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.

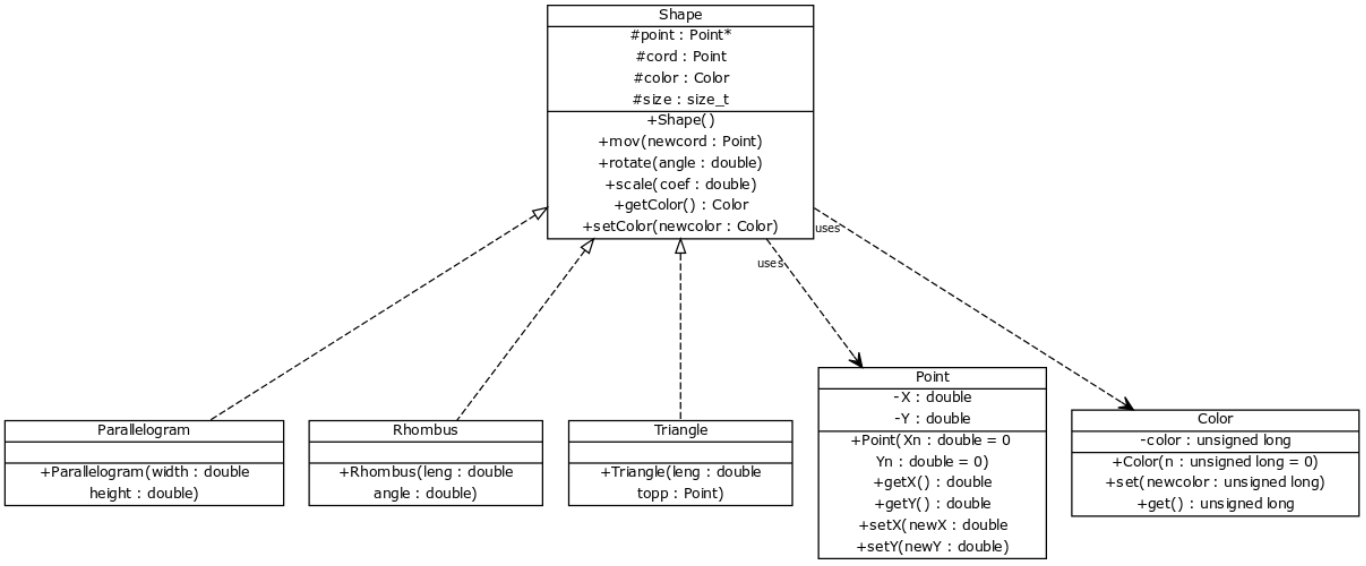
Решение должно содержать:

условие задания;
UML диаграмму разработанных классов;
текстовое обоснование проектных решений;
реализацию классов на языке C++.

Индивидальное задание

Треугольник, ромб, параллелограмм

UML диаграмма классов



Реализация классов на C++

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <vector>

class Color {
 private:
  unsigned long color;
 public:

  Color(unsigned long n = 0)
      : color(n) {
  }

  void set(unsigned long newcolor) {
    color = newcolor;
  }

  unsigned long get() {
    return color;
  }

  friend std::ostream& operator<<(std::ostream& os, const Color& tr) {
      os << "#" << std::setw(6) <<std::setfill('0') << std::hex << tr.color;
    return os;
  }
};


class Point {
 private:
  double X, Y;

 public:
  Point(double Xn = 0, double Yn = 0)
      : X(Xn)
      , Y(Yn)
  {
  }

  Point operator+(Point const& t) const;

  double getX() {
    return X;
  }

  double getY() {
```

```
    return Y;
  }

  void setX(double a) {
    X = a;
  }

  void setY(double a) {
    Y = a;
  }

  friend std::ostream& operator<<(std::ostream& os, const Point& tr) {
    os << "(" << tr.X << "," << tr.Y << ")";
    return os;
  }
};

Point Point::operator+(Point const& t) const {
  Point a(X+t.X, Y+t.Y);
  return a;
}

class Shape {
 public:
  Shape()
      : color(Color(0)) {
  }

  void move(Point newcord) {
    cord = newcord;
  }

  void rotate(double angle);

  void scale(double coef);

  Color getColor() {
    return color;
  }

  void setColor(Color newcolour) {
    color = newcolour;
  }

 protected:
  std::vector<Point> point;
  Point cord;
  Color color;
};
```

```
void Shape::scale(double coef) {
  for (int i = 0; i < point.size(); i++) {
    point[i].setX(point[i].getX()*coef);
    point[i].setY(point[i].getY()*coef);
  }
}

void Shape::rotate(double angle) {
  for (int i = 0; i < point.size(); i++) {
    point[i].setX(point[i].getX()*cos(angle)
                  - point[i].getY()*sin(angle));
    point[i].setY(point[i].getX()*sin(angle)
                  + point[i].getY()*cos(angle));
  }
}


class Triangle: public Shape {
 public:
  Triangle(double leng, Point topp) {
    cord = Point(0, 0);
    point.push_back(Point(leng, 0));
    point.push_back(topp);
  }


  friend std::ostream& operator<<(std::ostream& os, const Triangle& tr) {
    os << "Triangle(" << tr.cord << ";"
       << tr.cord+tr.point[0] << ";" << tr.cord+tr.point[1]
       << "):" << tr.color;
    return os;
  }
};

class Rhombus: public Shape {
 public:
  Rhombus(double leng, double angle);

  friend std::ostream& operator<<(std::ostream& os, const Rhombus& tr) {
    os << "Rhombus(" << tr.cord << ";"
       << tr.cord+tr.point[0] << ";" << tr.cord+tr.point[1]
       << ";" << tr.cord+tr.point[2] << "):" << tr.color;
    return os;
  }
};

Rhombus::Rhombus(double leng, double angle) {
  cord = Point(0, 0);
```

```
  point.push_back(Point(leng, 0));
  point.push_back(Point(leng*(cos(angle)+1), leng*sin(angle)));
  point.push_back(Point(leng*cos(angle), leng*sin(angle)));
}


class Parallelogram: public Shape {
 public:
  Parallelogram(double width, double height);

  friend std::ostream& operator<<(std::ostream& os, const Parallelogram& tr) {
    os << "Parallelogram(" << tr.cord << ";"
       << tr.cord+tr.point[0] << ";" << tr.cord+tr.point[1]
       << ";" << tr.cord+tr.point[2] << "):" << tr.color;
    return os;
  }
};


Parallelogram::Parallelogram(double width, double height) {
  cord = Point(0, 0);
  point.push_back(Point(width, 0));
  point.push_back(Point(width, height));
  point.push_back(Point(0, height));
}
```