## СЕМИНАР 8. Композитные классы

Композитным называется класс, содержащий в себе в качестве информационного члена (одного или нескольких) объект абстрактного типа. Иными словами композиция — это построение сложного объекта из более простых, часть которых относятся к пользовательскому типу данных. Например, организм состоит из органов или автомобиль — из двигателя, шасси и других элементов.

Композиция реализует отношение «СОДЕРЖИТ» (целое содержит часть). Для реализации композиции объект и часть должны иметь следующие отношения:

● часть (член) является составляющей объекта (класса);

● часть (член) может принадлежать только одному объекту (классу) в каждом конкретном случае;

● часть (член) существует под управлением объекта (класса);

● часть (член) не знает о существовании объекта (класса).

Композиция – это разновидность агрегации, так называемое сильное агрегирование. **Разница** между **композицией** и **агрегацией** заключается в том, что в случае **композиции** целое явно контролирует время жизни своей составной части (часть не существует без целого), а в случае **агрегации** целое хоть и содержит свою составную часть, время их жизни не связано (например, составная часть передается через параметры конструктора). В отличие от композиции, при агрегации части могут принадлежать более чем одному целому, и оно не управляет существованием и временем жизни частей. При создании и уничтожении агрегации целое не несет ответственности за создание и уничтожение своих частей.

При создании объектов композитных классов должны вызываться конструкторы со списком инициализации. Исключением является ситуация, когда классы подобъектов имеют только конструкторы без аргументов. В этом случае конструктор класса может не содержать списка инициализации или в нем может быть указана только инициализация собственных компонентных данных класса. Конструкторы подобъектов при этом перечисляются в списке инициализации через запятую после знака двоеточия. Порядок вызова конструкторов подобъектов должен совпадать с порядком объявления в композитном классе.

Аргументы конструкторам подобъектов композитного класса передаются через аргументы конструктора композитного класса.

Сперва всегда вызываются конструкторы подобъектов, затем конструктор самого композитного класса.

Конструкторы подобъектов в композитном классе вызываются в порядке объявления их в этом композитном классе. Деструкторы вызываются в обратном порядке.

С деструкторами все наоборот: сперва вызывается деструктор композитного класса, затем деструкторы подобъектов в последовательности противоположной вызову конструкторов. Деструкторы подобъектов вызываются при вызове деструктора композитного класса.

Важной особенностью композитных классов является то, что они не имею прав доступа к приватным полям своих подобъектов. Композитные классы, также как и внешние функции, имеют права доступа к публичным полям подобъекта. Чтобы запретить доступ внешних функций к публичным полям подобъекта их необходимо объявить в приватной области композитного класса.

Если композитному классу необходимо осуществить доступ к приватным полям подобъекта, его следует объявить дружественным классу подобъекта

Программа для подготовки к лабораторной работе.

Разработать ООП для поиска точек Сх и Сy осей координат, которые находятся на минимальном расстоянии от пары заданных точек A и B плоскости XY. Координаты пары заданных точек A и B должны передаваться программе через поток стандартного ввода. Искомые точки Сх и Сy образуют пересечения с осями координат либо отрезка AB, либо отрезка с симметричным отражением точки A или B относительно оси, которую отрезок AB не пересекает. Координаты полученных точек Сх и Сy должны отображаться через поток стандартного вывода. Программная реализация этих геометрических построений должна быть основана на разработке композитного класса отрезка прямой. Его приватные данные должны включать подобъекты класса точки для концов отрезка, а публичные методы должны обеспечивать построение точек пересечения отрезка с осями координат. Конструкторы классов отрезка и точки должны использовать списки инициализации своих приватных данных. Необходимо также предусмотреть перегрузку операторов ввода-вывода точек и обеспечить доступ к их приватным координатам.



**Комментарии к рисунку и заданию:**

На рисунке приведен пример поиска точек Cx и Cy. Кратчайшим расстоянием между двумя точками является соединяющая их прямая. Следовательно, если отрезок АВ пересекает одну из осей, точку пересечения можно принять за точку, находящуюся на минимальное расстоянии от А и В. Отрезок АВ пересекает ось Y, следовательно точка Cy – точка их пересечения. Отрезок АВ не пересекает ось Х, значит необходимо отразить точку А или В относительно этой оси, изменив знак отражаемой координаты на противоположный. На рисунке точка В отражена в точку В’, так что теперь отрезок АВ’ пересекает ось Х, а точка Сх – точка их пересечения.

**Нахождение точки пересечения прямой и очей координат:**

Координаты точек пересечения прямой с осями координат найдем из записи уравнения прямой, проходящей через точки и B, которое имеет вид:

В точке пересечения прямой с осью координат X, значение координаты *y* будет равно 0, а координату *х* можно выразить из исходного уравнения, подставив в него y=0.

Аналогично, в точке пересечения прямой с осью координат Y, значение координаты *x* будет равно 0, а координату *y* можно выразить из исходного уравнения, подставив в него *x*=0.

1: #include <iostream>

2: using namespace std;

3:

4: class Line;

5: class Dot

6: {

7: friend class Line;

8: friend istream& operator>>(istream&, Dot&);

9: friend ostream& operator<<(ostream&, Dot&);

10: private:

11: float x;

12: float y;

13: public:

14: Dot(float xx=0.0, float yy=0.0) : x(xx), y(yy) { };

15: float getx() { return x; };

16: float gety() { return y; };

17: Dot symx() { return Dot(-x, y); };

18: Dot symy() { return Dot(x, -y); };

19: };

20:

21: class Line

22: {

23: private:

24: Dot p1;

25: Dot p2;

26: public:

27: Line(Dot& p, Dot& q) : p1(p), p2(q) {};

28: Line(float x1, float y1, float x2, float y2) : p1(x1,y1), p2(x2,y2) {}

29: Dot clipx();

30: Dot clipy();

31: };

32:

33: Dot Line::clipx()

34: {

35: float dx = p2.x - p1.x;

36: float dy = p1.y - p2.y;

37: float x0 = (p2.y \* dx)/dy + p2.x;

38: return Dot(x0, 0.0);

39: }

40: Dot Line::clipy()

41: {

42: float dx = p2.x - p1.x;

43: float dy = p1.y - p2.y;

44: float y0 = (p2.x \* dy)/dx + p2.y;

45: return Dot(0.0, y0);

46: }

47:

48: istream& operator>>(istream& input, Dot& p)

49: {

50: char semicolon=';';

51: input >> p.x >> semicolon >> p.y;

52: return input;

53: }

54:

55: ostream& operator<<(ostream& output, Dot& p)

56: {

57: char semicolon = ';';

58: output << p.x << semicolon << p.y;

59: return output;

60: }

61:

62: int main()

63: {

64: Dot A, B, \_A, \_B, C;

65: float xa, ya, xb, yb, \_x, \_y;

66: cout << "Input xa;ya xb;yb or exit" << endl;

67: while(cin >> A >> B)

68: {

69: ya = A.gety();

70: yb = B.gety();

71: \_A = (ya \* yb) > 0 ? A.symy() : A;

72: Line lx(\_A, B);

73: C = lx.clipx();

74: cout << C << endl;

75: xa = A.getx();

76: xb = B.getx();

77: \_x = (xa\*xb > 0) ? -xb : xb;

78: Line ly(xa, ya, \_x, yb);

79: C = ly.clipy();

80: cout << C << endl;

81: cout << "Input xa;ya xb;yb or exit" << endl;

82: }

83: return(0);

84: }