

protected:

int x,y; // базовая точка

virtual void draw() const;

public:

Figure(int c, int x, int y);

~Figure();

void move(int dx, int dy); // сместить фигуру на (dx,dy) – только видимую

void setBorderColor(int c); // установить цвет фигуры – только видимой

int getBorderColor() const; // получить цвет

void setVisible( bool isVisible = true ); // показать/спрятать фигуру

bool isVisible() const; // признак видимости

virtual void calcParams( float& perimeter, float& area ) const;

// вычислить периметр и площадь фигуры

};

II. Выполнить:

1. Реализацию методов класса Figure.
2. Скорректировать интерфейс Figure, чтобы методы calcParams и draw стали чисто виртуальными.
3. Определить производный класс (в соответствии с вариантом задания).
4. Определить дополнительный метод в производном классе для изменения размеров:  
                 void setSizes(длина, толщина полосок);
5. От написанного класса произвести новый дочерний класс - закрашенная фигура.  
   Например, закрашенный крест (FilledCross ← Cross ← Figure).
6. Добавить к параметрам конструктора нового дочернего класса цвет заполнения.
7. Определить дополнительный метод у нового дочернего класса для изменения цвета заполнения:

       void setFillColor(int c);

1. Переопределить у нового дочернего класса метод изменения цвета границы:

void setBorderColor(int c); (если новый цвет границы совпадает c текущим цветом заполнения, то цвет границы не изменять)

1. Правильно реализовать деструкторы в родительских и всех производных классах (при необходимости скорректировать интерфейс базового класса).
2. При тестировании объектов классов продемонстрировать полиморфизм (при необходимости скорректировать интерфейс базового класса).
3. При тестировании динамически создать две фигуры 2 разных классов, вызвать все методы.

Вариант 18.Крест.

Cross(цвет линий, x и y центра, длина, толщина полосок);

**Листинг описания классов:**

#pragma once

#include <iostream>

using namespace std;

class Figure

{

int c; //цвет

bool visible;

protected:

int x, y; // базовая точка

virtual void draw() const = 0;

public:

Figure(int c, int x, int y) {this->c = c; this->x = x; this->y = y; visible = false;}

virtual ~Figure() {cout<<endl<<"delete object Figure"<<endl;}

void move(int dx, int dy); //сместить фигуру на (dx, dy) - только видимую

void setBorderColor(int c); //установить цвет фигуры - только видимой

int getBorderColor() const; // получить цвет

void setVisible(bool isVisible = true); //показать/спрятать фигуру

bool isVisible() const; // признак видимости

virtual void calcParams(float& perimeter, float& area) const = 0;

// вычислить периметр и площадь фигуры

};

class Cross: public Figure //(цвет линий, х и у центра, длина, толщина полосок)

{

public:

float lengthLines;

float thicknessLines;

Cross(int c, int x, int y, float lengthLines, float thicknessLines) : Figure(c, x, y)

{

setSizes(lengthLines, thicknessLines);

}

~Cross() {cout<<endl<<"delete object Cross"<<endl;}

void setSizes(float lengthLines, float thicknessLines);

void draw() const;

void calcParams(float& perimeter, float& area) const;

};

class FilledCross: public Cross

{

public:

int fillColor;

FilledCross(int c, int x, int y, int lengthLines, int thicknessLines, int fillColor) : Cross(c, x, y, lengthLines, thicknessLines)

{

this->fillColor = fillColor;

}

~FilledCross() {cout<<endl<<"delete object FilledCross"<<endl;}

void setFillColor(int c);

void setBorderColor(int c);

};

**Листинг реализации классов**

#include <iostream>

#include "classFigureCross.h"

#include <cmath>

void Figure::move(int dx, int dy)

{

if (visible)

{

x = x + dx;

y = y + dy;

draw();

}

}

void Figure::setBorderColor(int c)

{

if (visible)

{

this->c = c;

draw();

}

}

void Figure::setVisible(bool isVisible)

{

bool old\_visible = visible; //memorizing the visibility status of the figure

visible = isVisible;

if (visible && !old\_visible) //If the figure was not visible

{

draw();

}

}

int Figure::getBorderColor() const

{

return c;

}

bool Figure::isVisible() const

{

return visible;

}

void Cross::draw() const

{

cout<<"Figure - Cross"<<endl;

cout<<"x, y: "<<x<<","<<y<<endl;

cout<<"BorderColor: "<<getBorderColor()<<endl;

cout<<"lengthLines "<<lengthLines<<endl;

cout<<"thicknessLines "<<thicknessLines<<endl;

}

void Cross::calcParams(float& perimeter, float& area) const

{

perimeter = ((lengthLines/2. - thicknessLines/2.)\*8) + thicknessLines \* 4;

area = pow((thicknessLines\*(lengthLines/2. - thicknessLines/2.)),4);

}

void Cross::setSizes(float lengthLines, float thicknessLines)

{

this->lengthLines = lengthLines;

this->thicknessLines = thicknessLines;

if(isVisible())

{

draw();

}

}

void FilledCross::draw() const

{

cout<<"Figure - FilledCross"<<endl;

cout<<"x, y: "<<x<<","<<y<<endl;

cout<<"BorderColor: "<<getBorderColor()<<endl;

cout<<"lengthLines "<<lengthLines<<endl;

cout<<"thicknessLines "<<thicknessLines<<endl;

cout<<"fillColor "<<fillColor<<endl;

}

void FilledCross::setFillColor(int c)

{

fillColor = c;

if(isVisible())

{

draw();

}

}

void FilledCross::setBorderColor(int c)

{

if(fillColor != c)

{

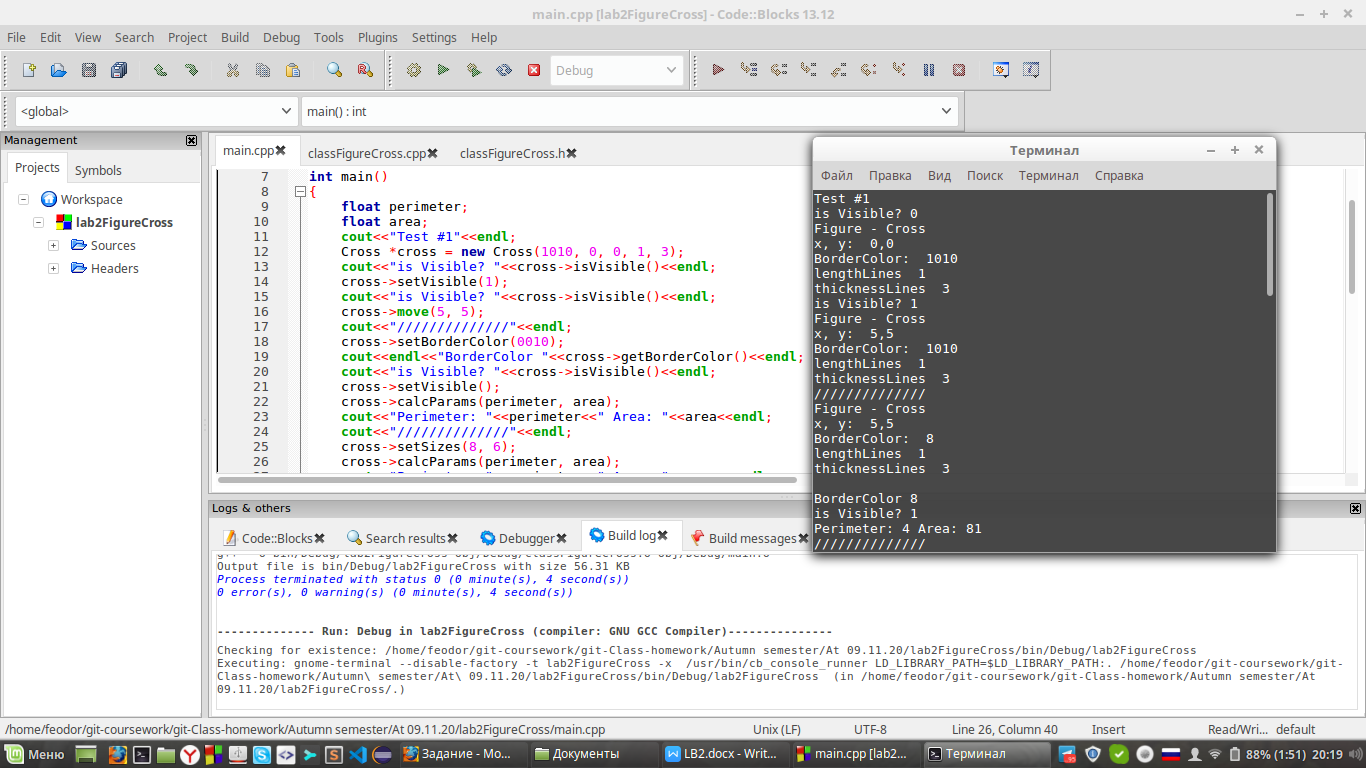
Figure::setBorderColor(c);

}

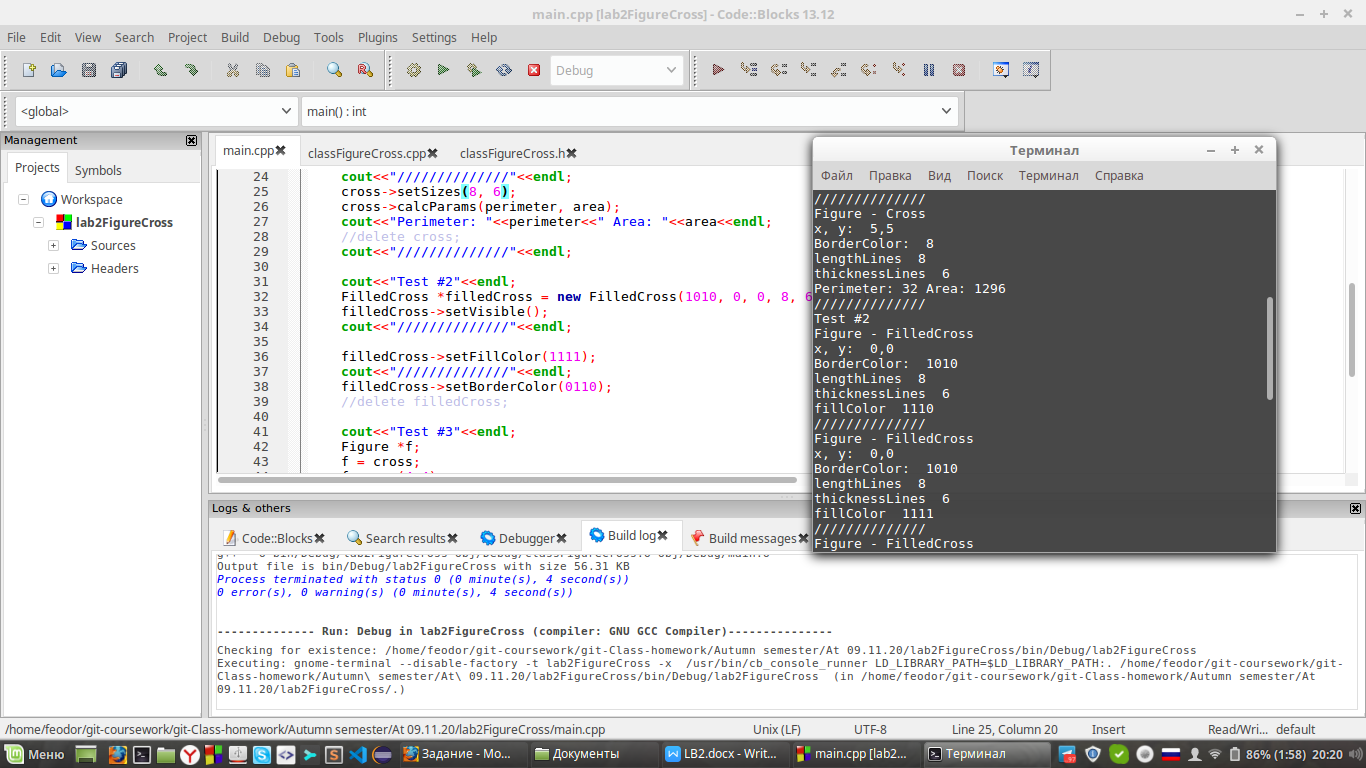
}

**Результаты работы программы проверки классов**

На данном скриншоте я проверял работу программы на создание объекта класса Cross с соответствующими ему методами (isVisible, setVisible, move, setBorderColor, getBorderColor, calcParams - все методы без ошибок сработали).



Проверив еще раз методы setSizes, calcParams для класса Cross (путем изменения ширины и длины креста) я стал тестировать на работоспособность класс FilledCross создав объект filledCross с соответствующими методами для этого класса. Далее проверил на работоспособность переопределенный в этом классе метод setBorderColor и новый метод setFillColor. Все сработало без ошибок.



В конце тестирования был продемонстрирован полиморфизм при помощи строчек кода начиная с 42 и заканчивая 50. Соответствующие результаты были выведены на экран. Программа сработала без ошибок. После тестирования так же была продемонстрирована работа деструкторов (delete object FilledCross, Cross, Figure).

